AUTORIZACIÓN PARA LA DIGITALIZACIÓN, DEPÓSITO Y DIVULGACIÓN EN RED DE PROYECTOS FIN DE GRADO, FIN DE MÁSTER, TESINAS O MEMORIAS DE BACHILLERATO

1º. Declaración de la autoría y acreditación de la misma.

El autor D. Ignacio Aguirre Panadero

DECLARA ser el titular de los derechos de propiedad intelectual de la obra: ELECTRIC ENERGY CONSUMER CHARACTERIZATION, CLASSIFICATION...,

que ésta es una obra original, y que ostenta la condición de autor en el sentido que otorga la Ley de Propiedad Intelectual.

2º. Objeto y fines de la cesión.

Con el fin de dar la máxima difusión a la obra citada a través del Repositorio institucional de la Universidad, el autor **CEDE** a la Universidad Pontificia Comillas, de forma gratuita y no exclusiva, por el máximo plazo legal y con ámbito universal, los derechos de digitalización, de archivo, de reproducción, de distribución y de comunicación pública, incluido el derecho de puesta a disposición electrónica, tal y como se describen en la Ley de Propiedad Intelectual. El derecho de transformación se cede a los únicos efectos de lo dispuesto en la letra a) del apartado siguiente.

3º. Condiciones de la cesión y acceso

Sin perjuicio de la titularidad de la obra, que sigue correspondiendo a su autor, la cesión de derechos contemplada en esta licencia habilita para:

a) Transformarla con el fin de adaptarla a cualquier tecnología que permita incorporarla a internet y hacerla accesible; incorporar metadatos para realizar el registro de la obra e incorporar "marcas de agua" o cualquier otro sistema de seguridad o de protección.
b) Reproducirla en un soporte digital para su incorporación a una base de datos electrónica, incluyendo el derecho de reproducir y almacenar la obra en servidores, a los efectos de garantizar su seguridad, conservación y preservar el formato.
c) Comunicarla, por defecto, a través de un archivo institucional abierto, accesible de modo libre y gratuito a través de internet.
d) Cualquier otra forma de acceso (restringido, embargado, cerrado) deberá solicitarse expresamente y obedecer a causas justificadas.
e) Asignar por defecto a estos trabajos una licencia Creative Commons.
f) Asignar por defecto a estos trabajos un HANDLE (URL *persistente*).

4º. Derechos del autor.

El autor, en tanto que titular de una obra tiene derecho a:

a) Que la Universidad identifique claramente su nombre como autor de la misma
b) Comunicar y dar publicidad a la obra en la versión que ceda y en otras posteriores a través de cualquier medio.
c) Solicitar la retirada de la obra del repositorio por causa justificada.
d) Recibir notificación fehaciente de cualquier reclamación que puedan formular terceras personas en relación con la obra y, en particular, de reclamaciones relativas a los derechos de propiedad intelectual sobre ella.

5º. Deberes del autor.

El autor se compromete a:

a) Garantizar que el compromiso que adquiere mediante el presente escrito no infringe ningún derecho de terceros, ya sean de propiedad industrial, intelectual o cualquier otro.
b) Garantizar que el contenido de las obras no atenta contra los derechos al honor, a la intimidad y a la imagen de terceros.
c) Asumir toda reclamación o responsabilidad, incluyendo las indemnizaciones por daños, que pudieran ejercitarse contra la Universidad por terceros que vieran infringidos sus derechos e

intereses a causa de la cesión.

d) Asumir la responsabilidad en el caso de que las instituciones fueran condenadas por infracción de derechos derivada de las obras objeto de la cesión.

6º. Fines y funcionamiento del Repositorio Institucional.

La obra se pondrá a disposición de los usuarios para que hagan de ella un uso justo y respetuoso con los derechos del autor, según lo permitido por la legislación aplicable, y con fines de estudio, investigación, o cualquier otro fin lícito. Con dicha finalidad, la Universidad asume los siguientes deberes y se reserva las siguientes facultades:

- La Universidad informará a los usuarios del archivo sobre los usos permitidos, y no garantiza ni asume responsabilidad alguna por otras formas en que los usuarios hagan un uso posterior de las obras no conforme con la legislación vigente. El uso posterior, más allá de la copia privada, requerirá que se cite la fuente y se reconozca la autoría, que no se obtenga beneficio comercial, y que no se realicen obras derivadas.
- La Universidad no revisará el contenido de las obras, que en todo caso permanecerá bajo la responsabilidad exclusive del autor y no estará obligada a ejercitar acciones legales en nombre del autor en el supuesto de infracciones a derechos de propiedad intelectual derivados del depósito y archivo de las obras. El autor renuncia a cualquier reclamación frente a la Universidad por las formas no ajustadas a la legislación vigente en que los usuarios hagan uso de las obras.
- La Universidad adoptará las medidas necesarias para la preservación de la obra en un futuro.
- La Universidad se reserva la facultad de retirar la obra, previa notificación al autor, en supuestos suficientemente justificados, o en caso de reclamaciones de terceros.

Madrid, a 19 de Julio de 2018

ACEPTA

Ignacio AGUIRRE

Fdo. Ignacio Aguirre Panadero

Motivos para solicitar el acceso restringido, cerrado o embargado del trabajo en el Repositorio Institucional:

Declaro, bajo mi responsabilidad, que el Proyecto presentado con el título

Electric energy consumer characterization / classification and forecasting using convolutional Neural

en la ETS de Ingeniería - ICAI de la Universidad Pontificia Comillas en el curso académico 2º M11 es de mi autoría, original e inédito y no ha sido presentado con anterioridad a otros efectos. El Proyecto no es plagio de otro, ni total ni parcialmente y la información que ha sido tomada de otros documentos está debidamente referenciada.

Fdo.: Ignacio Aguirre Panadero Fecha: 15 / 05 / 18

Ignacio Aguirre

Autorizada la entrega del proyecto

EL DIRECTOR DEL PROYECTO

Fdo.: José Ramón Vázquez Canteli Fecha: 15 / 05 / 18

J. Canteli

ESCUELA TÉCNICA SUPERIOR DE INGENIERÍA (ICAI)

INGENIERO INDUSTRIAL

ELECTRIC ENERGY CONSUMER CHARACTERIZATION, CLASSIFICATION AND FORECASTING USING CONVOLUTIONAL NEURAL NETWORKS

Autor: Ignacio Aguirre Panadero
Director: José Ramón Vázquez Canteli

Madrid

Julio 2018

Resumen

Introducción

Este trabajo presenta un procedimiento para apoyar a las comercializadoras en la extracción de información a partir de los datos de consumo eléctrico. El objetivo final es poder predecir curvas de demanda individuales para cada consumidor para cada una de las cuatro estaciones del año utilizando la información histórica de los meses previos. El algoritmo presentado en este trabajo consigue clasificar a los consumidores en una de los 7 clústeres propuestos, con una precisión del $70\,\%$ en el mejor de los casos, proponiendo a las comercializadoras un excelente punto de partida en la personalización de tarifas eléctricas. Los datos en forma de serie temporal son transformados en imágenes llamadas GAFs y MTFs [5] que son procesadas por un modelo de redes neuronales convolucionales para realizar tareas de clasificación. La estación del año y el día de la semana (laborable o festivo) tiene un gran impacto en los patrones de consumo. Por esta razón se usan estas dos características para separar la base de datos inicial en otros datasets reducidos. Dos datasets representan cada estación del año, uno para los días laborables y otra para los festivos. Los datos de consumo de los clientes durante una estación (3 meses) son reducidos a un único perfil diario conocido como el perfil diario representativo [2]. Los perfiles diarios representativos se obtienen promediando los perfiles de todos los días de una estación (laborables o festivos) hasta reducirlos a un único día. Los perfiles son promediados a lo largo de su dimensión temporal que son cada uno de sus datapoints representando un instante particular del dìa (24 o 96 dependiendo de si los datos se muestrean cada hora o cada 15 minutos). Por lo tanto, cada consumidor es descrito por dos perfiles diarios representativos para cada dataset (invierno, verano, ...), uno para

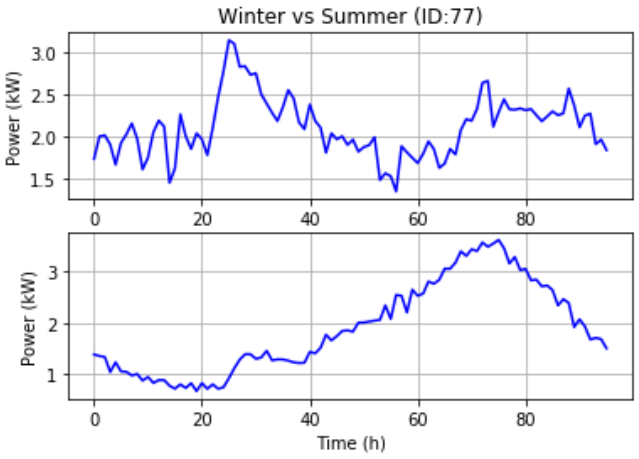


Figura 1. Perfiles diarios representativos de los días laborables de invierno y verano para el consumidor ID77. Datos muestreados cada 15 minutos.

los días laborables y otro para los festivos. La Figura 1 muestra los perfiles diarios representativos de los días laborables de invierno y verano para el consumidor ID77.

Metodología

En este trabajo se desarrolla un procedimiento para reducir la incertidumbre en el consumo eléctrico residencial para ayudar a las comercializadoras a ofrecer tarifas personalizadas a sus nuevos clientes. La Figura 2 muestra el procedimiento llevado a cabo usando un diagrama de flujo. Primero se realiza un estudio de caracterización de las diferentes clases de consumidores típicos entre nuestros clientes. Una vez que las clases han sido definidas, es posible personalizar tarifas eléctricas que se adapten a cada clase previamente definida. Una buena solución con los nuevos clientes sería ofrecerles inicialmente una tarifa fija hasta recoger información suficiente sobre sus patrones de consumo. Tras un mes de datos, es posible clasificar a estos nuevos clientes en una de las clases definidas por el clustering.

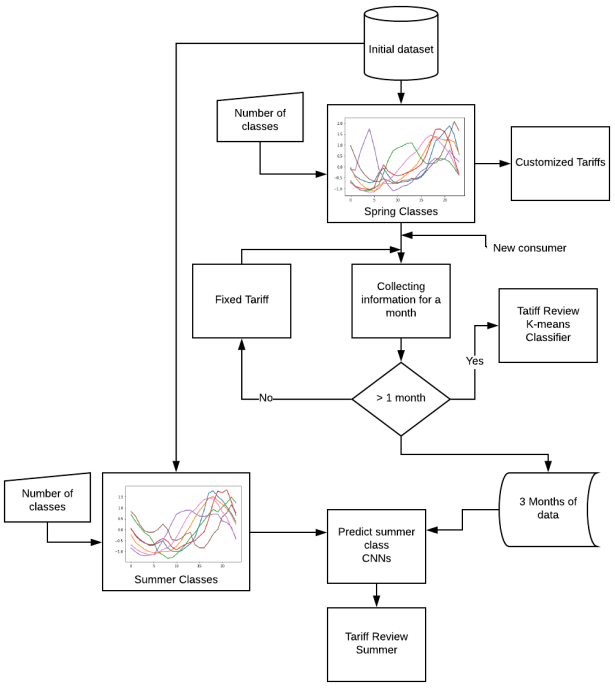


Figura 2. Procedimiento de clasificación

A la hora de clasificar se comparará el desempeño de dos algoritmos de clasificación como son las Redes Neuronales Convolucionales (CNNs) y K-means. Teniendo a los consumidores clasificados, es posible ofrecerles una tarifa que encaje con sus necesidades. Tras haber recolectado información del consumo durante una estación completa , es posible predecir en qué clúster se encontrará cada cliente en la próxima estación. Clasificar consumidores para la próxima estación es una tarea de mayor dificultad puesto que ahora los algoritmos clasificarán en base a información perteneciente a un periodo distinto al cual se quiere clasificar. Para las predicciones en vez de usar tres meses de información promediada para reconstruir un perfil diario representativo, se utilizará únicamente el último mes de información puesto que se ha demostrado en diferentes simulaciones que es el que tiene mayor poder de predicción. Esto se debe a que recoge la información temporalmente más próxima a la que se quiere predecir y por lo tanto la más parecida. Las tarifas eléctricas que se proponen en este trabajo serán revisadas cada 3 meses (cada estación). El caso de estudio presentado en este trabajo se centrará en las

estaciones de primavera y verano puesto que son las que dan mejores resultados. Más resultados respecto a otras estaciones serán también incluidas en este trabajo.

Resultados

La base de datos utilizada contiene información de 496 consumidores residenciales de las áreas de Tejas, California y Colorado. Esta base de datos es proporcionada por Pecan Street una organización local de Austin [6]. La Figura 3 muestra los centroides de los clústeres para Primavera 2015. Todos los consumidores deben ser etiquetados del 0 al 6 dependiendo de sus patrones de consumo en primavera. Con un mes de datos se obtiene precisión suficiente para clasificar a nuevos consumidores en una de las clases de la Figura 3. Los perfiles diarios representativos han sido normalizados mediante Normalización-Z para comparar su forma.

Wang Zhiguang en su trabajo [5] muestra los procedimientos para transformar series temporales en imágenes las cuales se emplean en este trabajo. Wang propone un framework en el que codifica series temporales en Markov Transition Fields (MTF) y Gramian Angular Fields (GAF). La definición de estas imágenes permite el uso de CNNs como algoritmo para clasificar nuevos consumidores a las clases especificadas. La Tabla 1 muestra un ejemplo de un perfil diario representativo transformado en GAFs y MTFs.

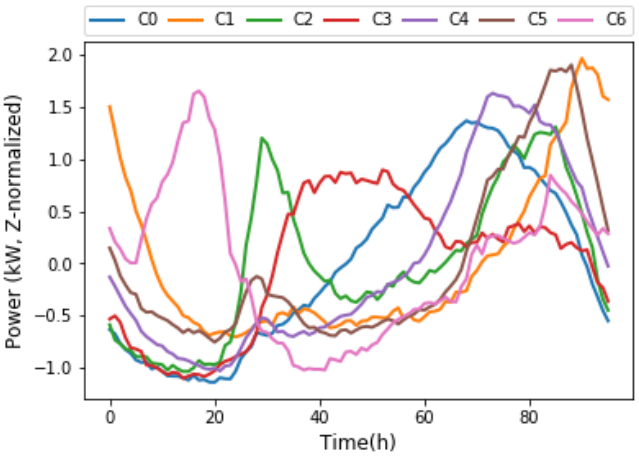
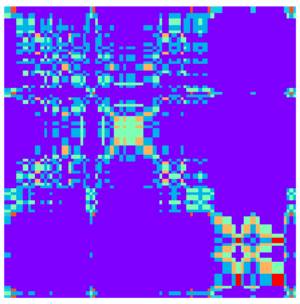
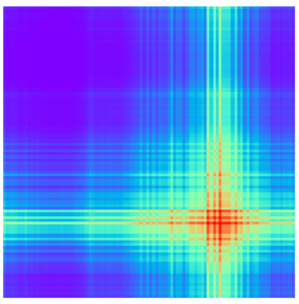
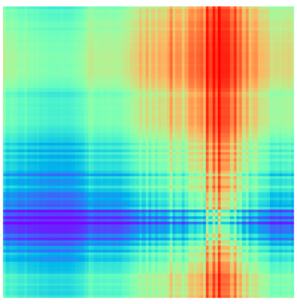


Figura 3. Representación de los centroides de los clústeres para los días laborables de Primavera 2015 utilizando una frecuencia de 15-min en los datos.

Tabla 1. MTF, GASF, y GADF de los perfiles diarios representativos para el último mes de Primavera 2015 para el ID77

MTF (Q=16)	GASF	GADF
		

Debido al pequeño tamaño del dataset (496 consumidores), la precisión de las redes neuronales depende en gran medida de que elementos han sido empleados en el training y cuales en el testing. Para obtener resultados que realmente representen el desempeño del algoritmo, se ha entrenado el algoritmo 20 veces distintas para presentar su precisión media con el test set y su desviación típica. El desempeño en la tarea de clasificación por el algoritmo K-means usando las series temporales y el desempeño de las redes neuronales usando los MTFs y los GAFs es comparado en la Tabla 2. En este caso la Tabla 2 muestra que el clasificador K-means funciona mejor.

Tabla 2. Resultados de la clasificación para Primavera 2015 usando el primer mes de primavera como entrada

Kmeans	CNNs			
Precisión TS	Precisión Test		Precisión Train	
0.7056	μ	s	μ	s
	0.6464	0.0608	0.6523	0.0595

En general, el algoritmo K-means presenta un mayor acierto que las CNNs para primavera. Esto se debe a que las clases en primavera han sido creadas utilizando un algoritmo de clasificación no supervisada K-means y por lo tanto el empleo de K-means en clasificación dará los mejores resultados posibles. El K-means no consigue un acierto del $100\,\%$ puesto que las clases o etiquetas han sido creadas usando la información de 3 meses de datos de primavera mientras que los algoritmos de clasificación han usado únicamente la información del primer mes de primavera. De la misma manera que se ha hecho con la Figura 2, la Figura 4 muestra las clases formadas en verano.

Para las predicciones se usarán los perfiles diarios representativos del último mes previo a la estación que se pretende predecir. Los dos algoritmos utilizados son comparados en la Tabla 3. En este caso, el modelo de CNNs tiene un mejor desempeño que el K-means al cual mejora en un $18\,\%$. De los resultados presentados en la Tabla 3 se puede afirmar que el modelo CNNs está ligeramente overfitted debido al reducido número de clientes usados en el training. En la predicción de verano los resultados obtenidos con el modelo de CNNs son mejores debido a la información extra aportada por los GAFs y los MTFs.

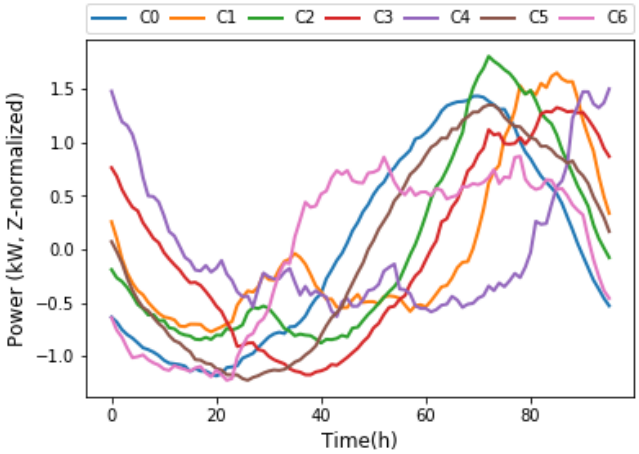


Figura 4. Centroides para los clústeres de los días laborables de Verano 2015 con los datos recogidos con una frecuencia de 15-min

Tabla 3. Resultados de las predicciones para Verano 2015 usando el último mes de primavera como entrada

Kmeans	CNNs			
Precisión	Precisión Test		Precisión Train	
	μ	s	μ	s
0.5968	**0.7023**	0.0371	0.7330	0.0345

Adicionalmente para medir la reducción de la incertidumbre en la demanda, se han calculado diferentes indicadores para cuantificar las diferencias entre la potencia real consumida y las predicciones para el Verano 2015. El problema es que cuando se hacen predicciones de las diferentes clases de verano se pierde información clave para poder reconstruir los perfiles diarios representativos en verano. En este caso lo que se pierde es valor medio y la desviación típica de los perfiles diarios representativos necesarios para poder de-normalizar los centroides de los clústeres. Se tiene la opción de usar la media y la desviación del perfil usado como input, en este caso el del último mes de primavera. En este caso, si calculamos las diferencias a lo largo de las 96 dimensiones (hora 0, hora 1) entre los valores reales y las predicciones, se obtiene una diferencia del $22{,}33\,\%$ para el algoritmo K-means y una diferencia del $23{,}18\,\%$ para las predicciones de las redes neuronales.

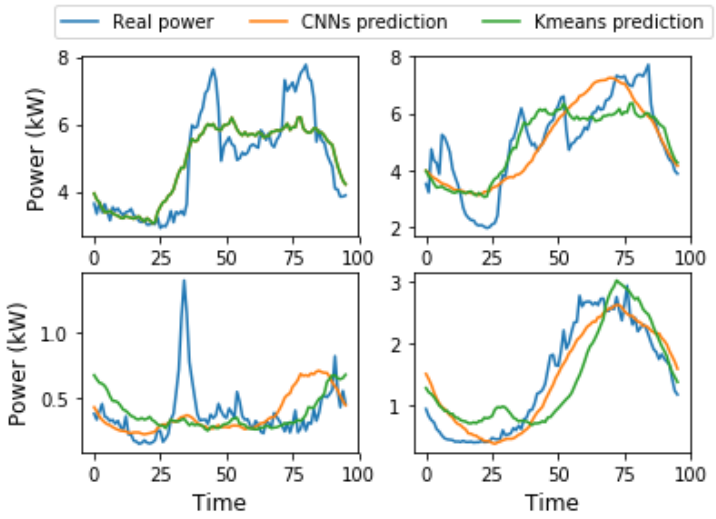


Figura 5. Comparativa de las predicciones de los perfiles diarios representativos obtenidas por los modelos K-means y CNNs usando los valores reales de la media y la desviación estándar para 4 consumidores

Por otro lado, si se tuvieran los valores reales de la media y la desviación típica, los resultados serían mucho más precisos. La Figura 5 muestra la comparación entre los perfiles representativos reales y las predicciones para 4 consumidores usando los valores reales de la media y la desviación típica para verano. En el caso en el que los algoritmos K-means y CNNs coinciden en la clasificación, los perfiles de las predicciones quedan solapados. En este caso las diferencias se reducen significativamente alcanzando una diferencia del $3{,}65\,\%$ para la predicción del K-means y una diferencia del $0{,}65\,\%$ para la predicción de las CNNs.

Conclusiones

En este trabajo se ha mostrado la utilidad de los más novedosos algoritmos de machine learning en aplicaciones reales. La alta precisión alcanzada por modelos de CNNs en tareas de clasificación de series temporales presentadas en otras trabajos fue la inspiración para aplicar estos conceptos en la predicción de demanda eléctrica. La definición de los GAFs y los MTFs con la combinación de las Redes Neuronales Convolucionales, aporta nuevas posibilidades en el análisis de series temporales de una manera totalmente diferente a como se ha hecho tradicionalmente. Entre otras cosas este trabajo explica como extraer información significativa a partir de las formas de los perfiles de demanda obtenidos a partir de grandes cantidades de datos. Además se explica como transformar las series temporales en imágenes que pueden ser clasificadas por modelos de redes neuronales. Este trabajo también desarrolla un procedimiento generalizado para obtener los perfiles de consumidores típicos presentes en cualquier base de datos y usar esta generalización para predecir futuras demandas. Estas predicciones ayudarán a reducir la incertidumbre en la demanda eléctrica lo que ayudará reducir los costes a las comercializadoras.

Summary

Introducction

This work presents a procedure to support the retail and distribution companies on the extraction of knowledge from electricity consumption data. Our final objective is to forecast individual load profiles of consumers for a particular season of the year using the historical information gathered during the months of the previous season. The algorithm classifies consumers in one of the 7 clusters, with 70% accuracy in the best case, which gives retail companies an excellent point to start in tariff customization. We encode the power use (time series) in GAFs and MTFs [5] to represent the images that are processed by the CNNs to perform classification tasks.

The season of the year and the type of day (working days or weekends) affect electricity consumption patterns. We use these characteristics to separate the data into smaller datasets. Two datasets represent each season of the year, one for working days and another for weekends. The consumer data of each season of the year is reduced to one daily load profile which is known as the representative load profile. Representative load profiles are obtained by averaging the load profiles of the whole season to one day. The data is averaged along its temporal dimension which are the 24 hours of a day (96 points, data collected every 15 minutes). Each consumer is then described by one single representative load profile in each dataset, for the different loading conditions. For example, Consumer X will have a representative daily load diagram for winter weekdays and a different one for summer weekday [2]. Figure 1 depicts the daily representative profiles for winter and summer workdays

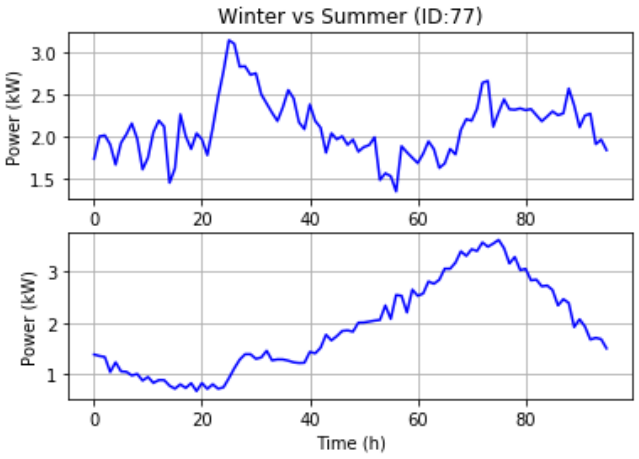


Figure 1. Daily Representative Profiles for winter and summer workdays with data collected every 15 minutes for the consumer ID77

for consumer ID77. The representative profiles are normalized using Z-Normalization to make shape comparisons possible.

Methodology

In this work, we have developed a procedure to reduce the uncertainty in residential electricity demand to help electricity providers in offering personalized electric tariffs to new clients. Figure 2 depicts this process using a block diagram to show the procedure flow. The first step is to carry out a characterization study of the different classes of electric behaviors among the consumers. Once we define the classes, it is possible to customize electric tariffs that will adapt to each of the classes defined. A good approach with new consumers would be to initially offer them a fixed price tariff until electricity providers gather enough information about their consumption patterns. After collecting information for the first month, we can classify consumers in one of the classes defined by the clustering. At this point we will compare the performance of the CNNs model and the K-means algorithm in the classification task. Having consumers classified, it is possible to offer them the tariff that will better fit their needs. Once we collect one season of data, it is possible to forecast in which cluster class the client will be the next season. Classifying consumers for the next season is a more difficult task because the algorithms now must classify consumers using information from a different period. For the forecasting, instead of using three months of data averaged to construct a representative daily profile, the data will be averaged per month with the purpose of losing the least information possible. But after doing some simulations, it can be stated that the data that have more predicting power is the most recent one to the season that wants to be predicted. An explanation of this fact is that recent data will be more similar in shape to the profiles that we pretend predict. For this reason, we decided to use just the third month of the previous season to predict consumer classes of the next season. Therefore, electric tariffs will be reviewed once every three months. Our case study will focus on the seasons of spring and summer which give the best results. More results concerning the rest of the year will also be provided.

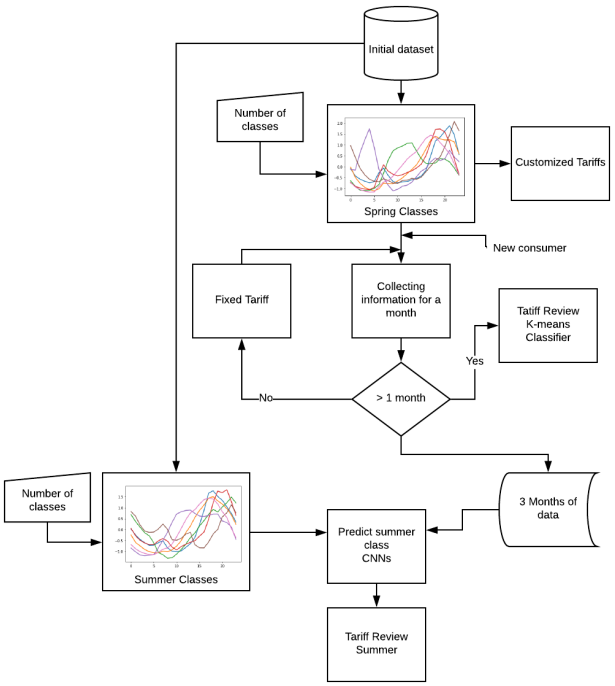
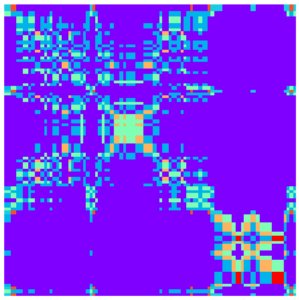
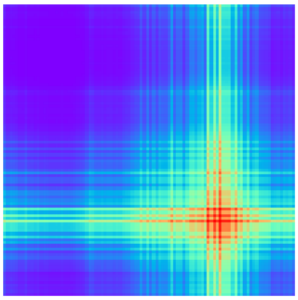
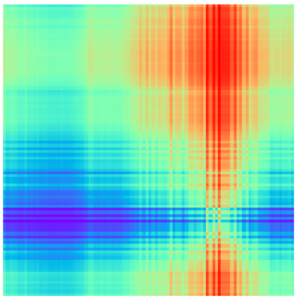


Figure 2. Classification procedure

Table 1. MTF, GASF, and GADF of the representative daily profile of the last month of spring 2015 for ID77

MTF (Q=16)	GASF	GADF
		

Results

The database used contains information from 496 residential consumers between Texas, California and Colorado. This database is provided by Pecan Street an Austin local organization [6]. Figure 3 illustrates the cluster centroids for Spring 2015. At this point, all the consumer have been labeled from 0 to 6 depending on their spring consumption patterns. With a month of data, there is enough accuracy to classify new consumers in one of the classes that Figure 3 describes. Wang in his paper [5] shows the techniques to encode time series as images that we can directly apply to our data. In his conference paper, he proposes a framework to encode time series as different types of images, namely, Gramian Angular Fields (GAF) and Markov Transition Fields (MTF). The definition of these fields allows the use of Convolutional Neural Networks as the algorithm to classify new consumers to the specified classes. Table 1 depicts an example of a representative daily profile encoded as GAFs and MTF.

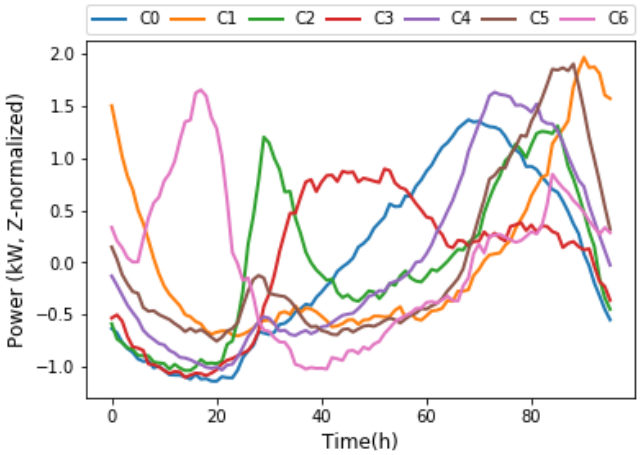


Figure 3. Cluster centroids for Spring 2015 workdays with the data collected in 15-min intervals

Due to the small size of the dataset (496 consumers), the accuracy of the neural network depends heavily on which elements were used for the training and which for the testing. Therefore, we trained the neural network model 20 different times to present its average test accuracy values and its standard deviation. The performance in the classification task of the K-means algorithm using the raw time and the CNNs model using the MTFs and GAFs is

Table 2. Classification results for Spring15 using the first month of Spring as input

Kmeans	CNNs			
TS Accuracy	Test Accuracy		Train Accuracy	
0.7056	μ	s	μ	s
	0.6464	0.0608	0.6523	0.0595

Table 3. Classification results for summer 2015 using the last month of spring as input

Kmeans	CNNs			
Accuracy	Test Accuracy		Train Accuracy	
0.5968	μ	s	μ	s
	0.7023	0.0371	0.7330	0.0345

compared in Table 2. In this case, Table 2 shows that the K-means classifier works better.

In general, the K-means algorithm has a better performance than the CNNs for spring. The reason of this is that the spring data have been labeled using a K-means algorithm and therefore the K-means algorithm for classification will give the best results possible. The K-means do not achieve an accuracy near to 100% because the labels have been created using the information of the three months of spring whereas the classification algorithm uses the data of the first month of spring. In a similar way we did in Figure 2, Figure 4 shows the different classes in summer.

For forecasting, we will be using the representative daily profile of the last month of the previous season to the one that is predicted. Two different classification algorithms are compared in Table 3. In this case, the CNNs model performs significantly better than K-means algorithm, achieving around an 18% improvement. From the results presented in Table 3, we can stated that the CNNs model is slightly overfitted due to the reduce number of consumers used in the training. When predicting in summer It seems that the results obtained by the CNNs are the best. This is due to the extra information provided by the GAFs and MTFs.

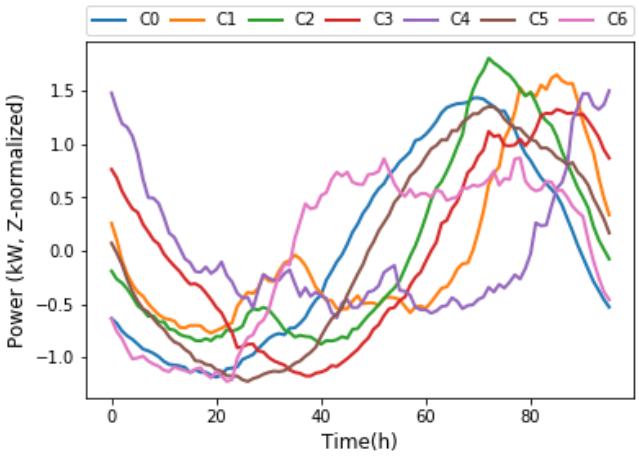


Figure 4. Cluster centroids for summer 2015 workdays with the data collected in 15-min intervals

Additionally, to measure the reduction of unpredictability in the demand, we have calculated different indicators to quantify the differences in power between the real values and the forecasts for Summer 2015. The problem is that when we forecast summer classes, we loss information that is key to reconstruct the power profiles. In this case, we do not have the values for the mean

and the standard deviation of each representative power profile that are used to de-normalize the cluster centroids. We have the option of using the mean and standard deviation extracted from the last month of spring data. In this case, if we compute the difference along the same dimension (hour 0, hour 1, ...) between the real values and the predictions, we obtain a difference of the 22.33% for the K-means prediction and a difference of the 23.18% for the CNNs prediction.

On the other hand, if we had the real values of the average and the standard deviation the results would be much more accurate. Figure 5 depicts the comparison between the real representative profiles and the predictions for 4 consumers using the real values of the average and standard deviation for summer. In this case the differences are significantly reduced reaching a 3.65% for the K-means prediction and a difference of 0.65% for the CNNs prediction.

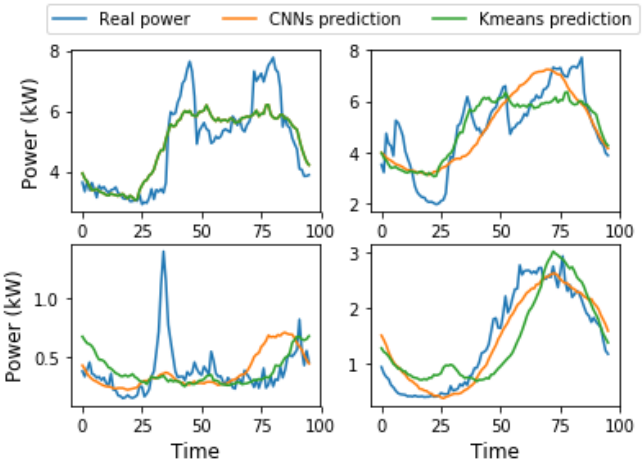


Figure 5. Comparison between the representative profile of the summer and the CNNs and K-means predictions using the real values of the average and standard deviation for 4 consumers

Conclusion

In this work, we showed how state-of-the-art machine learning algorithms can be used in real-world applications. The high accuracy of CNNs models in classifying time series presented in other works was the inspiration to try these concepts in electrical demand forecasting. The definition of the GAFs and the MTF with the combination of Convolutional Neural Networks, gives new possibilities in time series analysis in a different way as it has been done traditionally. This work has covered how to extract significant information of load profiles shapes from large quantities of data and how to encode this data in images that can be used in classification tasks. This work also covers a procedure to use this data to generalize the typical electric consumers and to use this generalization to predict future demands. This forecasts will help to reduce the uncertainty in electric demand which will reduce the costs of electricity providers. The next steps of our work, will be to prove these concepts in a real dataset provided by an electricity supplier to see how this procedure works and how it helps in tariff customization. Bigger datasets should give better results in terms of classification accuracy and consumer generalization.

Agradecimientos

En primer lugar agradecer a mis padres la posibilidad de haber estudiado en la Universidad Pontificia de Comillas con el correspondiente año de intercambio en The University of Texas at Austin donde he podido desarrollar este TFM.

En segundo lugar me gustaría agradecer a mi director de proyecto José Ramón Vázquez Canteli por todo el apoyo y consejo dado durante la realización de este Trabajo Fin de Máster sin el cual hubiera sido imposible realizar este trabajo. Sus buenas ideas siempre fueron un punto de partida para desarrollar mi trabajo. También agradecer a Stepan Ulyanin por compartir sus conocimientos de Machine Learning conmigo y aportar grandes ideas y trabajo a este proyecto.

Por último agradecer al Prof. Zoltan Nagy la posibilidad de formar parte del grupo de investigación Intelligent Environments Laboratory de la universidad de Tejas con el que hemos podido desarrollar este trabajo además de escribir un paper.

DOCUMENTO I

MEMORIA

Índice

Índice de figuras

Indice de tablas

Índice de extractos de código

PARTE I

MEMORIA

Capítulo 1

Introducción

A PARTIR de la liberalización que existe en el mercado eléctrico a finales del siglo XX, numerosos agentes se interesaron por la previsión de la demanda eléctrica. En estos últimos años, la previsión de demanda ha ganado mayor relevancia y se ha extendido su estudio. Uno de los mayores problemas de la generación de energía eléctrica es que no puede ser almacenada en grandes cantidades por lo que la demanda es satisfecha instantáneamente. Esto implica un balance permanente entre generación y demanda. Esta característica del sector eléctrico magnifica la importancia de la previsión de demanda eléctrica a corto plazo. La industria de la generación eléctrica debe anticiparse a la demanda futura de forma precisa para poder mantener el suministro de energía de forma continua así como asegurar la calidad de la energía suministrada. Con una correcta previsión de las demandas por parte de las comercializadoras podría reducirse el precio final de la energía entre un 2-10 $\%$ [1].

Las comercializadoras eléctricas están experimentado una época de cambios que están transformando el negocio tradicional del sector eléctrico. Uno de estos cambios es la cantidad de información que las comercializadoras disponen pero que no saben usar de una manera efectiva para reducir el coste de la energía a sus consumidores y mejorar la eficiencia energética del sistema en general. Este trabajo presenta un procedimiento con el objetivo de ayudar a las compañías comercializadoras en la extracción de conocimiento a través de los datos de consumo de sus clientes.

Una de las consecuencias más significantes de la liberalización del mercado eléctrico, es la libertad que tienen los consumidores a la hora de elegir la empresa que le suministre energía. Este libre mercado crea un entorno en el cual las empresas comercializadoras compiten por el suministro de energía. El conocimiento de cómo y cuándo los usuarios consumen electricidad es la clave para ser competitivo en el sector de la comercialización eléctrica [2]. Con la información histórica de la demanda de cada cliente, es posible obtener información valiosa para conocer mejor a los consumidores. Con esta información se puede obtener una generalización de los distintos tipos de consumidores presentes en nuestros datos. Una vez hecho esto, es

imprescindible clasificar a cada consumidor en la clase que mejor le represente en función de su curva de demanda. El número de clases en las que se separan los consumidores es un parámetro que debe ser definido en función del nivel de precisión requerida. Una vez que los consumidores han sido correctamente caracterizados, hay una mayor probabilidad de ofrecer una tarifa eléctrica que beneficie a los consumidores a largo plazo. Además esta caracterización ofrece otra serie de beneficios al sistema:

1. Mayor precisión en la predicción individual de demanda lo que permitiría una mejor planificación de la expansión del sistema manteniendo una alta calidad en el servicio [3].

2. Podría ayudar a reducir los costes asociados al exceso de capacidad en el diseño del equipo eléctrico empleado en la distribución como líneas o transformadores [4].

3. Podría ayudar a las comercializadoras a la hora de de determinar que clientes se verían beneficiados por programas de demand side response así como la posible implantación de tarifas variables o posibles deslastres de carga.

En este trabajo se aplican técnicas de clustering y de clasificación con el objetivo de obtener una serie de perfiles de demanda eléctrica que representen a los consumidores típicos presentes en nuestra base de datos. El objetivo final es poder predecir los perfiles de demanda de nuestros consumidores para las distintas estaciones del año utilizando información de la estación previa a la cual se quiere predecir. Para obtener las predicciones se emplean dos algoritmos de clasificación como son las Redes Neuronales Convolucionales y K-means quedándonos con aquel que presenta menor error en las predicciones.

Wang Zhiguang en su paper [5] presenta la técnica para transformar series temporales en imágenes la cual es aplicada sobre nuestros datos. En [5] se presentan los procedimientos para transformar series temporales en imágenes llamadas Gramian Angular Fields (GAF) y Markov Transition Fields (MTF). La presentación de estos "fields"permite usar redes neuronales convolucionales como algoritmo para clasificar nuevos consumidores a las distintas clases de consumidores típicos presentes en nuestra base de datos. La predicción de los hábitos de consumo eléctrico de los nuevos consumidores facilitará a las comercializadoras en la personalización de tarifas eléctricas y en la previsión del aprovisionamiento de energía necesario para cada hora del día. De esta manera se conseguirá reducir la incertidumbre en el consumo eléctrico lo que permitirá a las comercializadoras reducir sus costes. Esta reducción de costes afectará a los consumidores en forma de descuentos en sus facturas eléctricas.

Capítulo 2

Metodología

En este capítulo se pretende explicar la metodología llevada a cabo en el trabajo así como dar una visión general de los procedimientos desarrollados en los distintos capítulos de este documento. En este trabajo se ha desarrollado un procedimiento para reducir la incertidumbre en la demanda eléctrica residencial con la intención de ayudar a las comercializadoras a la hora de ofrecer tarifas personalizadas a nuevos clientes. La Figura 1 muestra el procedimiento propuesto en este trabajo usando un diagrama de bloques para el período de primavera y verano 2015. El hecho de escoger primavera-verano es únicamente un ejemplo de todas las posibles combinaciones que se pueden elegir.

Como se especifica en el título de este trabajo, se pretenden llevar a cabo tres análisis diferentes como son la caracterización de consumidores, la clasificación de consumidores y por último la predicción de demanda eléctrica de cada consumidor. La caracterización hace referencia a definir a cada individuo según su curva de demanda diaria. En cuanto a la clasificación, se pretende agrupar a los elementos en distintos clústeres de tal manera que los centroides de cada clase representen el comportamiento diario de los elementos agrupados en el mismo clúster. Para la clasificación no supervisada de consumidores se utilizará el algoritmo K-means utilizando como input los perfiles de demanda de cada consumidor. Por último con la predicción se pretende poder determinar en que clase quedaría agrupado cada usuario teniendo información del clúster en el que se encuentra en el período de tiempo inmediatamente anterior. En este caso será la estación inmediatamente anterior. Las estaciones quedan definidas según los equinoccios y solsticios definidos según el calendario para el hemisferio norte. En la predicción se comparará el desempeño de dos algoritmos completamente diferentes como son las Redes Neuronales Convolucionales y el algoritmo K-means.

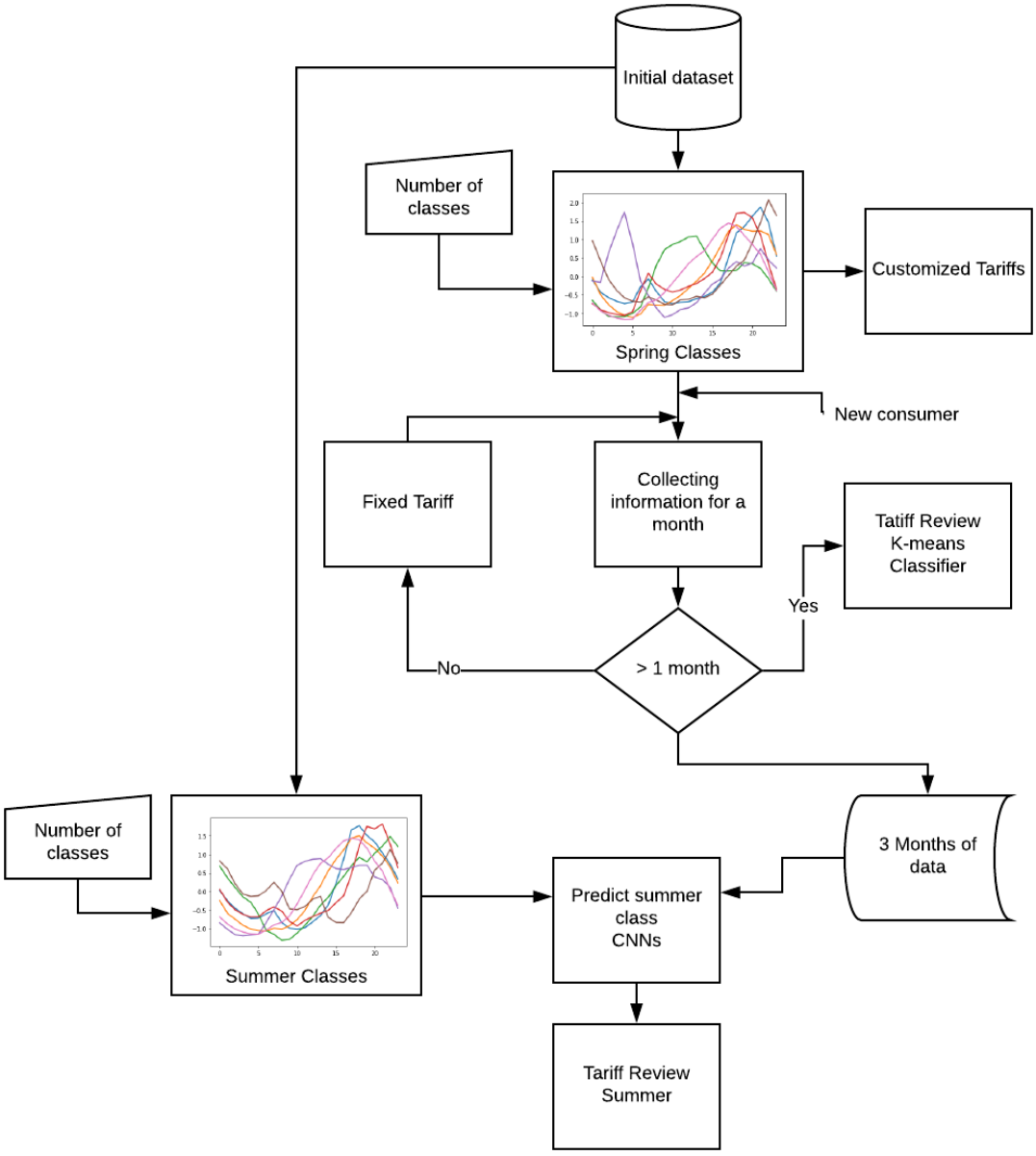


Figura 1. Diagrama de bloques para el período de primavera y verano 2015

Una vez hemos adquirido, limpiado y transformado los datos de los consumidores de la base de datos, el primer paso es llevar a cabo la caracterización de las distintas clases de consumidores típicos presentes en nuestros datos. Para definir las distintas clases de consumidores se ha utilizado un algoritmo K-means en el que es necesario introducir manualmente el número de clases en los que separar a los consumidores. El algoritmo K-means diferenciará a los consumidores según la forma de su curva de demanda. Una vez que se definen las distintas clases, es posible personalizar tarifas eléctricas que se adapten a cada una de las clases previamente definidas. Una buena estrategia aplicable a nuevos consumidores sería ofrecerles inicialmente una tarifa fija hasta que se recopile suficiente información para poder clasificarlos en una de las clases definidas por K-means. En este caso con un mes de datos, tenemos información

suficiente para clasificar a un nuevo consumidor de forma precisa. A la hora de clasificar a los nuevos consumidores con un mes de información, se comparará el desempeño de dos métodos distintos de clasificación: un algoritmo Kmeans y un modelo basado en Redes Neuronales Convolucionales. Con los consumidores clasificados, es más fácil que se le ofrezca una tarifa que se adapte mejor a sus necesidades. Tras recopilar información de la estación completa, en este caso los tres meses de primavera, es posible predecir en que clúster se encontrará un consumidor para la estación inmediatamente posterior. Clasificar a un consumidor para la siguiente estación es una tarea de mayor dificultad puesto que en este caso el algoritmo de clasificación utiliza información que no pertenece al período en el que se pretende clasificar, por eso a partir de ahora a este proceso lo llamaremos predicción en vez de clasificación. Para las predicciones en vez de usar tres meses de datos con los que hacer medias para obtener una curva diaria de demanda representativa del consumidor, se utilizará únicamente información del último mes puesto que este es el que tiene mayor poder de predicción ya que normalmente es el más parecido al período que se quiere predecir debido a la proximidad temporal. Por lo tanto las tarifas ofrecidas a nuestros consumidores serán revisadas una vez cada tres meses.

2.1. Caracterización de los elementos

La sección de caracterización resume los pasos necesarios para preparar los datos y obtener las diferentes clases de consumidores típicos presentes en nuestra base de datos. En esta sección se explican los parámetros clave para que la generalización de consumidores sea representativa. Esta sección es resumida en los siguientes procesos:

1. Selección y limpieza de los datos
2. Reducción de los datos
3. Normalización de los datos
4. Obtención de los perfiles de demanda

2.1.1. Selección y limpieza de los datos

El primer paso es la selección de los datos más significativos para el proceso. A la hora de realizar un estudio de este tipo se debe diferenciar entre distintos niveles de voltaje de los consumidores. Este trabajo se centra en el nivel residencial en el que el $90\,\%$ de los consumidores tienen una potencia contratada inferior a 10kW. El consumo de potencia es medido como la cantidad total de potencia consumida por todos los elementos de una vivienda para un instante en concreto. En la fase de limpieza se buscan las inconsistencias presentes en los datos como valores perdidos o valores inconsistentes los cuales son limpiados siguiendo los procedimientos descritos en el capítulo de limpieza y adquisición de datos.

2.1.2. Reducción de los datos

La estación del año (primavera, verano ...) y el día de la semana (día laborable o festivo) tienen un fuerte impacto en el consumo de energía eléctrica. Por este motivo se utilizarán estas características para separar los datos en distintas bases de datos de menor tamaño. Cada estación del año es representada por dos bases de datos según sean días laborables o festivos. Los datos de consumo de cada estación completa son reducidos a un único perfil diario representativo para cada consumidor. Este perfil representativo se obtiene promediando los perfiles de demanda de cada uno de los días de cada estación obteniendo finalmente un único perfil representativo que representa el consumo diario típico de un consumidor en función del día de la semana y la estación de año. Cada consumidor es descrito por un perfil diario representativo para cada base de datos según la estación del año. Por ejemplo el consumidor X tendrá un perfil diario representativo para los días laborables de invierno y un perfil totalmente distinto para los días laborables de verano. En la figura 2 se representan los perfiles diarios representativos del consumidor ID77 para las estaciones de invierno y verano según la frecuencia de muestreo de los datos. Las estaciones están definidas según los equinoccios y solsticios marcados por el calendario.

2.1.3. Normalización de los datos

Para poder comparar la forma de los perfiles diarios de demanda necesitamos normalizar los valores para poder comparar consumidores con distintas potencias contratadas. El objetivo es comparar la forma de los perfiles no el valor de potencia consumida por esto es importante normalizar los datos. Los perfiles diarios representativos son normalizados utilizando Normalización-Z. La fórmula de normalización es descrita a continuación:

$$x_i^{'} = \frac{x_i - \mu}{\sigma} \tag{1}$$

Este tipo de normalización asegura que todos los elementos del vector de entrada son transformados en un vector de salida cuya media es aproximadamente cero mientras que su desviación típica se encuentra en un rango cercano a 1. Este tipo de normalización permite mantener la forma del perfil para poder comparar las distintos patrones de consumo. La Figura 3 compara el perfil sin normalizar y normalizado para un consumidor en particular.

2.1.4. Obtención de los perfiles de demanda

El objetivo de esta sección es la partición de la base de datos inicial en un conjunto de clases definidas según la forma de los perfiles diarios representativos de cada consumidor. El modelo de data-mining para la caracterización de consumidores está basado en el algoritmo de clasificación no supervisada K-means para formar las clases. El perfil diario representativo del consumidor m es el vector $p^{(m)} = \left\{p_1^m, ..., p_h^m, ..., p_H^m\right\}$ donde p_h^m hace referencia a los valores normalizados de

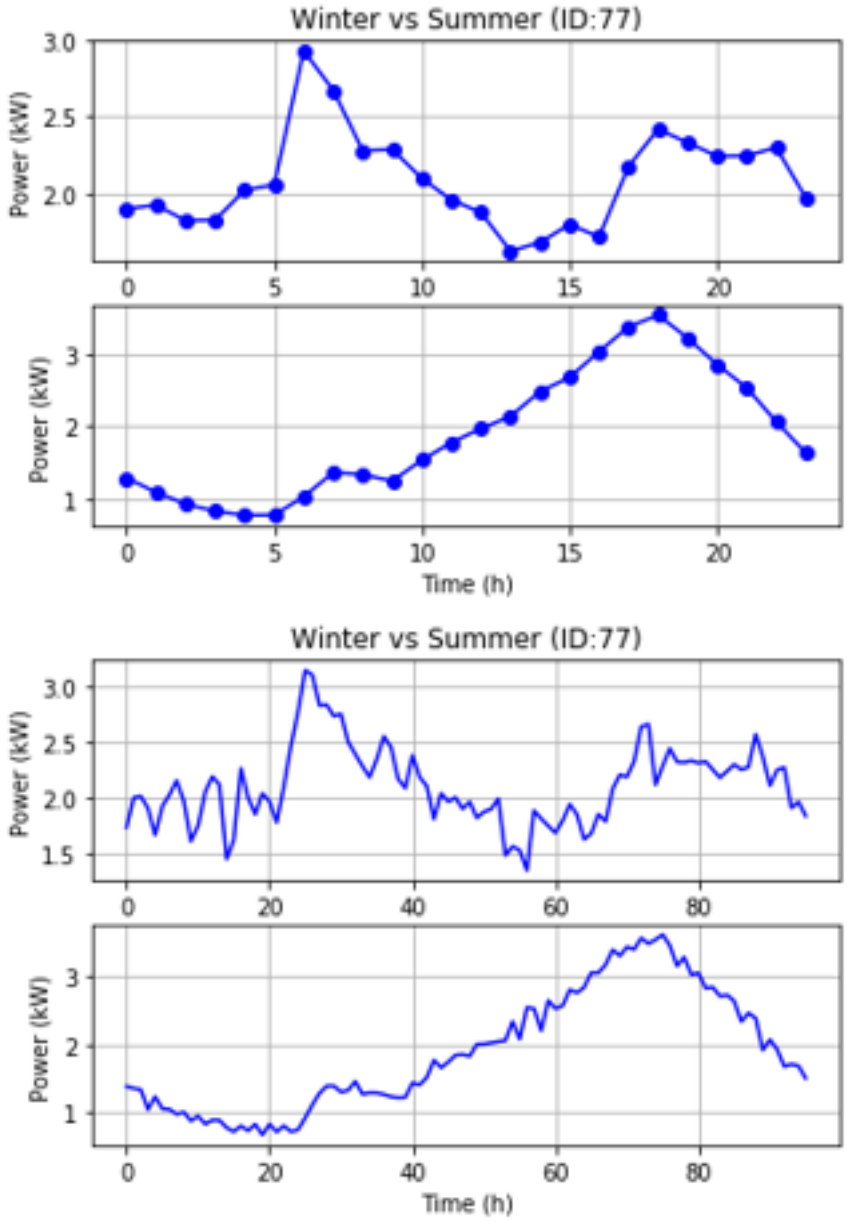


Figura 2. Comparativa entre el perfil diario representativo según la frecuencia de muestreo para primavera y verano

la potencia consumida en el instante h, donde $h = 1, ..., H$ con $H = 24$ o $H = 96$ dependiendo de la frecuencia de muestreo de los datos, 1 hora o 15 minutos. El uso de datos muestreados cada hora o cada 15 minutos dependerá de la aplicación del algoritmo de clasificación.

El algoritmo K-means compara la potencia consumida por los consumidores en cada una de las 24 o 96 dimensiones de manera que minimice la distancia entre elementos del mismo clúster. Los centroides representan el perfil promedio de todos los elementos agrupados en el mismo clúster. Uno de los problemas de K-means es que clasificaría dos time series idénticas en dos clústeres distintos si las curvas tienen un mínimo desfase temporal puesto que para calcular las distancias considera cada punto como una variable independiente (consumo de potencia en la hora 0, consumo de potencia en la hora 1...). Los perfiles muestreados con una frecuencia de 15

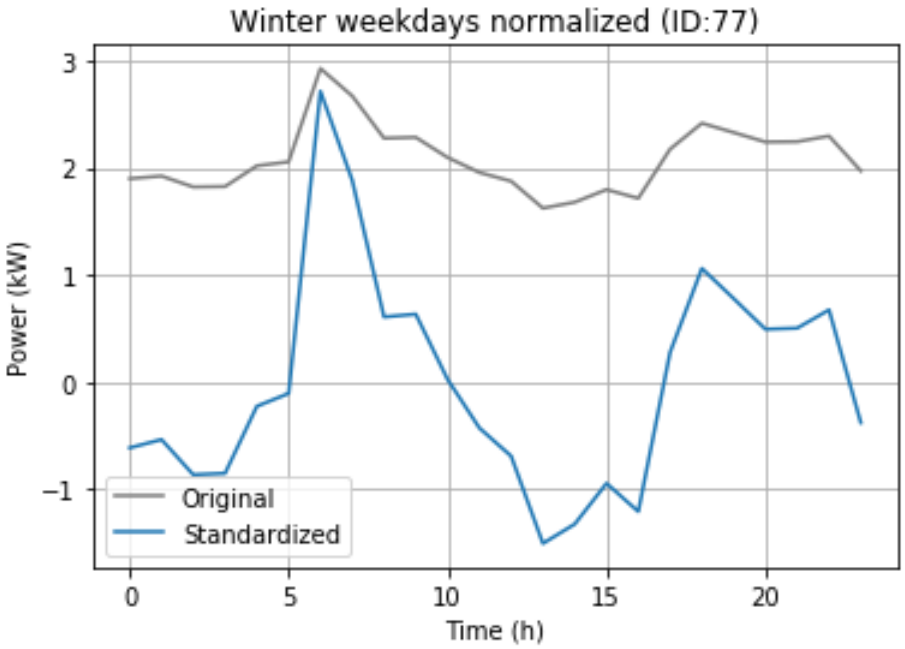


Figura 3. Comparativa entre el perfil diario representativo normalizado y sin normalizar

minutos son más sensibles a este problema y por lo tanto son una peor opción si el objetivo del clúster es encontrar clases en las que los elementos agrupados en el mismo clúster comparten metadatos como el tamaño de las casas o el número de individuos que viven en una casa. Por otro lado las CNNs tienen en cuenta las relaciones temporales entre cada una de las dimensiones de la serie temporal (consumo de potencia en la hora 0, consumo de potencia en la hora 1...) puesto que esta información es suministrada por los Markov Transition Fields y los Gramian Angular Fields de una forma parecida a la de una función de autocorrelación (ACF). Al utilizar datos muestreados cada 15 minutos obtenemos series temporales más precisas para el estudio de simultaneidad entre consumidores y para la personalización de tarifas eléctricas.

Las clases obtenidas por el clustering representan los diferentes patrones de consumo en la muestra de estudio. Los perfiles de demanda de los centroides representan a cada una de las clases y representan a los consumidores típicos dentro de nuestros datos. Una buena caracterización de los consumidores es muy importante puesto que tendrá un gran impacto en el éxito del módulo de predicción de demanda. Una caracterización funcional debe garantizar que las clases definidas estén bien diferenciadas (existan diferencias notorias entre los centroides de cada clase) y que los clústeres sean compactos (los elementos agrupados en un mismo clúster deben tener perfiles diarios representativos muy parecidos y deben ser parecidos al perfil del centroide). Para evaluar el desempeño del algoritmo de clustering se usan dos medidas introducidas por G.Chicco [3]: el MIA que mide cómo de compactos son los clústeres y el CDI que evalúa las diferencias entre clústeres. El objetivo es minimizar ambas medidas para asegurar que los clústeres tienen poder discriminatorio.

Considérese un conjunto X de perfiles de demanda separados en K clases con $k = 1, ..., K$ y cada clase formada por un subconjunto $C^{(k)}$ de perfiles de demanda pertenecientes al mismo clúster k donde $r^{(k)}$ hace referencia al centroide del clúster k. Teniendo en cuenta esto, se pueden definir las siguientes medidas:

1. Mean Index Adequacy (MIA):

$$MIA = \sqrt{\frac{1}{K}\sum_{k=1}^{K} d^2(r^{(k)}, C^{(k)})} \tag{2}$$

2. Clustering Dispersion Indicator (CDI):

$$CDI = \frac{1}{d(R)}\sqrt{\frac{1}{K}\sum_{k=1}^{K} d^2(C^{(k)})} \tag{3}$$

El Mean Index Adequacy (MIA) depende de la media de las distancias entre los elementos de un mismo clúster y su centroide. El Clustering Dispersion Indicator (CDI) es directamente proporcional a la distancia entre los perfiles de demanda en el mismo clúster e inversamente proporcional a la distancia entre los distintos centroides de cada clúster. El número de clases seleccionado para nuestro algoritmo K-means debe seguir un equilibrio entre cómo de compacto sean los clústeres y el objetivo del clustering. La Figura 4 muestra como disminuye la distancia media entre elementos de un mismo clúster en función del número de clases.

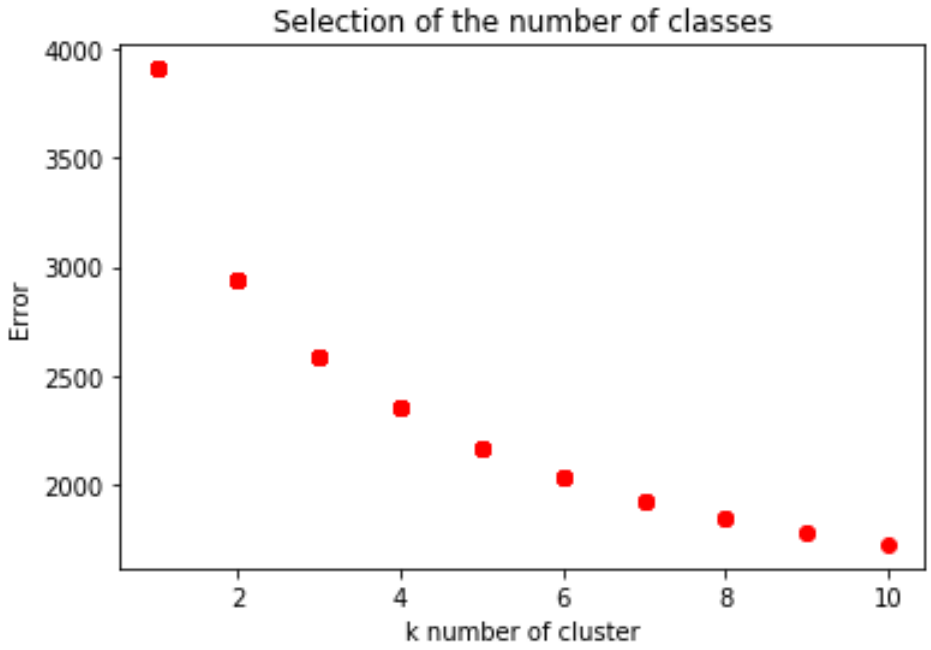


Figura 4. Disminución del error al aumentar el número de clases

Teniendo en cuenta lo mostrado en la Figura 4, un número razonable de clases debería estar estar entre 6 y 9 clases. Por debajo de 6, las clases son demasiado genéricas mientras que por

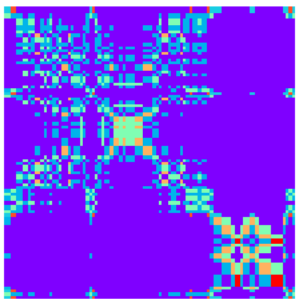
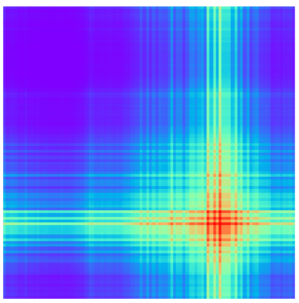
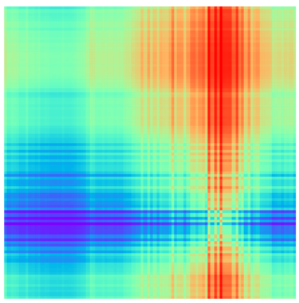
encima de 9 las clases son demasiado específicas. La idea es permitir a las comercializadoras eléctricas diseñar tarifas que se adapten a los hábitos de consumo de sus clientes. Por lo tanto ofrecer más de 6-7 tarifas diferentes parece demasiado mas aún teniendo en cuenta que normalmente las comercializadoras ofrecen dos tipos de tarifas una de precio fijo y otra de precio variable.

2.2. Sección de predicción

Este sección pretende explicar los pasos llevados a cabo para transformar las series temporales de los perfiles diarios representativos en imágenes que puedan ser procesadas por el modelo de redes neuronales convolucionales. También se explica por que esta visión es efectiva a la hora de realizar predicciones.

La efectividad de los Gramian Angular Fields (GAF) y de los Markov Transition Fields (MTF) en problemas de clasificación quedó patente tras la publicación *Encoding Time Series as Images for Visual Inspection and Classification Using Tiled Convolutional Neural Networks* [5]. El trabajo de Wang considera el problema de transformar series temporales en imágenes para que máquinas puedan visualmente reconocer, clasificar y aprender estructuras y patrones. En este trabajo se resumen las tres representaciones conocidas como Gramian Agular Summation/Difference Fields y Markov Transition Fields introducidas por Wang Zhiguang. La Tabla 1 muestra un ejemplo de estas tres imágenes que representan el perfil diario representativo de un consumidor en particular para el último mes de la primavera de 2015. Estas imágenes han sigo generadas utilizando la librería Pyts [7] de Python.

Tabla 1. MTF, GASF, y GADF de los perfiles diarios representativos para el último mes de primavera 2015 para el consumidor ID77

MTF (Q=16)	GASF	GADF
		

2.2.1. Gramian Angular Fields

Los Gramian Angular Fields (GAF) propuestos por Wang Zhiguang, son una manera de transformar series temporales en imágenes. Dada una serie temporal $X = \{x_1, x_2, ..., x_n\}$ de n valores reales, se escala X de tal manera que todos sus valores quedan recogidos en el intervalo $[0, 1]$:

$$\tilde{x}_i = \frac{x_i - min(X)}{max(X) - min(X)} \tag{4}$$

Una vez ha sido escalada se puede representar la serie temporal $\tilde{X}$ en coordenadas polares codificando sus valores como el coseno del ángulo y la dimensión temporal como el radio según las siguientes ecuaciones:

$$\phi = \arccos(\tilde{x}_i), -1 \leq \tilde{x}_i \leq 1, \tilde{x}_i \in \tilde{X} \tag{5}$$

$$r = \frac{t_i}{N}, t_i \in \mathbb{N} \tag{6}$$

Esta forma de representar series temporales en coordenadas polares es una novedosa forma de transformar las series temporales. El mapa de codificación explicado en las ecuaciones anteriores presenta dos propiedades importantes. Primero de todo, es biyectivo puesto que $\cos(\phi)$ es monótono cuando $\phi \in [0, \pi]$. Esto quiere decir que dada una serie temporal, el mapa propuesto produce un único resultado en el sistema polar con una única función inversa. La segunda propiedad es que a diferencia de las coordenadas cartesianas, la coordenadas polares preservan las relaciones temporales absolutas. En coordenadas cartesianas, el área viene definida por $S_{i,j} = \int_{x(i)}^{x(j)} f(x(t))dx(t)$ donde $S_{i,i+k} = S_{j,j+k}$ si $f(x(t))$ tiene los mismos valores para $[i, i + k]$ y $[j, j + k]$. Sin embargo, en coordenadas polares el área es definida como $S'_{i,j} = \int_{\phi(i)}^{\phi(j)} r[\phi(t)]^2 d(\phi(t))$ por lo tanto $S'_{i,i+k} \neq S'_{j,j+k}$. Esto significa que el área correspondiente desde el punto temporal i hasta el punto temporal j no solo depende del intervalo temporal $|i - j|$ sino que también depende de los valores absolutos de i y j.

Una vez la serie temporal ha sido escalada a coordenadas polares, podemos fácilmente explotar la perspectiva angular considerando las sumas y diferencias trigonométricas entre cada punto para identificar las correlaciones temporales entre diferentes intervalos de tiempo. Los Gramian Angular Summation Fields (GASF) y los Gramian Angular Difference Fields (GADF) son matrices definidas como:

$$GASF = \begin{bmatrix} \cos(\phi_1 + \phi_1) & ... & \cos(\phi_1 + \phi_n) \\ \cos(\phi_2 + \phi_1) & ... & \cos(\phi_2 + \phi_n) \\ ... & ... & ... \\ \cos(\phi_n + \phi_1) & ... & \cos(\phi_n + \phi_n) \end{bmatrix} \tag{7}$$

$$GADF = \begin{bmatrix} \cos(\phi_1 - \phi_1) & ... & \cos(\phi_1 - \phi_n) \\ \cos(\phi_2 - \phi_1) & ... & \cos(\phi_2 - \phi_n) \\ ... & ... & ... \\ \cos(\phi_n - \phi_1) & ... & \cos(\phi_n - \phi_n) \end{bmatrix} \tag{8}$$

La matrices generadas por los GAFs son matrices casi-gramianas tal que cada uno de sus elementos son productos vectoriales definidos como:

$$< x, y >= x \cdot y - \sqrt{1 - x^2}\sqrt{1 - y^2} \tag{9}$$

$$< x, y >= y \cdot \sqrt{1 - x^2} - x \cdot \sqrt{1 - y^2} \tag{10}$$

Tal que los GAF son matrices gramianas:

$$GAF = \begin{bmatrix} < \tilde{x_1}, \tilde{x_1} > & ... & < \tilde{x_1}, \tilde{x_n} > \\ < \tilde{x_2}, \tilde{x_1} > & ... & < \tilde{x_2}, \tilde{x_n} > \\ ... & ... & ... \\ < \tilde{x_n}, \tilde{x_1} > & ... & < \tilde{x_n}, \tilde{x_n} > \end{bmatrix} \tag{11}$$

Es muy importante escalar las series temporales en el intervalo $[0, 1]$ puesto que esto permitirá reconstruir la serie temporal a partir de un GASF. La diagonal de un GASF, i.e. $\{G_{ii}\}$ permite reconstruir precisamente la serie temporal original como:

$$\cos(\phi) = \sqrt{\frac{\cos(2\phi) + 1}{2}} \tag{12}$$

Según [5] la representación en forma de GAF presenta una serie de ventajas: los elementos situados abajo a la derecha de la matriz contienen la información más reciente de la serie temporal mientras que la información menos reciente se sitúa en los elementos superiores y a la izquierda de la matriz. A medida que te desplazas hacia abajo derecha en la matriz se produce un avance temporal en la serie temporal. La imagen generada retiene la dependencia temporal característica de las series temporales de tal manera que toda la información presente en la serie temporal es preservada. Los elementos de la diagonal de los GAFs aportan al menos la misma información que la serie temporal por si sola mientras que el resto de elementos aportan información que no está disponible explícitamente en la serie temporal como las correlaciones temporales entre elementos separados por un espacio de tiempo de $|i - j| = k$.

2.2.2. Markov Transition Fields

En [5] también se propone una segunda forma de transformar series temporales en imágenes extendiendo la idea de las cadenas de transiciones de Markov. Con este framework se pretende codificar las características dinámicas de transición representando las probabilidades de transición de Markov de forma secuencial para preservar la información en el dominio temporal.

Dada una serie temporal X, se separa en Q cuantiles-bins y se asigna cada x_i a su cuantil correspondiente $q_j(j \in [1, Q])$. La Figura 5 representa la separación de los datos en cuantiles.

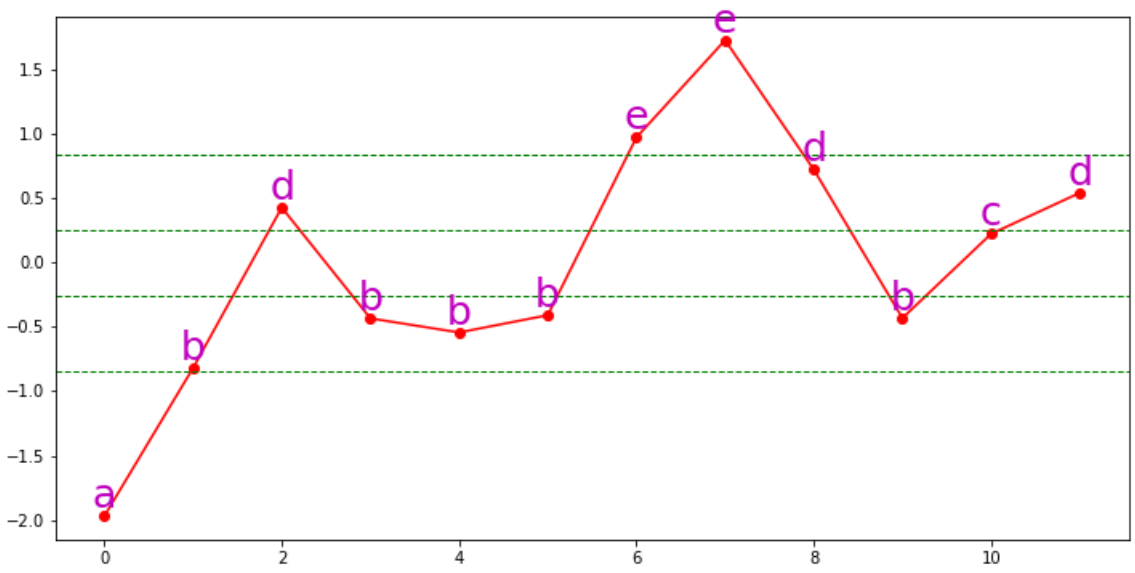


Figura 5. Separación de los valores de una serie temporal en cuantiles

Una vez separado en cuantiles, se puede construir la matriz W de tamaño QXQ que recoge las transiciones entre cuantiles de la misma manera que lo hace una cadena de Markov de primer orden a lo largo del eje temporal. $\{W_{ii}\}$ recoge la probabilidad de transición de elementos un cuantil a su mismo cuantil mientras que $\{W_{ij}\}$ recoge la probabilidad de transición de un elemento del cuantil q_i al cuantil q_j. W es conocida como la matriz de transiciones de Markov y tiene la característica que:

$$\sum_{j=1}^{j=n} w_{ij} = 1 \tag{13}$$

La matriz W de la serie temporal representada en la Figura 5 es mostrada en la Tabla 2:

Tabla 2. Matriz de transición de Markov para la serie temporal de la Figura 5

	a	b	c	d	e
a	0	1	0	0	0
b	0	$\frac{2}{5}$	$\frac{1}{5}$	$\frac{1}{5}$	$\frac{1}{5}$
c	0	0	0	1	0
d	0	1	0	0	0
e	0	0	0	$\frac{1}{2}$	$\frac{1}{2}$

La matriz W es insensible a la distribución de X y a la dependencia temporal para cada lapso de tiempo t_i. La pérdida de la dependencia temporal supone una pérdida de información demasiado grande por lo que para solucionar esta desventaja se define el Markov Transition Field como una matriz $M = |NXN|$ donde N representa el número de elementos de la serie temporal. $\{M_{ii}\}$ representa la probabilidad de transición propia del cuantil q_i en el instante t_i mientras que $\{M_{ij}\}$ representa la probabilidad de transición del cuantil en el instante de tiempo t_i al cuantil en el instante de tiempo t_j. La fórmula para codificar los Markov Transition Fields asegura producir una y solo una imagen para un número de cuantiles fijados Q para cada serie temporal.

El Markov Transition Field de la serie temporal representada en la Figura 5 es mostrado en la Figura 6

	0	1	2	3	4	5	6	7	8	9	10	11
0	0	1	0	1	1	1	0	0	0	1	0	0
1	0	2/5	1/5	2/5	2/5	2/5	1/5	1/5	1/5	2/5	1/5	1/5
2	...	...	...	...	...	...	...	...	...	...	...	...
3	...	...	...	...	...	...	...	...	...	...	...	...
4	...	...	...	...	...	...	...	...	...	...	...	...
5	...	...	...	...	...	...	...	...	...	...	...	...
6	...	...	...	...	...	...	...	...	...	...	...	...
7	...	...	...	...	...	...	...	...	...	...	...	...
8	...	...	...	...	...	...	...	...	...	...	...	...
9	...	...	...	...	...	...	...	...	...	...	...	...
10	...	...	...	...	...	...	...	...	...	...	...	...
11	...	...	...	...	...	...	...	...	...	...	...	...

Figura 6. Matriz M

2.2.3. Beneficios de la combinación de GAFs y MTFs

Según lo demostrado en [5], G_{ij} representa la superposición/diferencia de la direcciones para los instantes de tiempo t_i y t_j mientras que M_{ij} representa la probabilidad de transición del cuantil en el instante de tiempo t_i al cuantil en el instante de tiempo t_j. Los GAFs capturan información estática acerca de la serie temporal ya que contienen las relaciones temporales de una manera parecida a como un modelo de autoregresión (AR) hace. Por otro lado, los MTFs capturan información acerca de la dinámica de la serie temporal de una manera parecida a como lo hace una regresión logística, puesto que indican la probabilidades en la transición. Desde este punto de vista, se pueden considerar las tres imágenes como tres canales ortogonales como los diferentes colores rojo, verde y azul de una imagen RGB. Por lo tanto se pueden combinar las imágenes generadas por los GAFs y MTFs siempre y cuando sean del mismo tamaño (i.e. $S_{GAFs} = S_{MTF}$) para construir una imagen de tres canales (GASF-GADF-MTF).

Capítulo 3

Adquisición, limpieza de los datos y transformación de los datos

En este capítulo se explica el contenido de la base de datos utilizada y la selección de los datos más significativos. También se pretende explicar que estructura tienen los datos descargados y los procedimientos necesarios de limpieza indispensables en cualquier base de datos. Para poder entender plenamente la posterior transformación de los datos es imprescindible conocer la estructura que tiene los datos una vez descargados.

La base de datos utilizada es conocida como Dataport suministrada por la compañía PecanStreet [6]. Dataport contiene información sobre el consumo instantáneo de energía eléctrica de más de 1000 consumidores residenciales distribuidos por Tejas, California y Colorado. Se dispone de varios años de información desde 2012 hasta la actualidad. La potencia eléctrica consumida viene detallada según los diferentes electrodomésticos y equipos de consumo o también agrupada como el consumo eléctrico total de la vivienda. Los datos de potencia son adquiridos con una frecuencia de 15 minutos o 1 hora. Dependiendo de la finalidad del estudio se trabajará con los datos recogidos cada 15 minutos o cada hora. También se dispone de la metadata de los distintos consumidores como por ejemplo el tipo de vivienda (piso, chalet, etc..), el año de construcción de la vivienda, la superficie total de la casa o el número de habitantes.

Para este trabajo se ha reducido la base de datos mencionada anteriormente a 496 consumidores para los años 2014 y 2015. Puesto que los consumidores se inscriben y se borran del programa promocionado por PecanStreet, es necesario asegurarse que se dispone de información de los usuarios para todo el período de estudio. El volcado de los datos se realiza mediantes queries de PostgreSQL que son ejecutadas a través de un script en Python. Para poder visualizar la base de datos se ha empleado la aplicación PgAdmin con la que se ejecutan queries postgreSQL para filtrar los datos. A continuación se describen los distintos procedimientos ejecutados en el archivo *DataAcquisitionV8.py* y que tienen el objetivo de descargar, limpiar y transformar los datos para un posterior estudio en los siguiente capítulos de este trabajo.

3.1. Selección del periodo de estudio y automatización del código

El código se encuentra totalmente automatizado para que únicamente cambiando las variables de activación podamos descargar información perteneciente a distintos periodos de tiempo o con una frecuencia de colección diferente. El primer parámetro a definir es la frecuencia de muestreo de los datos (*CONST_SAMPLE_SIZE*) la cual podemos ajustar a 24 o 96 en función de si el período de muestreo es 1 hora o 15 minutos. Se ha de tener en cuenta que con un período de muestreo más pequeño el número de datos aumenta y por lo tanto el código incrementa notoriamente su tiempo de ejecución. Se debe seleccionar una estación del año para la descarga de los datos pero dentro de cada estación el período de estudio puede ser desde 3 meses a un día. El código tiene en cuenta los cambios de hora, a través de la variable *Winter_DaylightSavingTime*, que se producen durante los meses marzo y octubre ya que suponen que el número de datos para cada estación varíe.

3.2. Queries SQL

Las queries SQL sirven para descargar datos específicos contenidos en la base de datos que cumplan una serie de condiciones. Es una forma rápida de seleccionar lo datos que son útiles para el estudio sin tener que descargar la base de datos completa. En una base de datos SQL los datos se organizan en tablas y dentro de las tablas la información se organizan en columnas. Cada columna corresponde a una variable distinta de la base de datos mientras que cada fila representa el siguiente dato recogido. La estructura típica las queries SQL empleadas en este trabajo tiene la siguiente forma:

```
"SELECT columna
 FROM tabla
 WHERE columna BETWEEN inicio AND final
 ORDER BY columna ASC;"
```

Codigo 1. Ejemplo SQL query

Columna hace referencia a la variable que se quiere estudiar mientras que tabla hace referencia a la estructura tabular dentro de la base de datos a la que queremos acceder. Se pueden añadir condiciones como el *WHERE* y el *BETWEEN* para filtrar los datos a los que se quiere acceder. Como se comentó anteriormente, el código está automatizado por lo que dependiendo de la frecuencia en la adquisición de datos (1h o 15 minutos) la tabla a la que se accede será *university.electricity_egauge_hours* o *university.electricity_egauge_15min*. De esta

misma manera la columna a la que se accederá sera *localhour* para 1 hora y *local_15min* para intervalos de 15 minutos.

3.3. Conexión con la base de datos

Para conectarse a la base de datos se utilizará la librería "Psycopg2"que permite conectarnos a una base de datos online a través de un script de Python. Es necesario tener un usuario y contraseña para poder tener acceso a la base de datos.

```
conn = psycopg2.connect(host="67.78.67.93",database="postgres", user="",
    password="", port="5434")
cur = conn.cursor()
```

Codigo 2. Conexión a la base de datos SQL

3.4. Separación entre elementos perfectos y elementos a limpiar

En cualquier base de datos siempre se producen pequeños errores debidos a la falta de valores. Para poder diferenciar entre consumidores que no necesitan ser limpiados y consumidores a limpiar debemos separar a los consumidores en función del tamaño de los datos. La Figura 7 resume el proceso de separación entre elementos perfectos y elementos imperfectos.

Los elementos perfectos son aquellos que tienen el mismo tamaño que el periodo de estudio mientras que los imperfectos tiene un tamaño menor. También se incluye la separación entre elementos a limpiar y elementos desechables dentro de los elementos imperfectos.

Para calcular el tamaño de los datos se necesita saber el número de días dentro del periodo de estudio seleccionado y la frecuencia con la que han sido adquiridos los datos. Por ejemplo si se selecciona un día de inverno y los datos son registrados cada 15 minutos, el tamaño total de los datos debe ser de 96 elementos. Una vez conocido el tamaño de los datos, se lanza una query SQL para descargar los elementos cuyo tamaño es inferior al tamaño esperado. Estos elementos son catalogados como elementos imperfectos. La query SQL para descargar estos elementos es definida en Código 3:

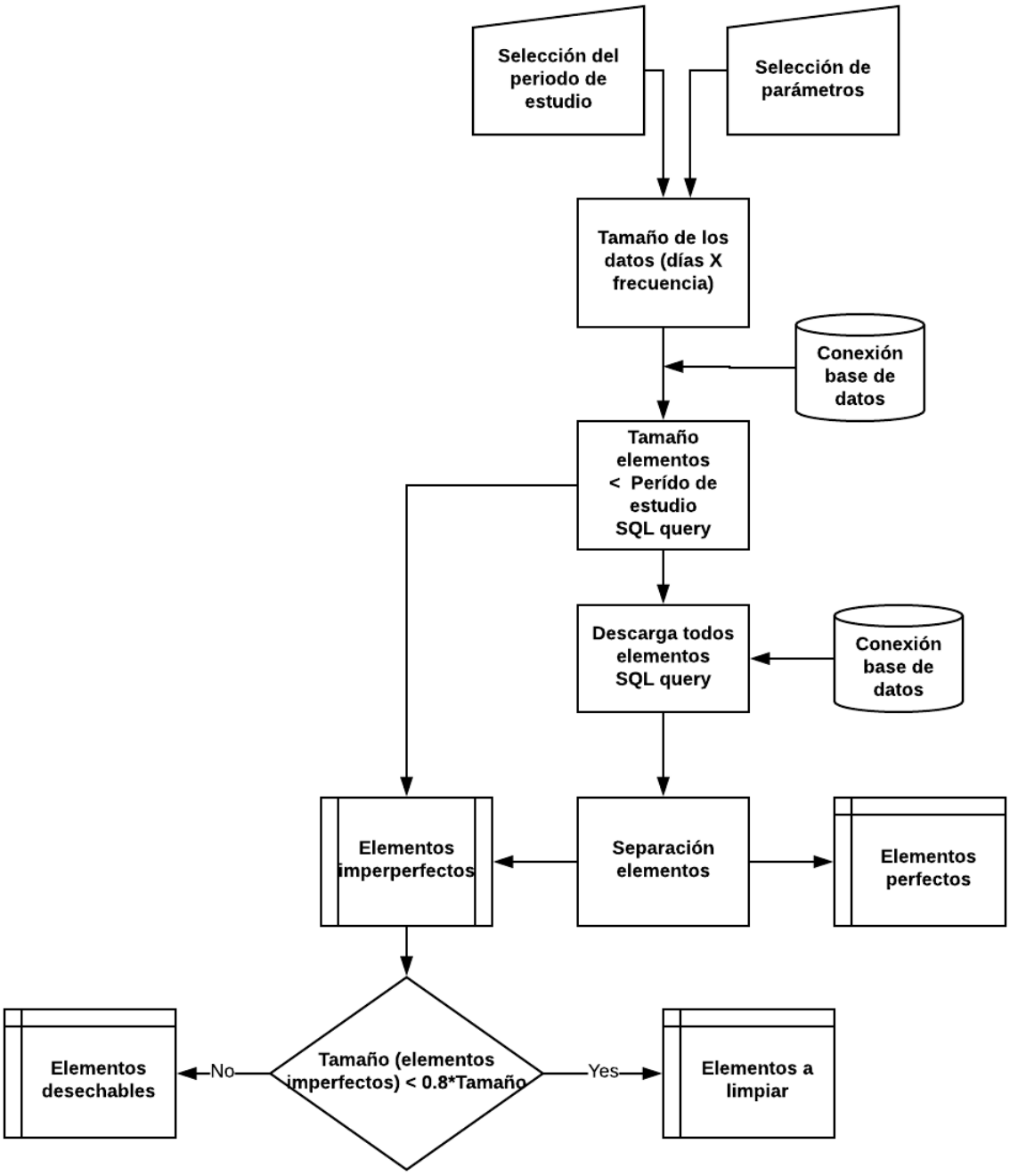


Figura 7. Diagrama de bloques para la separación de elementos

```
SELECT  dataid, COUNT(use) FROM university.electricity_egauge_15min
WHERE local_15min BETWEEN 'startdate' AND 'end date'
GROUP BY dataid
HAVING COUNT(use)<= data interval
ORDER BY dataid ASC;
```

Codigo 3. Separación elementos a limpiar

Acto seguido, descargaremos todos los elementos disponibles y separaremos entre elementos con el tamaño correcto y elementos con menor tamaño (elementos imperfectos). Si los elementos imperfectos conservan el $80\,\%$ del tamaño de los elementos perfectos serán considerados como elementos a limpiar. El código que ejecuta las acciones descritas por la Figura 7 es mostrado en su totalidad en los anexos de este trabajo. El Código 4 hace referencia a la query SQL para descargar todos los elementos:

```
SELECT local_15min, use, dataid
FROM university.electricity_egauge_15min
WHERE local_15min BETWEEN 'startdate' AND 'end date'
ORDER BY dataid ASC, local_15min ASC;
```

Codigo 4. Descarga de todos los elementos

3.5. Limpieza de elementos y transformación de los datos

En esta sección se describe como se ha llevado a cabo la limpieza de elementos y se describe la transformación de los datos para obtener los perfiles diarios representativos de cada consumidor. Aquellos elementos de los que disponemos al menos de el $80\,\%$ de la información podrán ser limpiados. A la hora de limpiar elementos podemos diferenciar entre 3 tipos de elementos:

- Tipo 1: son elementos que tienen un tamaño menor al tamaño esperado porque los consumidores han finalizado o comenzado un acuerdo con PecanStreet en mitad del periodo de estudio. Por esta razón, no se dispone de más información a partir de la fecha de finalización fijada. Con este tipo de consumidores se construirán sus perfiles diarios representativo promediando lo valores de los que se disponen. Esto supone una pequeña pérdida de información sin gran relevancia puesto que se ha garantizado que se dispone de al menos el $80\,\%$ de su información.

- Tipo 2: elementos con valores nulos en casillas sueltas del período de estudio. Estos valores serán recuperados siempre que no falten más de 2 horas seguidas de valores. Estos valores se obtendrán promediando el valor inmediatamente anterior y posterior o manteniendo el valor anterior.

- Tipo 3: estos elementos son una combinación de los elementos tipo 1 y tipo 2.

Los elementos Tipo 2 una vez limpiados tiene el mismo tamaño que los elementos perfectos por lo que se pueden obtener los perfiles diarios representativos de forma conjunta mientras que los elementos Tipo 1 y 3 al tener cada elemento un tamaño particular deben ser procesados de forma independiente para poder obtener el perfil diario representativo de cada cliente.

3.6. Función daily profile

Una vez que los todos los elementos han sido limpiados deben pasar por la función *daily_profile_V2(consumers, data_interval, timelag, CONST_SAMPLE_SIZE, WEEKDAYS, WEEKENDS)* que se encarga de calcular los perfiles diarios representativos a partir de los datos de consumo de los usuarios. El diagrama de flujo representado en la Figura 8 describe el proceso ejecutado en la función *daily_profile_V2()*. Como describe la Figura 8, el primer paso es la separación de los datos en días laborables y festivos. Esto se realizará a través de la función *numpy.is_busday()* que da un vector booleano señalando con 1 los días laborables y con 0 los días festivos. Conociendo el tamaño de los datos podemos calcular el número de consumidores que se pasan a la función, el número de días festivos y el número de días laborables. El siguiente paso es reconstruir los datos por días de manera que se construyan los perfiles de cada día. Una vez que se construyan todos los perfiles para la estación seleccionada, se computan las medias de todos los perfiles para obtener un único perfil diario representativo por cada consumidor. Una vez se haya hecho esto para todos los consumidores se saldrá de la función.

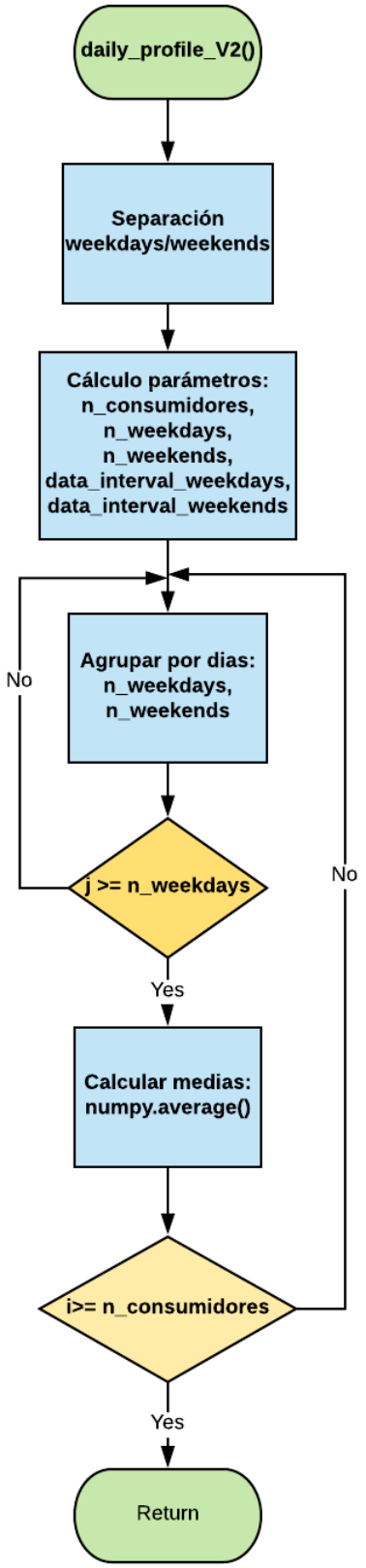


Figura 8. Diagrama de bloques función daily_profile_V2

.

3.7. Estructura final de los datos

Por último se comenta las estructura que tiene la variable final que almacena los perfiles diarios representativos de todos los consumidores presentes en la base de datos. Cada una de estas variables representa los perfiles diarios representativos para los días laborables o festivos para una estación concreta del año. Por esta razón el nombre seleccionado tiene la siguiente estructura *AV_consumer_Wdays_15min_Summer15V2* donde se especifica el tipo de día de la semana, la frecuencia de muestreo y la estación del año.

La estructura interna de esta variable en python es un numpy array con la forma (n_consumidores, n_datos_día, 3). N_consumidores hacer referencia al número de usuarios de los que disponemos su perfil diario representativo, n_datos_día hace referencia al número de puntos que describen su perfil diario representativo que puede ser 24 o 96 según la frecuencia de muestreo 1h o 15-minutos. Por último, el 3 hace referencia a 3 variables diferentes: fecha con día, hora y minutos, potencia consumida en ese instante y ID del consumidor. Por la tanto se puede decir que la información está almacenada de forma tabular de tal manera que disponemos de n_consumidores tablas con 96 o 24 filas y 3 columnas. En la Tabla 3 se muestra un ejemplo de la estructura para los días laborables del Verano 2015.

Tabla 3. Ejemplo de la estructura de las variables que almacenan los perfiles diarios representativos

Estructura de los datos en python			
Nombre	Tipo	Tamaño	Valor
AV_consumer_Wdays_15min_Summer15V2	object	(536, 96, 3)	ndarray object of numpy

Capítulo 4

Caracterización de los consumidores mediante Clustering K-means

En este capítulo se pretende encontrar los distintos patrones de consumidores típicos presentes en nuestra base de datos. Para caracterizar a los consumidores se utilizan los perfiles diarios representativos de los consumidores como input del algoritmo de clasificación no supervisada K-means. De esta manera creamos los labels o clases para cada consumidor. El número de clases en los que se separan los clientes debe ser un input introducido con el objetivo de buscar clústeres lo más compactos posible y que realmente representen las características de un grupo particular de consumidores.

El algoritmo K-means clasifica a los usuarios en distintos clústeres con el objetivo de minimizar las distancias entre elementos. Cada una de las dimensiones que componen los perfiles diarios representativos de los usuarios se traducen como las variables que utiliza el algoritmo para calcular distancias. Por ejemplo compara distancias entre los valores de consumo en la hora 0, en la hora 1 y así para todas las dimensiones. El algoritmo K-means ha sido implementado utilizando la librería Cluster de Scikit-Learn [8]

No interesa clasificar en función de la cantidad de energía consumida si no en función de la forma de sus curvas de demanda. De tal manera que lo que se pretende estudiar son lo hábitos de consumo del consumidor y no el poder adquisitivo de este o el volumen de potencia consumida. En función del tipo de normalización y de la frecuencia de muestreo en los datos elegidos los resultados de los clústeres serán diferentes. Puesto que el objetivo final del trabajo es poder predecir el consumo eléctrico de nuestros clientes con los datos de la estación anterior debemos asegurarnos que los consumidores seleccionados están presentes en ambos periodos.

La figura 9 representa el diagrama de flujo del archivo de código *ClusteringV2.py*

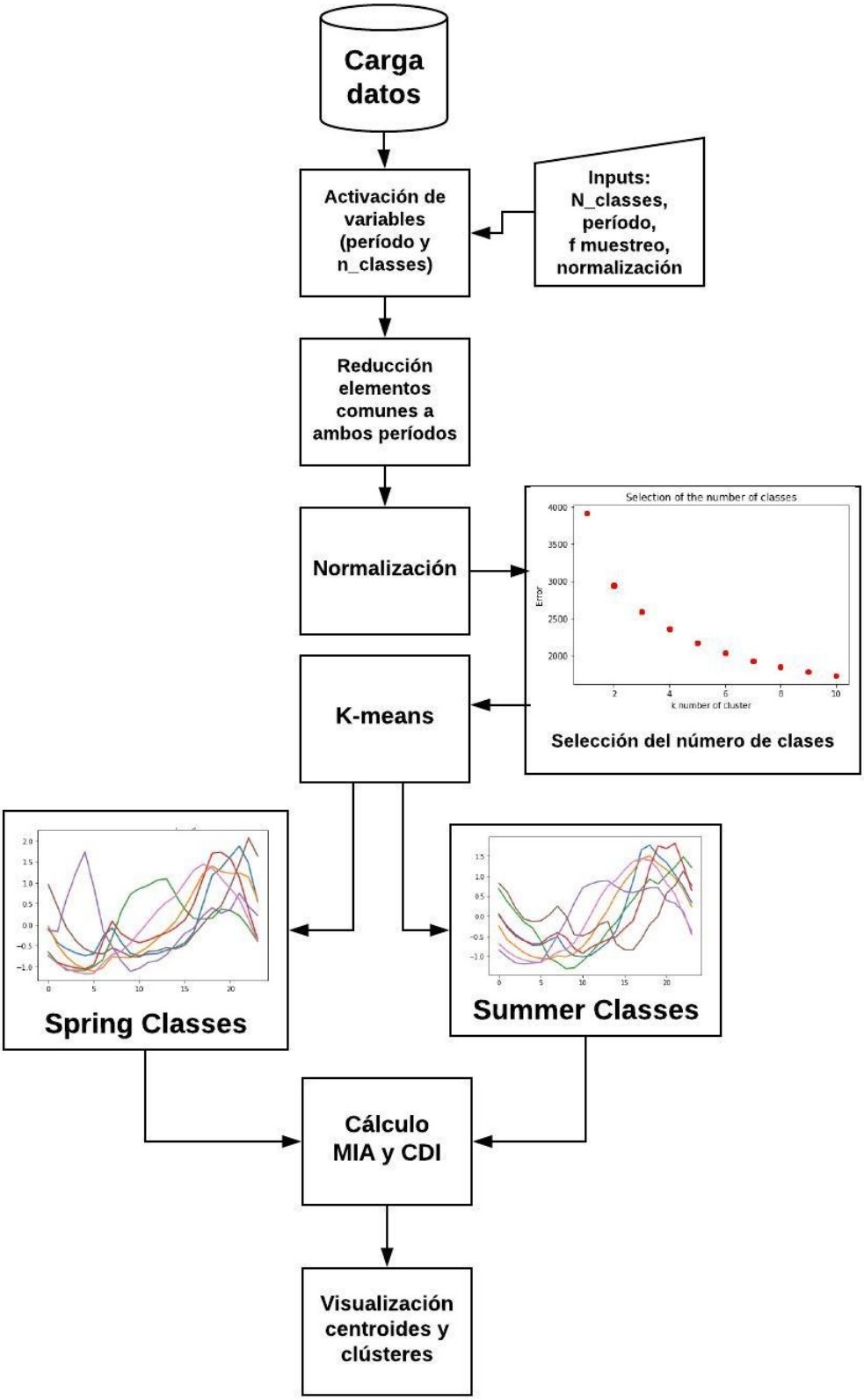


Figura 9. Diagrama de bloques para describir el proceso de caracterización

4.1. Descripción del procedimiento

Como describe la Figura 9, el primer paso es cargar los perfiles diarios representativos que han sido obtenidos a partir del código *DataAcquisitionV9.py* descrito en el capítulo anterior. Para guardar y cargar los datos en formato numpy se utilizan los comandos *numpy.save()* y *numpy.load()*. A continuación se definen distintos parámetros con las variables de activación. Entre otras cosas se selecciona el número de clústeres en los que se clasificarán a los usuarios, la frecuencia de muestreo con la que se utilizarán los datos (15 minutos o 1 hora), el tipo de normalización a utilizar (Z-normalización o normalización por potencia pico) y las estaciones seleccionadas (invierno, primavera, verano u otoño).

Es muy importante asegurarse que los usuarios a los que se clasifica están presentes en ambos períodos de estudio puesto que el objetivo es poder predecir en que clúster estará cada cliente en la próxima estación conociendo en que clúster se encuentra actualmente. Por lo tanto, para comprobar la precisión del algoritmo es necesario tener los resultados reales para poder comparar con las predicciones. Para lograr esto se ha utilizado el comando *numpy.intersect1d()*. Una vez que se ha reducido el número de clientes a aquellos comunes a ambos periodos, se normalizan los valores de sus perfiles diarios representativos. En este apartado tiene una gran influencia el método de normalización elegido a la hora de formar los clústeres como se comprobará a continuación.

A la hora de seleccionar el número de clases, se representa el error en función del número de clases. Como se comentó en la Figura 4, un número razonable debería estar entre 6 y 9 clases. De todas maneras como se explicará más adelante, utilizaremos otras medidas como el CDI y el MIA además de los resultados de los algoritmos de predicción para elegir el número de clases. En el siguiente paso se ejecuta el algoritmo de clasificación K-means de cual obtenemos dos outputs los labels y los centroides. Con el comando *kmeans.labels_* obtenemos el label o número que identifica la clase a la que pertenece cada consumidor mientras que con el comando *kmeans.cluster_centers_* obtenemos los centroides que representan los perfiles diario representativos medios de los usuarios clasificados en una misma clase.

Acto seguido, se puede calcular como de compactos son los clústeres utilizando las medidas MIA y CDI definidas en el Capítulo 2 para comprobar la precisión de nuestro modelo. Por último quedaría visualizar la forma de los centroides y de los elementos dispuestos en las mismas clases con la idea de poder identificar los distintos tipos de consumidores típicos presentes en nuestra base de datos. Para concluir también se ha comprobado con que probabilidad los usuarios se mantienen en los mismos clústeres para todas las estaciones del año.

4.2. Selección del número de clases

En la Tabla 4 se representan los valores de MIA y CDI para los clústeres de primavera y verano. En general a medida que se aumenta el número de clases K, el MIA disminuye mientras que el CDI aumenta excepto para K=9. Esto tiene sentido puesto que el MIA mide cómo de parecidos son los perfiles diarios representativos de los elementos de un mismo clúster por lo que si aumentas el número de clases consigues clústeres más compactos. Por otro lado el CDI al final mide como de diferenciados son los clústeres por lo que si aumentas el número de clases habrá clústeres más parecidos.

Tabla 4. MIA y CDI para los clústeres de verano e invierno dependiendo de número de clases

Compacidad de los clústeres				
	Primavera		Verano	
clústeres	MIA	CDI	MIA	CDI
6	0.5637	0.9236	0.5555	0.9277
7	0.5541	0.9335	0.5491	0.9471
8	0.5402	0.9475	0.5260	0.9448
9	0.5345	0.9246	0.5208	0.8776

Parece ser que K=9 es la mejor respuesta respecto a la compacidad de los clústeres pero dificultará en gran medida la precisión de las predicciones. Para K= 8 los resultados son aun peor que para K=9. Entre K=6 y K=7, se decide que con una clúster adicional será más fácil aislar a los elementos con comportamientos inusuales. Por todas estas razones K=7 es la respuesta final. Todos estos valores han sido calculados utilizando normalización Z y frecuencia de muestreo de 15 minutos puesto que como se verá más adelante dan los mejores resultados.

4.3. Visualización de los clústeres

En esta sección se analizan a los consumidores típicos según las distintas estaciones del año, según la frecuencia de muestreo de los datos y según el tipo de normalización empleada. Las mayores diferencias entre los clústeres se producen debido al tipo de normalización escogida.

4.3.1. Clústeres en función de la técnica de normalización

En este trabajo se han probado dos técnicas de normalización: Z-normalización y normalización por potencia pico. El la Z-normalización obtenemos un perfil de media aproximadamente cero y desviación tipo aproximadamente igual a uno mediante la siguiente ecuación:

$$x_i^{'} = \frac{x_i - \mu}{\sigma} \tag{14}$$

Con la normalización por potencia pico, dividimos cada valor de potencia de los perfiles por la potencia máxima de la serie temporal. De tal manera que todos los valores de los perfiles diarios representativos se encuentran en el rango 0-1. A continuación se comparan los centroides para primavera y verano en función de la técnica de normalización seleccionada para una frecuencia de muestreo de 15 minutos.

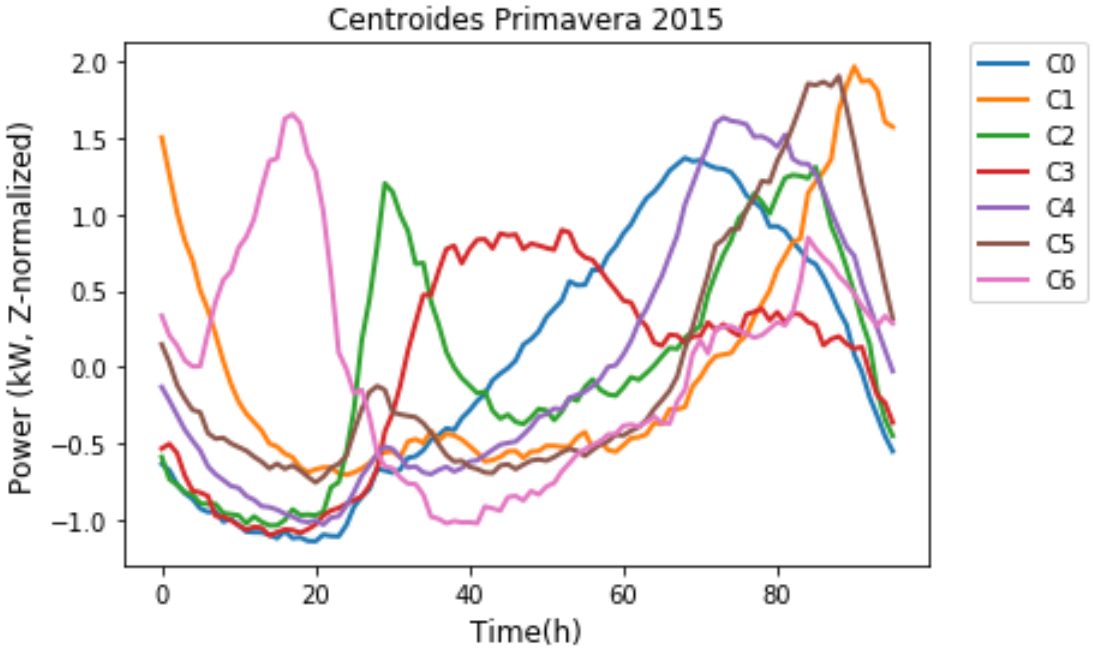


Figura 10. Centroides usando normalización Z para primavera

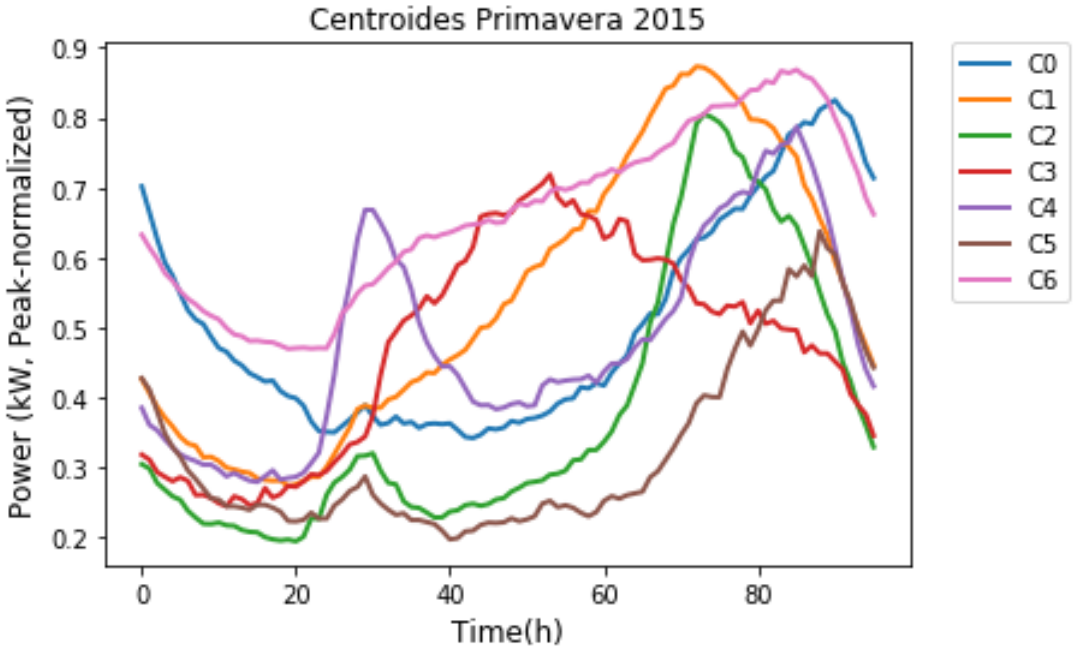


Figura 11. Centroides usando normalización pico para primavera

Como se puede observar en las Figuras 10 y 11, las clases formadas son muy distintas. Para decidir por uno de los dos métodos se debe observar como de compactos son los clústeres formados por cada técnica. A continuación también se muestran los centroides para verano.

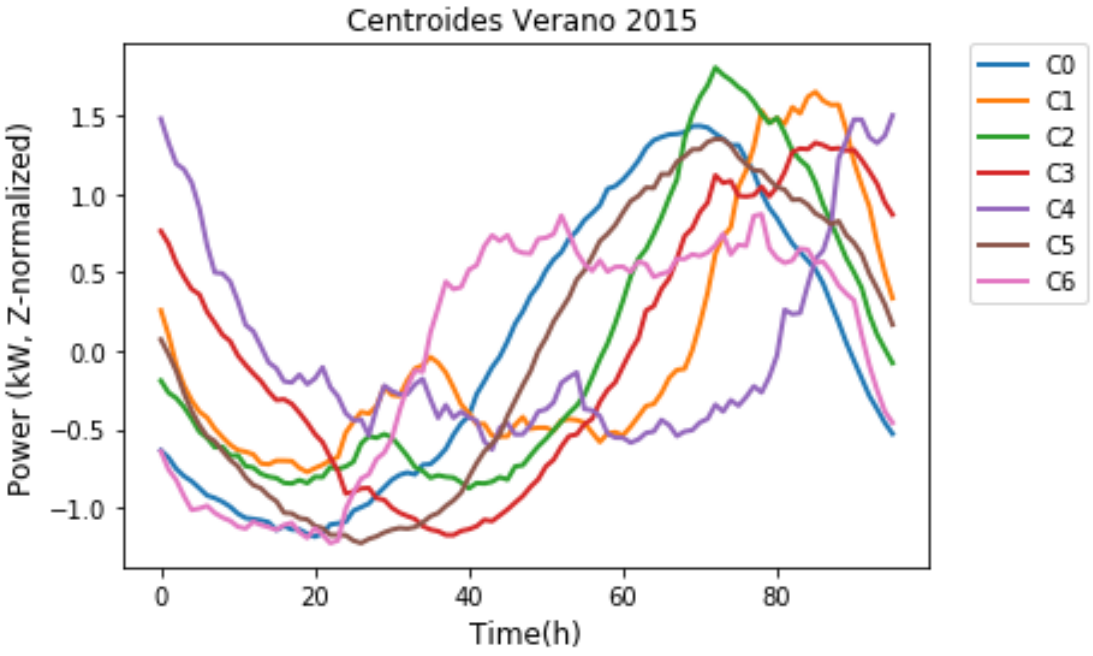


Figura 12. Centroides usando normalización Z para verano

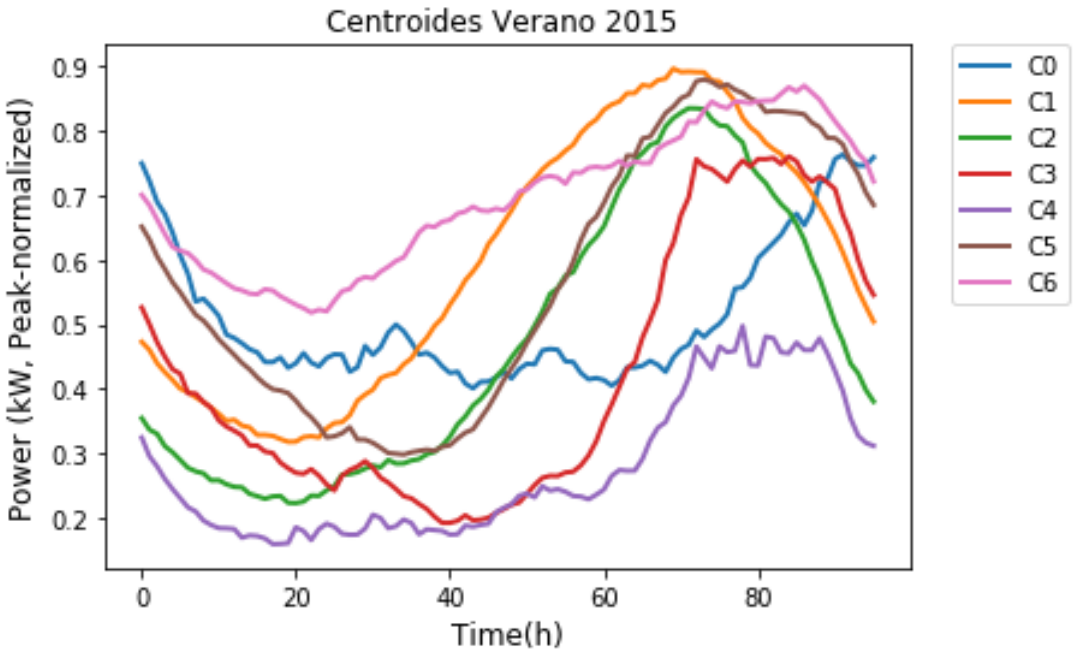


Figura 13. Centroides usando normalización pico para verano

Los centroides de verano están aun más diferenciados. Si se observa cada clúster por separado se llega a la conclusión de que los clústeres formados con normalización Z son mucho más compactos que los formados por normalización pico. A continuación se compara cada clúster

individualmente para observar el hecho de que los clústeres con normalización Z son más compactos y darán un mejor resultado a la hora de predecir. Con mostrar únicamente tres de los clústeres de verano es suficiente para corroborar este hecho. A la izquierda se encuentran los clústeres con normalización Z mientras que a la derecha son los de normalización pico.

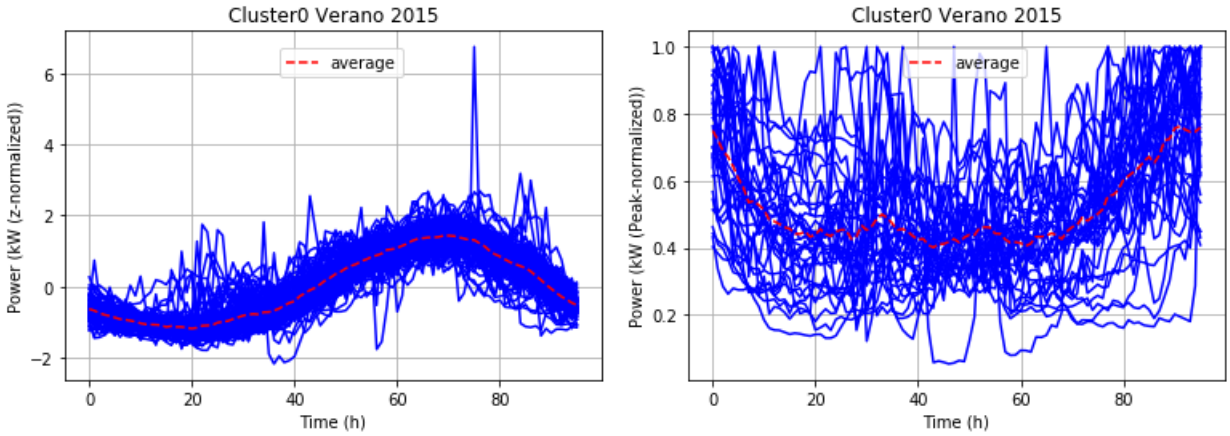


Figura 14. Clúster 0 según el tipo de normalización

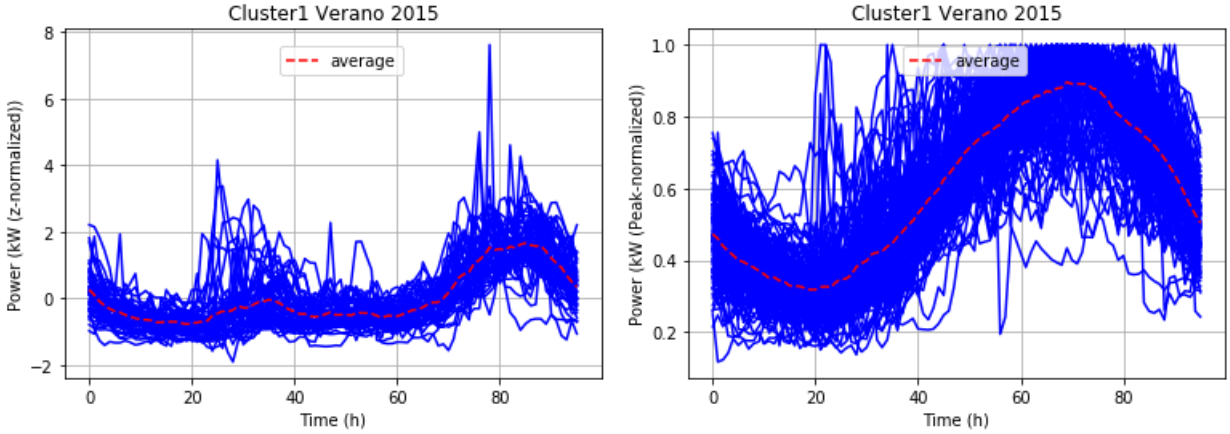


Figura 15. Clúster 1 según el tipo de normalización

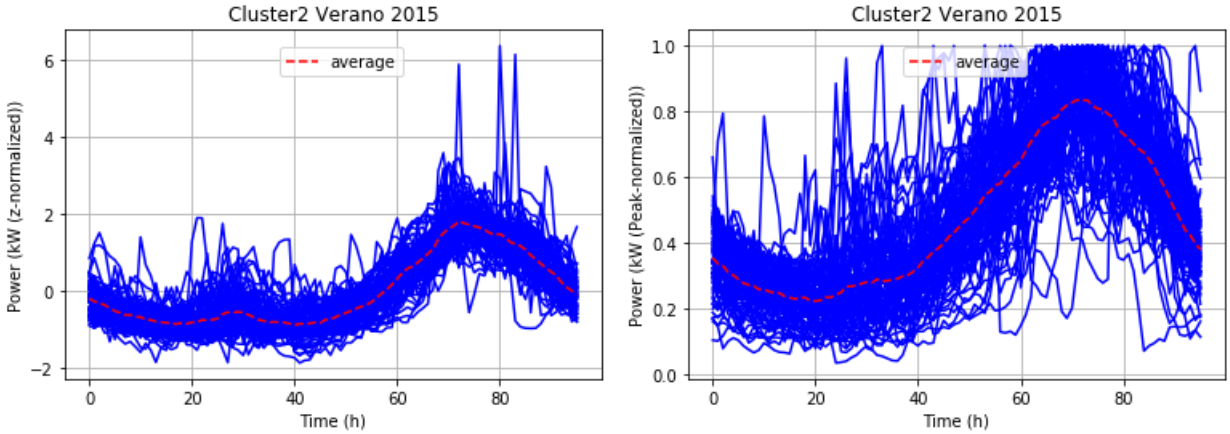


Figura 16. Clúster 2 según el tipo de normalización

4.3.2. Clústeres según el periodo de muestreo

Como se ha ido comentando a lo largo de este trabajo se dispone de dos periodos de muestreo, 1 hora o 15 minutos. Uno de los problemas de K-means es que clasificaría dos time series idénticas en dos clústeres distintos si las curvas tienen un mínimo desfase puesto que para calcular las distancias considera cada punto como una variable independiente (consumo de potencia en la hora 0, consumo de potencia en la hora 1...). Los perfiles muestreados con una frecuencia de 15 minutos son más sensibles a este problema. En las Figuras 17 y 18 se presentan los centroides para primavera y verano en función del periodo de muestreo utilizando normalización Z. A la izquierda los centroides están muestreados cada hora mientras que a la derecha cada 15 minutos.

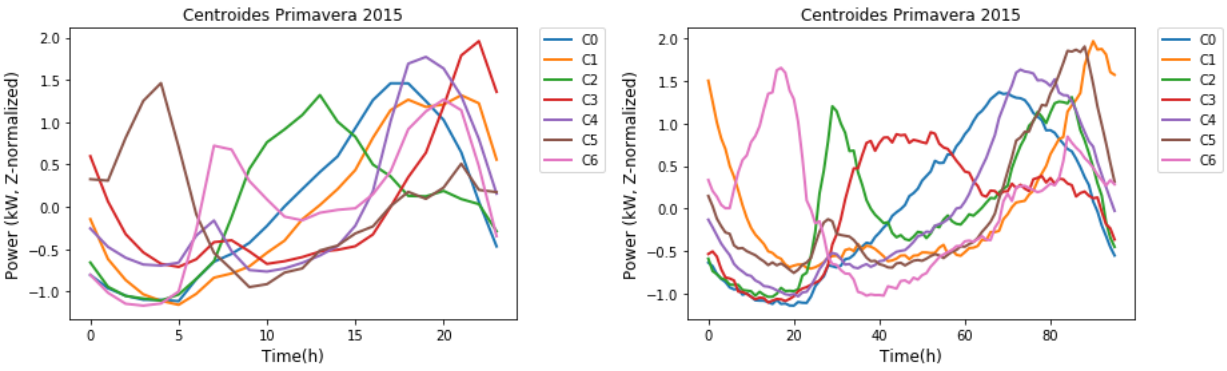


Figura 17. Centroides Primavera 2015 según la frecuencia de muestreo

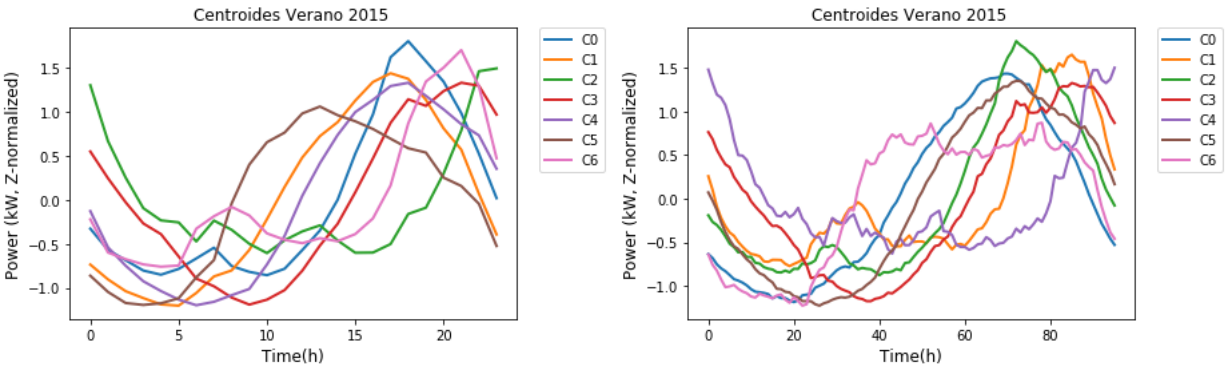


Figura 18. Centroides Verano 2015 según la frecuencia de muestreo

Como se puede observar en la Figuras 17 y 18, las clases que se forman son muy parecidas para ambas frecuencias de muestreo. Puesto que actualmente el sector suele utilizar datos de consumo medidos cada 15 minutos de aquí en adelante se utilizarán perfiles diarios representativos cuyos datos de potencia hayan sido medidos cada 15 minutos.

4.4. Análisis de los clústeres

En esta sección se analizarán los clústeres formados para el periodo de invierno. La frecuencia de muestreo escogida son 15 minutos y la normalización empleada es Normalización-Z puesto que de aquí en adelante se trabajará siempre con esos 2 parámetros. La figura 19 muestra los centroides para la primavera de 2015.

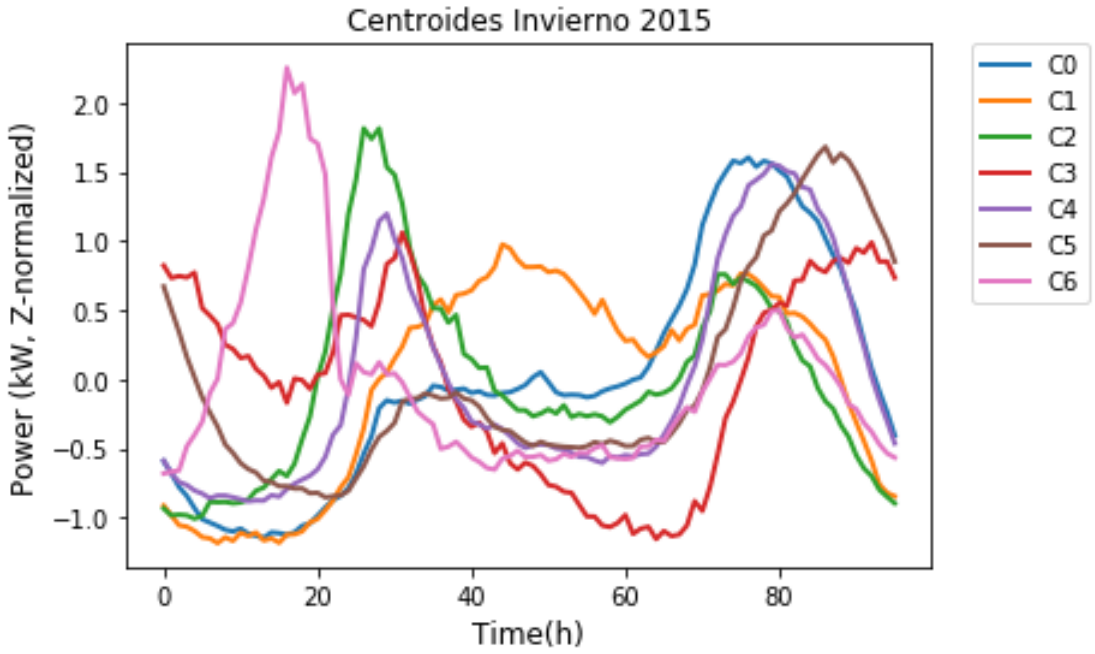


Figura 19. Centroides Invierno 2015

La caracterización de consumidores permite encontrar los 6-7 comportamientos típicos en el consumo de energía diario. Una vez que los comportamientos quedan definidos se pueden diseñar tarifas personalizadas para cada clúster que minimicen la tarifa diaria eléctrica. De esta manera los consumidores reducen su tarifa y las comercializadoras reducen sus costes de desvío puesto que conocen mejor a sus clientes. Además del diseño de tarifas, la caracterización de consumidores se puede emplear para realizar estudios de simultaneidad entre clases y así reducir el exceso de capacidad en los equipos de la red de distribución como transformadores o líneas. La segmentación de los clientes a través de K-means puede ser también empleado en previsión de demanda como se verá mas adelante en este trabajo.

En cuanto a las clases formadas, no todas tienen el mismo número de elementos y no todas son igual de compactas. A continuación en la Figura 20 se observarán los clústeres con mayor detalle y se obtendrán medidas para determinar como de compactos son cada uno de ellos.

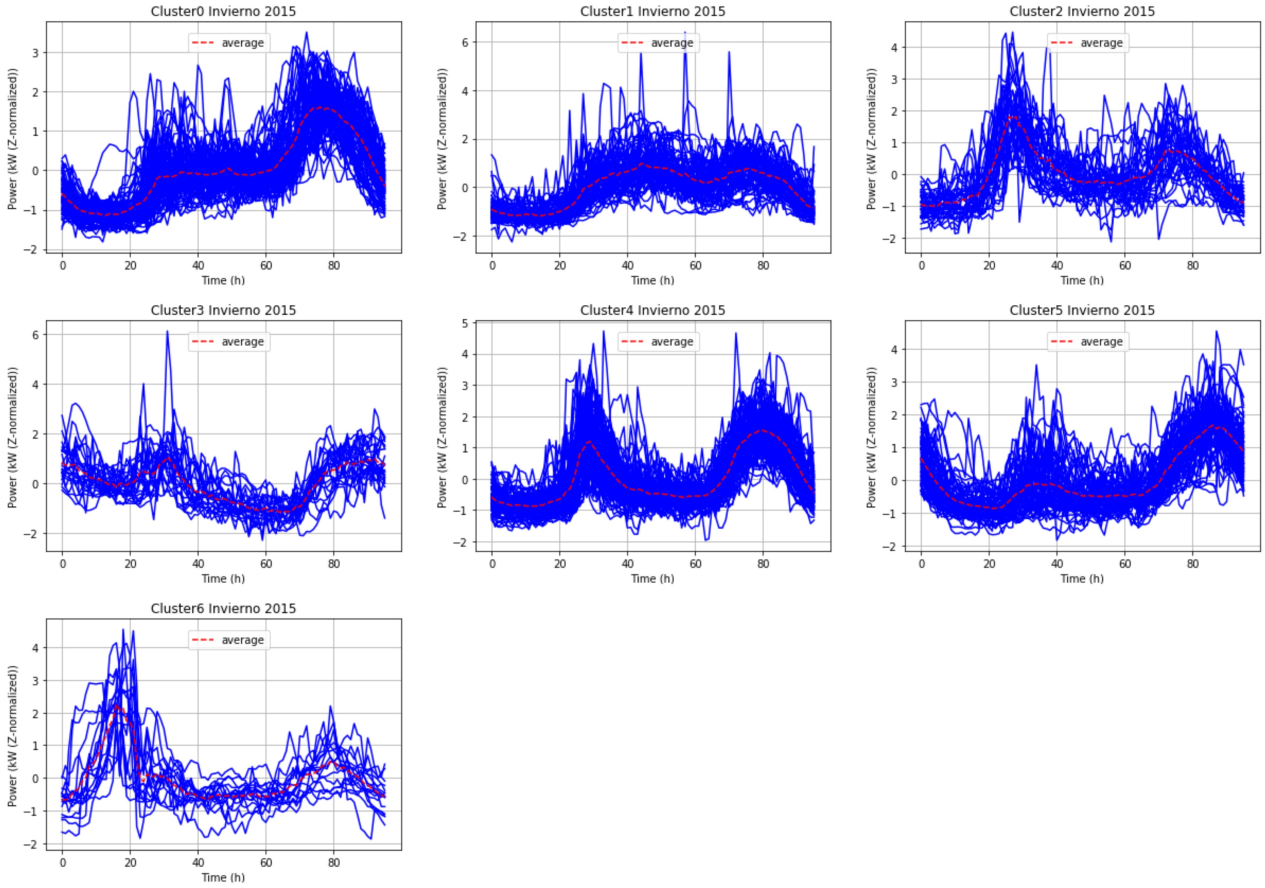


Figura 20. Centroides Invierno 2015

Tabla 5. MIA y número de elementos por clúster para el periodo de Invierno 2015

Análisis de los clústeres Invierno 2015							
Cluster	0	1	2	3	4	5	6
MIA	0.521	0.624	0.665	0.668	0.560	0.593	0.681
n elementos	82	69	36	23	99	72	16

A través de la Tabla 5 y la Figura 20 podemos determinar que el clúster número 6 recoge a aquellos elementos con un comportamiento inusual y que no pueden ser clasificados en ninguna otra clase. Por esta razón el clúster 6 presenta una mayor dispersión que el resto de los clústeres como se puede observar en la medida MIA. Además es el clúster que presenta el menor número de elementos. Por estas razones el clúster 6 debería ser suprimido de este estudio. De está misma manera se actuaría sobre los clústeres de verano.

4.5. Análisis de permanencia en los mismos clústeres según la estación del año

Una de las hipótesis con las que se partía en este trabajo es que los elementos agrupados en un mismo clúster para una estación determinada mantendrían su comportamiento para el resto de estaciones y por lo tanto los mismos elementos quedarían agrupados en los mismos clústeres para todas las estaciones de año. Las últimas líneas del archivo *ClusteringV2.py* tratan de cuantificar este hecho. Para poder cuantificar este hecho, se ha comprobado elemento a elemento con quién compartía clúster en la estación de primavera respecto a con quién comparte clúster en la estación de verano. También se ha comprobado invierno-verano. Los resultados para primavera-verano determinan que un $37\,\%$ de los elementos comparten los mismos clústeres. Para invierno-verano únicamente un $22\,\%$ de los elementos comparten mismos clústeres. Dicho esto se puede descartar la hipótesis de que los elementos se mantendrán en los mismos clústeres para todas las estaciones del año.

Capítulo 5

Personalización de tarifas eléctricas

En este capítulo se proponen diferentes tipos de tarifas eléctricas que se adapten a los hábitos de consumo de los usuarios de los que se disponen. Al adaptarse a la curva de demanda de los consumidores, estas tarifas tienen el objetivo final de minimizar la factura diaria eléctrica de cada consumidor. Aunque las comercializadoras vean reducidas sus ganancias, el hecho de tener tarifas personalizadas permite a las comercializadoras conocer mucho mejor cuándo y cómo consumen sus clientes de manera que puede reducir sus costes de desvío a la hora de acudir al mercado eléctrico. Por lo tanto se da una situación en las que ambas partes ganan.

Para este capítulo se empleará una base de datos reducida con respecto a la base de datos empleada en anteriores capítulos de tal manera que solo se centra en los usuarios de Austin, Tejas. De esta manera se pueden conseguir unos clústeres más compactos y que representan mejor los patrones de consumo de los usuarios. Por lo tanto la base de datos es reducida a 358 consumidores. El diseño de las tarifas ha estado basado en los precios reales propuestos por la compañía Austin Energy en su tarifa fija y su tarifa con discriminación horaria (TOU) [9]. Mientras que la definición de las horas promocionadas y penalizadas esta basado en las nuevas tarifas con discriminación horaria propuestas por Iberdrola S.A. [10].

5.1. Personalización de tarifas para primavera

El primer paso es obtener las clases de consumidores típicos presentes en nuestros datos en base a las tarifas eléctricas que se pretenden diseñar. La idea es que a cada una de las clases le corresponda una tarifa eléctrica diferente. Para el estudio de las clases se emplearán las estaciones de primavera y verano. A continuación en la Figura 21 se representan las clases para la estación de primavera. El número de clases seleccionado será 7 puesto como se demostró en capítulos anteriores supone la mejor elección.

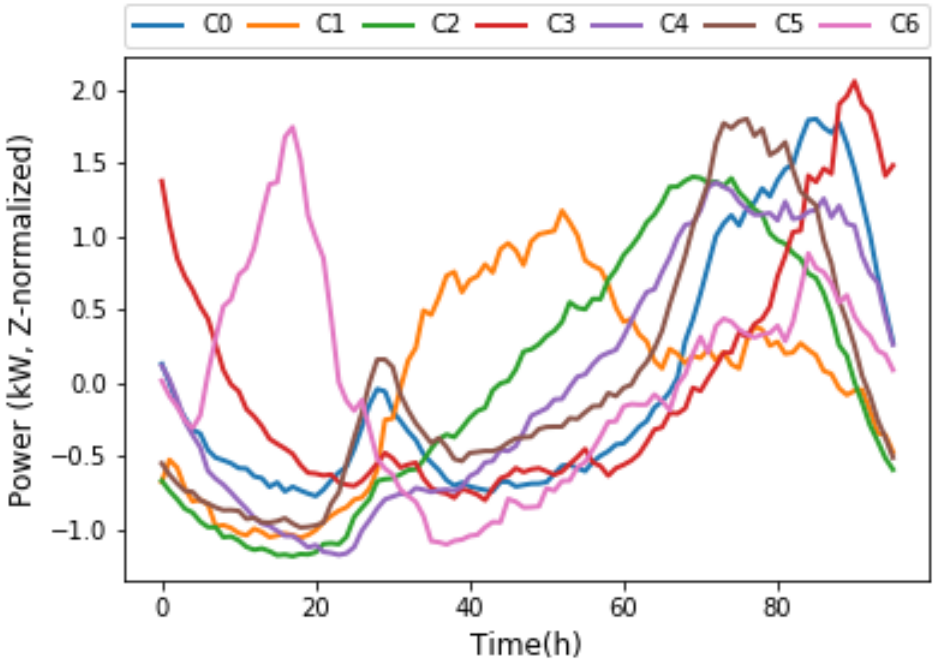


Figura 21. Representación de los centroides de los consumidores de Austin durante los días laborables de Primavera 2015 con una frecuencia de muestreo de 15 minutos

Una vez que las clases son definidas, la idea es crear tarifas eléctricas que se adapten a la forma de los centroides de cada clase. Estas tarifas que se adaptan a los comportamientos de los consumidores eléctricos pueden ser implementadas gracias a la adopción masiva de contadores inteligentes que permiten a las comercializadoras tener información en tiempo real del consumo eléctrico de los clientes. La mayoría de estas tarifas están basadas en el principio de ofrecer precios bajos durante ciertas horas del día en las que la demanda eléctrica es baja mientras que en las horas pico de demanda se ofrecen precios altos. La Tabla 6 describe los distintos tipos de tarifas propuestas en este trabajo.

Tabla 6. Tarifas eléctricas propuestas para primavera

Nombre	Horas promocionadas	Precios ($/kWh)
Plan Fijo	–	$c_1 = 0{,}0384$ $c_2 = 0{,}0384$
Plan 8h	Elección del cliente 8h o 4h-4h (8 horas)	$c_1 = 0{,}0110$ $c_2 = 0{,}0580$
Plan Noche	Desde 10pm hasta mediodía (14 horas)	$c_1 = 0{,}0246$ $c_2 = 0{,}0465$
Plan TOU	Off-peak (8 horas) Mid-peak (16 horas)	$c_1 = 0{,}0082$ $c_2 = 0{,}0465$

El Plan Fijo es la clásica oferta en la que todas las horas de día tienen el mismo precio. Por otro lado, en el Plan 8h se le permite al cliente elegir las 8h horas del día en las que el precio de la electricidad estará promocionado. Las 8h deben ser elegidas en un único bloque seguido o en dos bloques de 4h-4h. Este plan no está basado en el principio de horas valle y horas pico, pero permite a las comercializadoras comprender mejor los hábitos de consumo de sus clientes de tal manera que reducen la impredecibilidad en el consumo de energía. El Plan Noche se aprovecha de las horas de demanda baja para ofrecer precios bajos mientras que el precio durante el día supera el precio fijado por el Plan Fijo en horas pico. Por último, el Time-Of-Use Plan hace una clasificación de las horas del día en off-peak (horas valle), mid-peak (horas de consumo medio) y on-peak (horas pico) ofreciendo distintos precios de acuerdo a esta clasificación. Esta clasificación de las horas del día es resumida en la Tabla 7.

Tabla 7. Clasificación de las horas para el TOU Plan según la estación del año

	off-peak	mid-peak	on-peak
Primavera	10pm-6am	6am-10pm	–
Verano	10pm-6am	6am-2pm and 8pm-10pm	2pm-8pm

Los precios para todas estas tarifas dependen de la estación del año, teniendo precios más altos durante los meses de verano cuando la demanda eléctrica alcanza sus picos de potencia anuales. Los precios presentados en la Tabla 6 no se corresponden del todo con los precios publicados por Austin Energy en [9] puesto que ellos no distinguen entre primavera y verano a la hora de dar un precio fijo en su tarifa fija. Sin embargo Austin Energy ofrece un precio kWh por tramos en los que los primeros kWh consumidos son notablemente más baratos que los últimos. Para evitar esta característica, el precio $c_1 = 0{,}0384$ \$/kWh ha sido calculado como la media ponderada de los precios de cada uno de los los tres tramos de precios teniendo en cuenta que el consumo medio residencial en Austin en primavera es de $761{,}75$kWh al mes. El precio fijo del kWh en verano cambia con respecto al de primavera porque el consumo medio residencial en verano es de $1170{,}75$kWh al mes. Por otro lado, los precios de la tarifa TOU propuestos por Austin Energy dependen de la estación del año y de los kWh consumidos hasta el momento. Por lo tanto los precios mostrados para TOU plan han sido calculados con medias ponderadas como se hizo anteriormente.

El siguiente paso sería calcular las tarifas ficticias diarias para cada centroide para comparar cuál de las tarifas propuestas minimizan la tarifa final. Este cálculo ha sido considerado ficticio puesto que los centroides de cada clase no muestran los valores reales de potencia instantánea consumida debido a la normalización. De todos modos, la tarifa que minimice la factura ficticia también minimizará la factura real. A la hora de comprobar el Plan 8h es necesario realizar una media móvil de N=4 para cada clase con el objetivo de asegurarse que se le asigna a los clientes el bloque de 8 o 4 horas que minimiza su factura diaria. El bloque de 8h óptimo contendrá los

dos valores máximos de la media móvil separados por al menos 4 horas. La Figura 22 presenta las medias móviles para las clases de primavera con una frecuencia de 15 minutos entre cada dato.

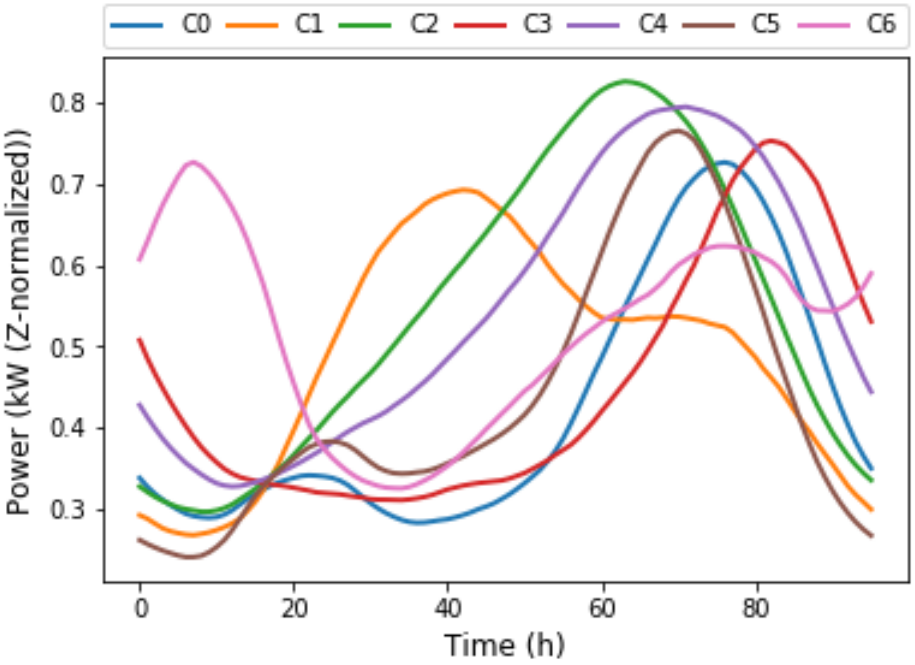


Figura 22. Representación de las medias móviles (N=4) de consumo para las clases de primavera con una frecuencia de 15 minutos

La Tabla 8 muestra las facturas diarias ficticias para los diferentes centroides mostrados en la Figura 21. El verde representa la factura más barata mientras que el rojo representa la más cara. El resto de colores representan la diferencia con la opción más barata.

Tabla 8. Facturas eléctricas ficticias

Cluster	Fija	8h	Noche	TOU
0	0.3898	0.3640	0.3605	0.3501
1	0.4492	0.4459	0.4160	0.4446
2	0.5005	0.4775	0.4839	0.4955
3	0.4147	0.4217	0.3666	0.3376
4	0.5031	0.4878	0.4751	0.4656
5	0.4009	0.3755	0.3849	0.3939
6	0.4772	0.4665	0.4203	0.3842

La mayoría de las comercializadoras eléctricas solo ofrecen dos tipos de tarifas la fija y la TOU puesto que estas son las más sencillas de diseñar. Como se puede observar en la Tabla 8 la tarifa Fija nunca es elegida y debe ser revisada puesto que ofrece precios demasiados altos en comparación con el resto de tarifas. El Plan 8h encaja a la perfección para aquellos clientes que usan energía durante las horas pico porque usando otras tarifas como la TOU o el Plan

Noche estarían fuertemente penalizados. El Plan Noche y la tarifa TOU están basadas en el principio de horas pico y valle, la diferencia que da cierta ventaja al Plan Noche es aprovechada por aquellos consumidores que consumen entre las 6 am y mediodía puesto que estas horas están promocionadas por el Plan Noche mientras que son consideradas de consumo medio por el Plan TOU. Un claro ejemplo de esta caso es el Clúster 1.

5.2. Personalización de tarifas para verano

De la misma manera que se ha hecho en primavera, la Figura 23 representa las clases de consumidores para verano.

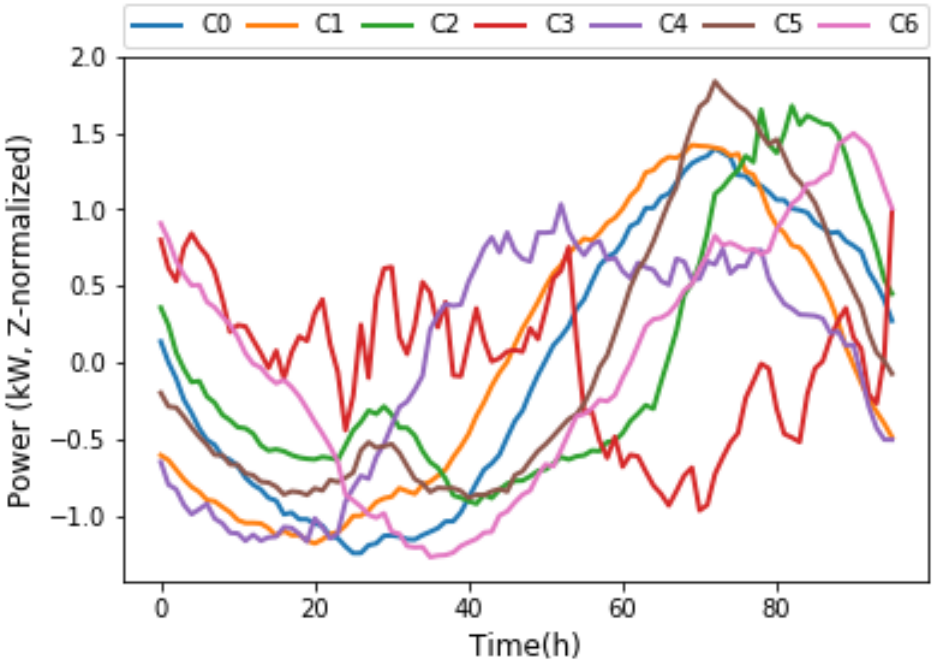


Figura 23. Representación de los centroides de los consumidores de Austin durante los días laborables de Primavera 2015 con una frecuencia de muestreo de 15 minutos

El C3 agrupa a aquellos consumidores con un comportamiento inusual y que deberían ser suprimidos de este estudio. Los precios entre primavera y verano varían debido a que en verano se alcanzan los máximo anuales por lo que el precio del KWh es significativamente más alto. La Tabla 9 presenta los precios definidos para verano. Teniendo en cuenta estos precios parece ser que el Plan Fijo es la mejor elección de lejos para todas las clases.

Tabla 9. Tarifas eléctricas propuestas para primavera

Nombre	Horas Promocionadas	Precios ($/kWh)
Plan Fijo	–	$c_1 = 0{,}0483$
	–	$c_2 = 0{,}0483$
Plan 8h	Elección cliente	$c_1 = 0{,}0285$
	8h o 4h-4h (8h)	$c_2 = 0{,}0905$
Plan Noche	Desde las 10pm	$c_1 = 0{,}0309$
	hasta mediodía (14 horas)	$c_2 = 0{,}0872$
TOU plan	Off-peak (8 hours)	$c_1 = 0{,}0104$
	Mid-peak (10 hours)	$c_2 = 0{,}0585$
	On-peak (6 hours)	$c_2 = 0{,}1064$

La Tabla 10 muestra los precios de la tarifa fija que permitiría al resto de tarifas ser competitivas.

Tabla 10. Precios de la tarifa fija que haría al resto de tarifas competitivas

Clúster	Precio Tarifa Fija	Plan alternativo
0	$c_1 = 0{,}0602$	Plan 8h
1	$c_1 = 0{,}0590$	Plan 8h
2	$c_1 = 0{,}0559$	Plan 8h
3	$c_1 = 0{,}0494$	Plan TOU
4	$c_1 = 0{,}0598$	Plan Noche
5	$c_1 = 0{,}0541$	Plan 8h
6	$c_1 = 0{,}0536$	Plan TOU

5.3. Análisis de los resultados y conclusiones

Los precios de la electricidad fluctúan a lo largo del año alcanzando sus valores pico durante los meses de verano e invierno debido al consumo masivo de energía para enfriar y calentar las viviendas. Mientras tanto los precios para primavera y otoño son considerablemente más bajos. Los comercializadores de energía como Austin Energy que ofrecen tarifas fijas para todo el año deben tener en cuenta estas fluctuaciones a la hora de ofrecer un único precio para todo el año. Por esta razón, el precio fijo es más caro que el precio variable durante primavera pero es más barato durante verano. Dependiendo de la comercializadora, los contratos pueden tener una duración de 12 meses en los cuales tienes que mantener la misma tarifa. El comercializador ideal debe permitir al consumidor ajustar su tarifa en función de la época del año. Los precios fijos deberían cambiar también función de la época del año.

En el ejemplo mostrado, se consigue un ahorro medio del $10{,}57\,\%$ sobre el precio fijo durante primavera. Estos ahorros medios son calculados teniendo en cuenta los ahorros de cada clase. Los ahorros de cada clase son calculados como la diferencia entre el Plan Fijo y el plan que minimice la factura de cada clase. Por otro lado, parece que en verano el consumidor no obtendrá ningún tipo de beneficio escogiendo una tarifa variable.

Todo los procedimientos mostrados en este capítulo han sido implementados en el archivo de python *Tarifa_V2.py* cuyo código es recogido en la sección de código fuente de este trabajo.

Capítulo 6

Obtención de los GAFs y MTF a partir de los perfiles diarios representativos

En este capítulo se pretende explicar las funciones utilizadas para la obtención de los distintos tipos de imágenes a partir de los perfiles diarios representativos. Por lo tanto este capítulo resume las acciones del archivo de código *MTF-GAF_V6.py* que se puede encontrar en la parte de Código Fuente de este al final de este trabajo. Toda la teoría acerca de las definición de los MTF y GAFs ha sido explicada en el Capítulo 2 Metodología por lo que este capítulo no incluirá definiciones teóricas.

Para la obtención de los GAFs y MTFs se utilizará la librería de Python Pyts [7] puesto que facilita enormemente la obtención de las imágenes. La figura 24 muestra la estructura de las operaciones que se ejecutan en *MTF-GAF_V6.py*. El primer paso es cargar los perfiles diarios representativos seleccionando la estación con la que se quiere trabajar. Unas vez seleccionado el número de clases, la frecuencia de muestreo y el tipo de normalización, se normalizan los datos. La primera función que se ejecuta es *MTF(image_size=,n_bins, quantiles, overlapping)* donde se especifican los parámetros de la MTF para luego ejecutarlos sobre la serie temporal con el comando *mtf.transform(time series)*. En el parámetro image_size se especifica el tamaño de la imagen cuadrada. Si se selecciona un tamaño de la imagen inferior al tamaño de la serie temporal se aplicarán técnicas de blurring kernel (difuminar el núcleo) para reducir los píxeles de la imagen con esto lo que se consigue es reducir la potencia computacional requerida para procesar las imágenes. El número de quantile bins es uno de los parámetros a especificar en los MTF. Cualquier número entre 10 y 20 es óptimo en este caso, conclusión que ha sido obtenida tras probar numerosas opciones. En el parámetro quantiles se puede seleccionar empirical o gaussian haciendo referencia a la forma de separar los datos en quantiles. Con empirical se separan los datos en quantiles de manera que todos los quantiles tengan el mismo número de elementos. Por otro lado con gaussian se busca que todos los quantiles tengan el mismo tamaño. La Figura 25 muestra la diferencia entre gaussian arriba y empirical abajo. Para este trabajo se ha usado siempre empirical a la hora de separar en bins. Por último para la variable overlapping

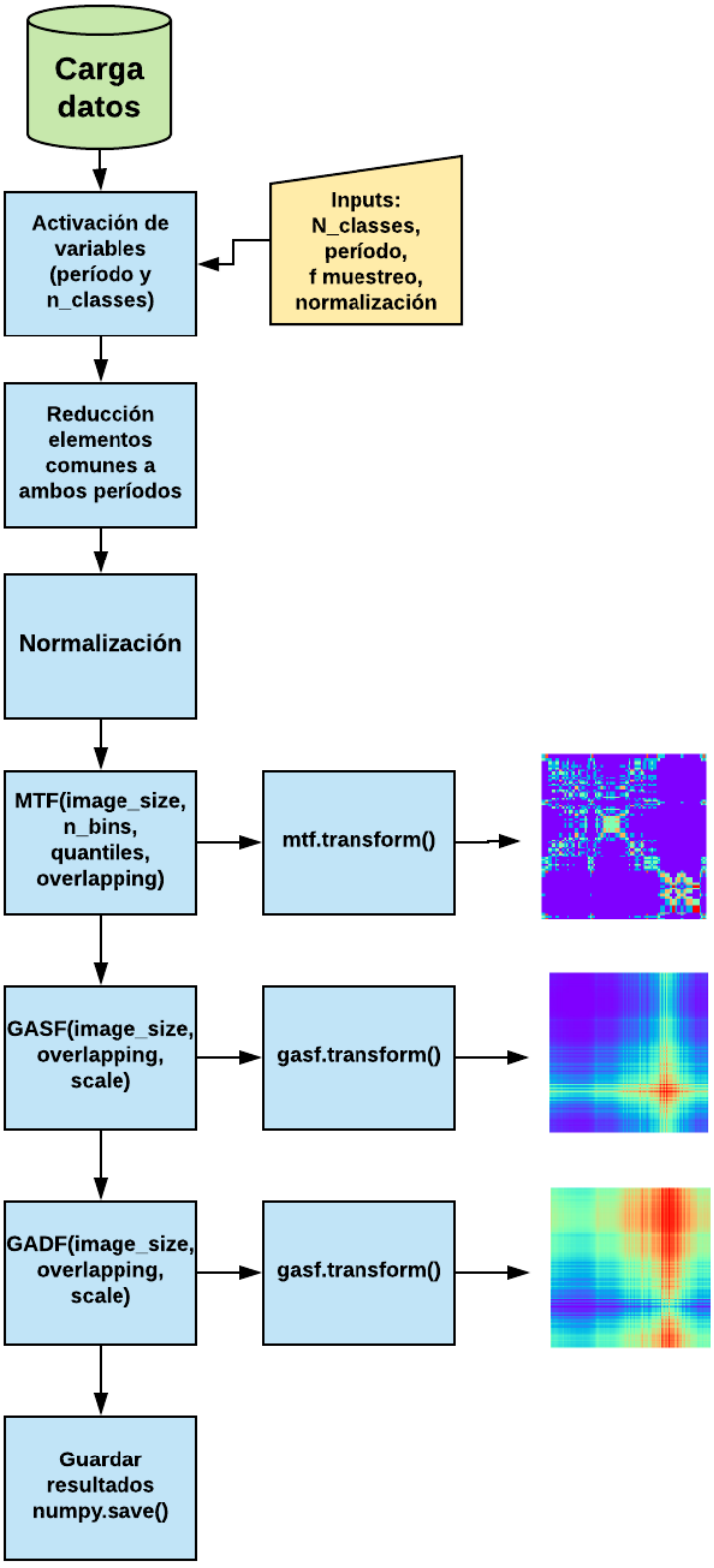


Figura 24. Estructura *MTF-GAF_V6.py*

se puede seleccionar True o False para que los valores de la serie temporal puedan pertenecer a dos quantiles diferentes o no. En este caso se ha seleccionado siempre False.

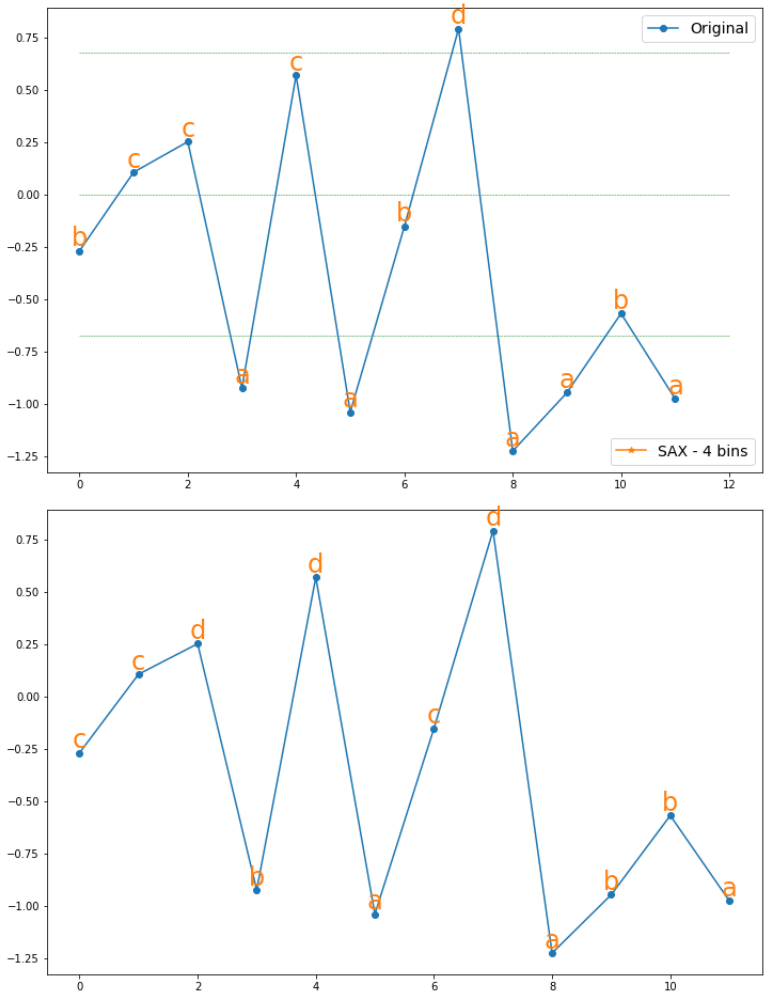


Figura 25. Diferencias entre la separación de quantiles 'gaussian' y 'empirical'

Una vez que hemos separado la serie temporal en quantiles podemos obtener la matriz de probabilidades de transición entre quantiles W y a partir de esta obtener la matriz M. Todo esto se encuentra ya implementado en la función mtf.transform(). Los GASF y GADF se obtienen de esta misma manera a partir de las funciones gasf.transform() y gadf.transform() con la salvedad de que no se especifica número ni tipo de quantiles. Lo que si queda especificado es que se escale la serie temporal entre 0 y 1 a través de la variable 'scale=0' para poder reconstruir la serie temporal a partir de la diagonal los GASFs.

Capítulo 7

Redes Neuronales Convolucionales

En este capítulo se pretende explicar uno de los algoritmos utilizados en la clasificación y predicción de consumidores eléctricos. Se comentará la utilidad del algoritmo empleado así como su estructura. También se comentará el código *DemandForecasting.ipynb* en el que se implementa el algoritmo. El agoritmo es implementado gracias a la librería Keras [11] que trabaja sobre TensorFlow permitiendo crear complejos grafos a través de líneas de código en Python. Más adelante se comentarán y compararán los resultados obtenidos frente al desempeño obtenido por el algoritmo K-means.

Una red neuronal convolucional es un tipo de red neuronal artificial. Este tipo de red es una variación de un perceptron multicapa, sin embargo, debido a que su aplicación es realizada en matrices bidimensionales, son muy efectivas para tareas de visión artificial, como en la clasificación y segmentación de imágenes, entre otras aplicaciones [12]. El éxito de las Redes Neuronales Convoluciones viene de la mano del desarrollo de las GPU que permiten implementar y ejecutar complejas estructuras obteniendo resultados rápidamente. En este trabajo se empleó el superordenador Maverick utilizando los recursos del Texas Advanced Computing Center (TACC) [13]. Gracias a esta gran potencia computacional se han podido probar diferentes tipos de entradas y comprobar su desempeño. Entre otras cosas se ha probado utilizar las diferentes imágenes (MTF, GASF y GADF) de forma individual y combinada obteniendo los mejores resultados en la combinación de las tres imágenes. También se ha probado a variar el número de quantiles en los MTF obteniendo los mejores resultados con Q=16. También se ha comprobado el desempeño del algoritmo en función del número de clústeres que es mostrado en el capítulo de análisis de resultados.

En las CNNs se consigue reducir el número de parámetros gracias a que las neuronas de las layer de convolución están únicamente conectadas a una red NxN marcada por el tamaño de los filtros. Del tal manera que esta red comparte los valores de los weights. Este enfoque tiene una serie de ventajas entre las que destaca la reducción de parámetros del modelo permitiendo

utilizar menores potencias computacionales. Con respecto a las redes neuronales simples tiene la ventaja de que se puede entrenar la red con menos datos [14]

7.1. Objetivo

Como se puede observar en la Figura 26, se pretende utilizar la combinación de imágenes de los MTF+GAFs como input para clasificar a los clientes según las clases de primavera o verano, ya sea clasificación o predicción.

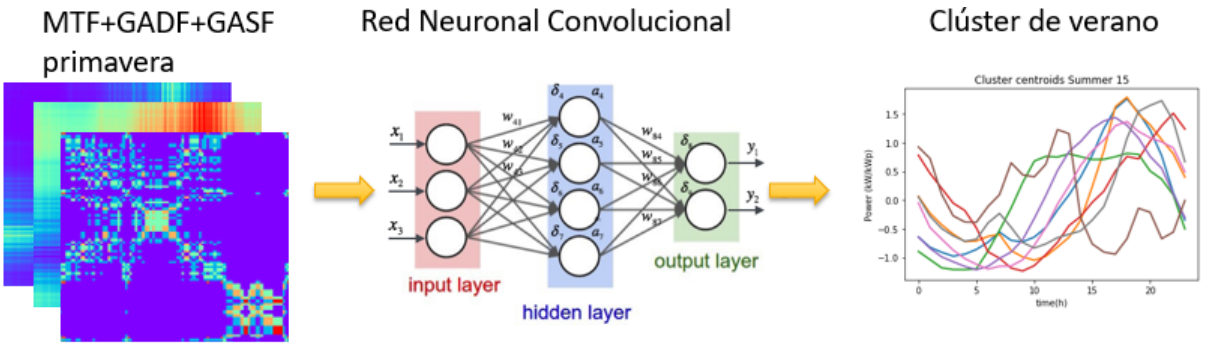


Figura 26. Objetivo de las redes neuronales

7.2. Arquitectura

Las redes neuronales convolucionales consisten en múltiples capas de filtros convolucionales de una o más dimensiones. Después de cada capa, por lo general se añade una función para realizar un mapeo causal no-lineal. Como redes de clasificación, al principio se encuentra la fase de extracción de características, compuesta de neuronas convolucionales y de reducción de muestreo. Al final de la red se encuentran neuronas de perceptron sencillas para realizar la clasificación final sobre las características extraídas [12]. La estructura empleada en este trabajo queda resumida en la Figura 27.

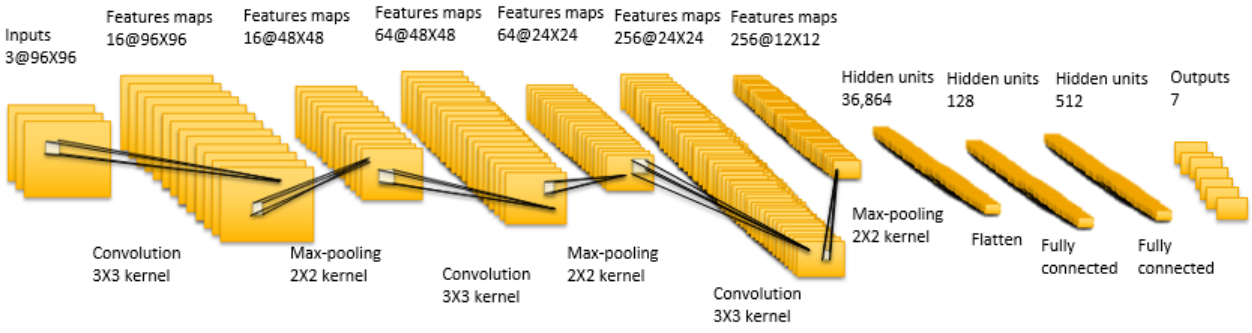


Figura 27. Estructura de las Redes Neuronales Convolucionales

Como se puede apreciar en la Figura 27, la capa de input está formada por las tres imágenes (MTF+GASF+GADF) de tamaño 96x96 píxeles. De tal manera que cada una de las imágenes es uno de los tres channels que forman el input. La fase de extracción de características se compone de capas alternas de neuronas convolucionales y neuronas de reducción de muestreo. Según progresan los datos a lo largo de esta fase, se disminuye su dimensionalidad, siendo las neuronas en capas lejanas mucho menos sensibles a perturbaciones en los datos de entrada, pero al mismo tiempo siendo estas activadas por características cada vez más complejas. En las capas de convolución se emplean siempre filtros con un núcleo de 3X3 píxeles de tal manera que se utiliza padding para que no se reduzca la dimensión. Se utilizan 3 capas de convolución cuyo número de filtros por capa va ascendiendo de 16 a 64 hasta llegar a 256. Entre cada capa de convolución se utiliza la función max-pooling para reducir el tamaño de la información que manejan las capas para que la información sea manejable a nivel computacional. Originalmente, las redes neuronales convolucionales utilizaban un proceso de subsampling para llevar a cabo esta operación. Sin embargo, estudio recientes han demostrado que operaciones como max-pooling, son mucho más eficaces en resumir características sobre una región. La operación de max-pooling encuentra el valor máximo entre una ventana de muestra y pasa este valor como resumen de características sobre esa área. Como resultado, el tamaño de los datos se reduce por un factor igual al tamaño de la ventana de muestra sobre la cual se opera. En este caso se utiliza una ventana de 2x2 por eso la información se reduce a la mitad. Las últimas capas de la estructura son las capas conocidas como fully connected puesto que juntan todos los píxeles de los filtros en un único array. Estas capas funcionan como redes neuronales artificiales sencillas de las cuales, las dos primeras capas utilizan como función de activación Rectifier Linear Units (ReLU) mientras que la capa que opera los scores utiliza la función de activación Softmax. La función Softmax se utiliza comúnmente para calcular los scores puesto que el output es equivalente a una distribución de probabilidad categórica, de tal manera que proporciona la probabilidad que tiene cada clase de ser verdadera. Si se tienen 7 clases, se obtendrán 7 scores diferentes de los cuales se mantendrá el más alto. En la última capa de outputs se computan los scores atribuyendo el elemento a la clase que alcance el máximo score.

7.3. Implantación en Python

Esta estructura de las redes neuronales ha sido implementado en Python gracias a librería Keras que trabaja sobre TensorFlow. La Figura 28 muestra el diagrama de bloques que describe el proceso para definir, entrenar y testear la estructura de red neuronal descrita en la Figura 27. La primera parte consiste en decidir con que periodo de los datos se quiere trabaja, cuál es el número de clases y el tipo de red convolucional que se implementa. En este caso se han definido dos modelos de Redes Neuronales Convolucionales la CNNs simple y la Tiled CNNs. Los mejores resultados han sido obtenidos con la CNNs simple por lo que será la usada en todos los resultados mostrados en próximos capítulos. A continuación se cargarán las imágenes de los

consumidores generadas a partir del archivo *MTF-GAF_V6.py* y se ordenarán de tal manera que el primer canal será el MTF, el segundo canal será el GADF y el tercer canal será el GASF. Se definirán los filtros de las capas de convolución que en este caso el número de filtros aumentará de 16 a 64 hasta llegar a 256. Se utilizará un kernel cuadrado de 3X3 píxeles para todas las capas de convolución. Por último el número de neuronas de las capas Fully connected irá de 128 a 512 para acabar en las 7 clases.

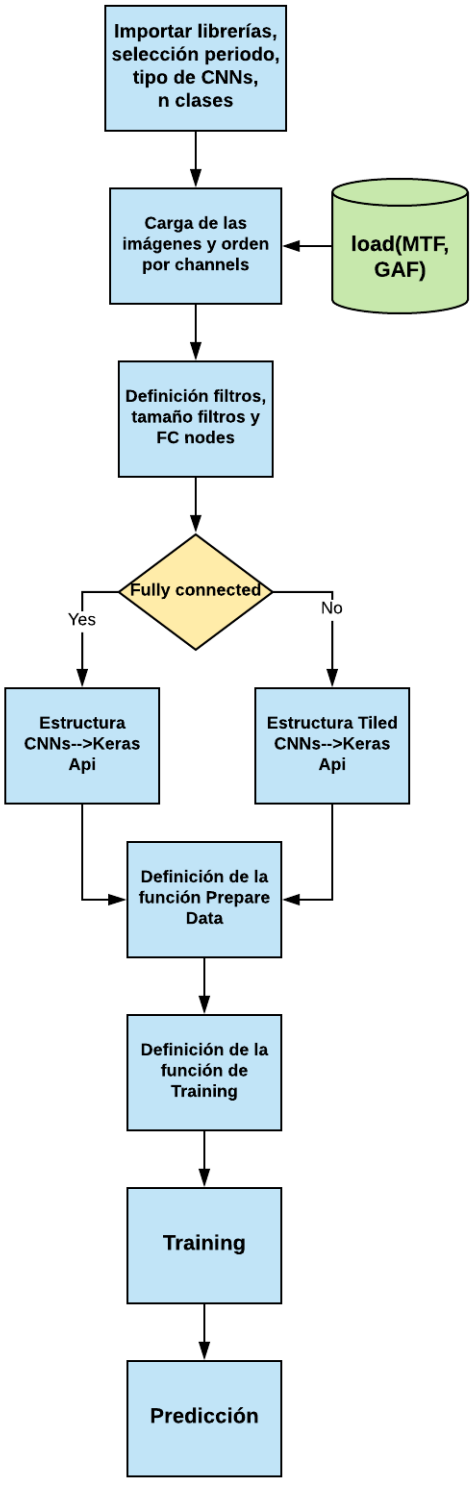


Figura 28. Estructura *DemandForecasting.ipynb*

Como se comentó anteriormente el modelo puede ser Convolucional simple o Tiled CNNs. La diferencia entre ambos modelos se diferencia el número de neuronas que comparten los mismos weights y que por lo tanto están conectadas entre si. De tal manera que en las Tiled CNNs también conocidas como Locally Connected, las neuronas comparten menos weights y por lo tanto el número de parámetros final es mucho mayor. Las layers de convolución son definidas según el Código 5.

```
# first convolution
input_layer_conv = Conv2D(filters=_FILTERS[0], kernel_size=(_KERNEL_SIZE
    [0],_KERNEL_SIZE[0]),strides=(1,1), padding="same", data_format="
    channels_first", name="input_conv", activation="relu")(input_layer)
# max pool
input_max = MaxPooling2D(pool_size=(2,2), strides=None, padding="same",
    data_format="channels_first", name="input_max")(input_layer_conv)
```

Codigo 5. Ejemplo layer de convolución y max pooling

Se utiliza la función CONV2D donde se definen el número de filtros, el tamaño de los filtros, los strides y el padding. Los strides controlan como los filtros realizan la convolución alrededor de un volumen de entrada. En el ejemplo mostrado en la figura 29, el filtro realiza la convolución alrededor del volumen de entrada moviendo su posición de uno en uno. Por lo tanto el stride ha sido fijado en 1. El ejemplo mostrado en la figura 29 muestra como de un volumen de entrada de 7x7 pasamos a un volumen de salida de 5x5 que representa las 5 posiciones posibles que ocupa el filtro [15].

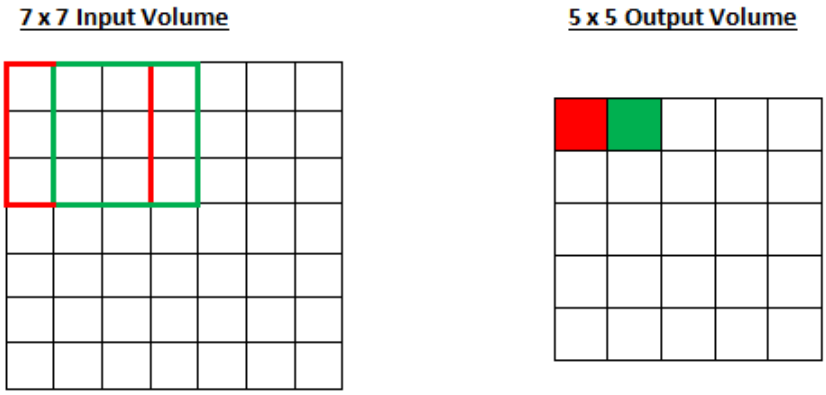


Figura 29. Ejemplo de strides = (1,1) en la capa de convolución [15]

Si pasamos a un stride de 2 el filtro se desplaza de 2 en 2 reduciendo aun más el volumen de salida como se puede observar en la Figura 30.

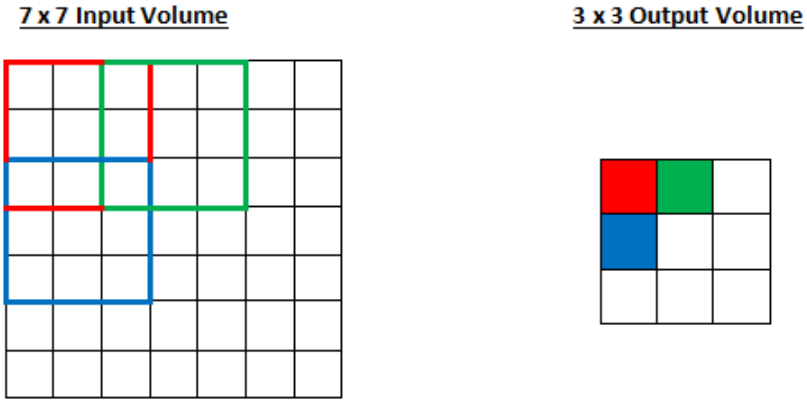


Figura 30. Ejemplo de strides = (2,2) en la capa de convolución [15]

El hecho de que el tamaño disminuya supone un problema puesto que se pierde demasiada información en las capas iniciales en las que se debe preservar el volumen de entrada lo máximo posible. Para solucionar este problema entra en juego el padding. Imagínese que se aplica un filtro de 5x5x3 a una imagen de 32x32x3. El volumen de salida con una stride=1 tendrá un tamaño de 28x28x3. Para evitar la reducción de tamaño podemos aplicar un zero padding de tamaño 2 que consiste en rellenar con ceros los bordes de la imagen, en este caso con dos ceros. En la Figura 31 se puede apreciar el zero padding de tamaño 2 [15]. Así transformamos el volumen de 32x32x3 en un volumen de 36x36x3 de manera que al aplicar los filtros de 5x5x3 mantengamos un volumen de 32x32x3

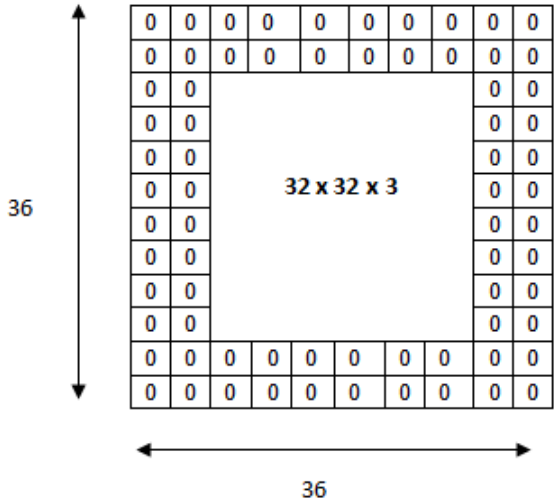


Figura 31. Ejemplo de zero padding de tamaño 2 en la capa de convolución [15]

El siguiente paso es la preparación de los datos donde se separan los datos en training set y test set así como los targets. También se mezclan los datos para asegurar aleatoriedad en el training. A continuación se define la función de training. A la hora de diseñar el training tenemos que definir la loss function y el optimizador. En este caso utilizaremos como loss function la Cross-Entropy y como optimizador utilizaremos el optimizador de ADAM para minimizar la

loss function lo más rápido posible. Para evitar overfitting en el trainig se utilizará early stop de tal manera que cuando la categorical accuracy cambie de signo o varíe menos que un delta se parará de entrenar la red.

Los últimos pasos ya serían simplemente entrenar la red y guardar el modelo en caso de ser los resultados satisfactorios. Una vez que tenemos el modelo entrenado podemos probarlo con nuevos clientes.

Capítulo 8

Clasificación utilizando K-means

En este capítulo se pretende explicar la utilización del algoritmo K-means con diferentes tipos de inputs para clasificar a los usuarios en función de la estación de año. También se pretende explicar los procedimientos implementados en el código *SpringSummer_Kmeans_Classification_imagesV2.py* en cual se prueba el algoritmo K-means con distintos tipos de inputs.

En la primera parte del código, utilizando únicamente el primer mes de la estación de primavera se pretende clasificar a un cliente en una de las clases de primavera, las cuales han sido definidas previamente utilizando también un algoritmo K-means pero que tiene en cuenta el consumo de los clientes durante los 3 meses de primavera. Para la segunda parte del código se pretende usar K-means para clasificar a los consumidores en las clases de verano utilizando su consumo del último mes previo a la estación de verano. Puesto que se utiliza información de un periodo diferente al que se pretende clasificar se trata de una predicción.

Se probará el algoritmo K-means con distintos inputs entre lo se destaca clasificación K-means con raw time series y clasificación K-means con imágenes en las que se probará la combinación MTF+GASF+GADF o las combinaciones GASF+GADF y MTF por separado. Los resultados de estas clasificaciones serán mostrados en el capítulo 9 donde se comparan los distintos procedimientos.

A continuación en la Figura 32 se representa el diagrama de bloques que explica el código del archivo *SpringSummer_Kmeans_Classification_imagesV2.py*. El primer paso es como en los demás archivos anteriores la activación de variables. Tras la activación de variables se cargan los datos de los periodos seleccionados, se reducen los datos a los elementos comunes a ambos periodos y se normalizan los datos. Para la clasificación de Primavera 2015 se va a probar el algoritmo K-means utilizando tres tipos de inputs diferentes como son los perfiles diarios representativos del primer mes de primavera en forma de serie temporal, como combinación de tres imágenes (MTF, GASF y GADF) y como combinación de dos imágenes (GASF y GADF).

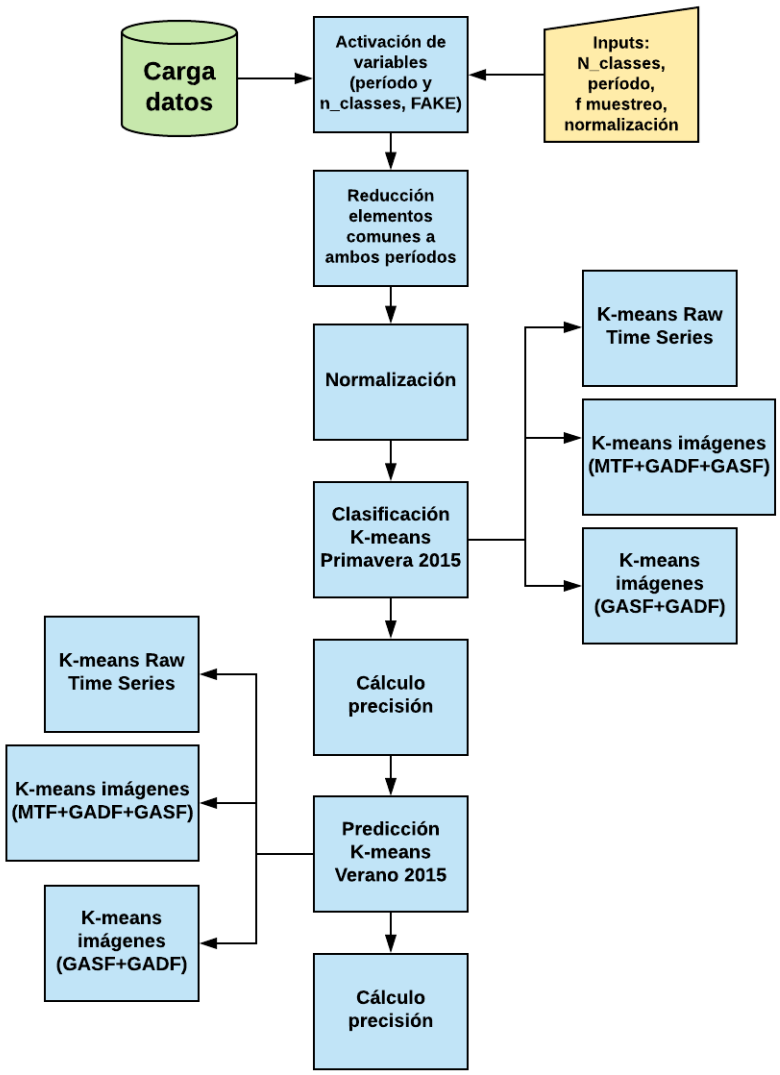


Figura 32. Estructura *SpringSummer_Kmeans_Classification_imagesV2.py*

Para el K-means con raw time series primero se crearán las labels de los consumidores utilizando los tres meses de Primavera 2015 sin embargo se clasificará utilizando únicamente el primer mes de Primavera 2015 mediante el comando kmeans.predict(). El Código 6 representa la clasificación empleando raw time series. Para la clasificación en formato imágenes es necesario que tras obtener las imágenes (MTF+GASF+GADF) están sean estructuradas como un único vector de 1D cuyo tamaño sea 96x96x3 (27648). Cuando únicamente se utilizan 2 imágenes el tamaño será 96x96x2. Esta forma de estructurar los inputs está recogida en el Código 7. El último paso es calcular el desempeño del algoritmo de clasificación. Para ello se comparan las labels reales con las que determinan los algoritmos. El cálculo de la precisión o accuracy queda recogida en el Código 8. La segunda parte de predicción del código se ejecuta de la misma manera que esta primera parte de clasificación con la diferencia de que se utiliza el último mes de primavera como input y se utilizan las clases definidas para Verano 2015.

```
#Kmeans for Time Series
kmeans = KMeans(n_clusters=N_CLUSTERS, random_state=0).fit(season2_use)
labels_spring_TimeSeries = kmeans.labels_
labels_spring_classified_TimeSeries = kmeans.predict(season2_1stmonth_use
    )
```

Codigo 6. Clasificación Primavera 2015 empleando el primer mes de primavera en formato raw time series como input

```
for i in range(X_mtf_1.shape[0]):

    fila = np.array([], dtype=float).reshape(0,size)
    fila = np.append(fila, [X_mtf_1[i,:,:].flatten(),X_gadf_1[i,:,:].
    flatten(),X_gasf_1[i,:,:].flatten()])
    m_1 = np.r_[m_1, [fila]]
```

Codigo 7. Estructura vector 1D para clasificación de imágenes

```
#K-means Accuracy
Common_labels_Images = np.sum(labels_spring_Images ==
    labels_spring_classified_Images)
Common_labels_TimeSeries = np.sum(labels_spring_TimeSeries ==
    labels_spring_classified_TimeSeries)

ACC_Spring_Images = Common_labels_Images/len(labels_spring_Images)
ACC_Spring_TimeSeries = Common_labels_TimeSeries/len(
    labels_spring_TimeSeries)

print("Accuracy Classification Time Series: %.4f" % (
    ACC_Spring_TimeSeries))
print("Accuracy Classification images MTF+GAFs: %.4f" % (
    ACC_Spring_Images))
```

Codigo 8. Cálculo desempeño del algortimo de clasificación

Capítulo 9

Comparación de los métodos de clasificación

En este capítulo se pretende comparar los métodos de clasificación empleados mostrando el desempeño de cada uno a la hora de clasificar a los consumidores según la estación del año y el número de clústeres. Los resultados se centrarán en las estaciones de primavera y verano aunque también se mostrarán resultados de otras estaciones.

En la primera parte de este capítulo se presentará la clasificación de consumidores para la estación de primavera utilizando únicamente la información del primer mes de primavera. Los resultados se mostrarán en forma de tabla donde se represente la precisión en la clasificación del algoritmo. Esta primera parte del capítulo simula la situación de la llegada de un nuevo consumidor al que se le ofrece una tarifa fija durante el primer mes. Con la información de consumo del primer mes se podrá caracterizar al consumidor y clasificar en una de las clases previamente definidas a la que corresponde una tarifa particular.

La segunda parte de este capítulo muestra los resultados a la hora de clasificar a consumidores en una de las clases de verano utilizando la información del último mes de primavera, es decir, utilizando la información del último mes previo al periodo que se pretende clasificar. Puesto que la información que se utiliza y el período a clasificar corresponden a diferente estaciones, en esta clasificación realmente se hacen predicciones de las futuras clases de los consumidores.

Por último se mostrará la precisión en las predicciones para las distintas estaciones del año utilizando como única información el consumo del último mes previo a la estación que se pretende clasificar. Mientras que en la primera y segunda parte de este capítulo se utilizará una base de datos de 496 consumidores, para la última parte se reducirá la base de datos anterior a los 358 consumidores de Austin.

9.1. Clasificación Primavera 2015

Como se ha comentado anteriormente en esta sección se clasificarán a clientes utilizando únicamente el primer mes del consumo para la estación de primavera. Debido al pequeño tamaño de la base de datos (496 consumidores) el desempeño de la red neuronal se ve fuertemente afectada por los elementos que se emplean en el training y los elementos que se emplean en el testing. Por lo tanto, se ha entrenado el modelo de redes 20 veces diferentes para presentar los valores medios en la precisión del training set y el test set así como sus desviaciones típicas para ambos sets. En la Tabla 11 se muestra el desempeño en las labores de clasificación para el algoritmo K-means utilizando las series temporales (perfiles diarios representativos) y el desempeño del modelo de CNNs que utiliza como input la combinación de imágenes MTF+GAF.

Tabla 11. Resultados de la clasificación para Primavera 2015 usando el primer mes de primavera como entrada

Kmeans	CNNs			
Precisión TS	Precisión Test		Precisión Train	
0.7056	μ	s	μ	s
	0.6464	0.0608	0.6523	0.0595

En este caso la Tabla 11 muestra que el algoritmo K-means obtiene mejores resultados. Con el objetivo de dar una mayor visión general de los resultados en la Tabla 12 se presentan los resultados de la clasificación para la Primavera 2015 en función del número de clases. Se han incluido algunas variaciones para extraer otras conclusiones, por ejemplo, se ha llevado a cabo una clasificación utilizando el algoritmo K-means pero utilizando como input imágenes. En este caso las imágenes empleadas han sido la combinación GASF+GADF y los MTFs por separado para ver la influencia de cada uno de estos en la clasificación. Cuando aparece raw time series hace referencia a la serie temporal tal y como es sin ninguna transformación.

Tabla 12. Resultados de la clasificación para Primavera 2015 en función del número de clase y el tipo de input utilizado para K-means y CNNs

Clasificación Primavera 2015							
Número de Clases	K-means Precisión			Convolutional Neural Networks			
				Precisión Test		Precisión Train	
	Raw TS	GAFs	GAFs+MTF	μ	s	μ	s
6	**0.7096**	0.6108	0.6189	**0.7171**	0.0451	0.7330	0.0525
7	**0.7056**	0.5887	0.6108	**0.6464**	0.0608	0.6523	0.0595
8	**0.6995**	0.5987	0.6230	**0.6969**	0.0607	0.6973	0.0726
9	**0.6673**	0.5947	0.5987	**0.6363**	0.0412	0.6398	0.0241

En líneas generales, el algoritmo K-means ofrece mejores resultados en cuanto a precisión en la clasificación. Esto tiene bastante que ver con el hecho de que los consumidores han sido etiquetados empleando también un algoritmo K-means. El hecho de que la precisión no sea del $100\,\%$ en la clasificación con K-means se debe a que las clases se definieron utilizando tres meses mientras que se está clasificando únicamente con el primer mes de primavera. De la Tabla 12 se puede concluir que el modelo de CNNs tiene un peor desempeño conforme aumentan el número de clases, esto parece coherente puesto que a mayor número de clases más difícil es para el modelo diferenciar entre clases que se parecen mucho. También parece que las MTFs tienen una menor contribución que los GAFs a la hora de clasificar a los consumidores utilizando K-means puesto que las diferencias entre usar la combinación de tres imágenes o solo los GAFs es muy pequeña. Esto es debido a que el algoritmo K-means comprende mucho mejor a los GAFs que a los MTFs. Los GAFs representan de forma precisa a la serie temporal en un sistema de coordenadas polares mientras que los MTFs proporcionan información relacionada con la probabilidad de transición en la serie temporal que el algoritmo K-means no comprende.

9.2. Predicción Verano 2015

En la predicción se empleara la información relativa al consumo eléctrico durante el mes previo a verano. Es decir se emplearán los días laborables que hay desde el 21 de mayo al 21 de junio para construir el perfil diario representativo que se utilizará como input para clasificar a los consumidores en una de las clases definidas para el periodo de verano. El hecho de usar el último mes únicamente en vez del periodo completo de primavera se debe a que se ha demostrado que el último mes previo al periodo que se pretende clasificar es el tiene mayor poder predictivo. Esto se puede deber al hecho de que es el periodo más parecido puesto que es el más cercano en el tiempo. En la Figura 33 se comparan los perfiles diarios representativos para el usuario ID de el último mes de Primavera 2015 y del periodo completo de Verano 2015.

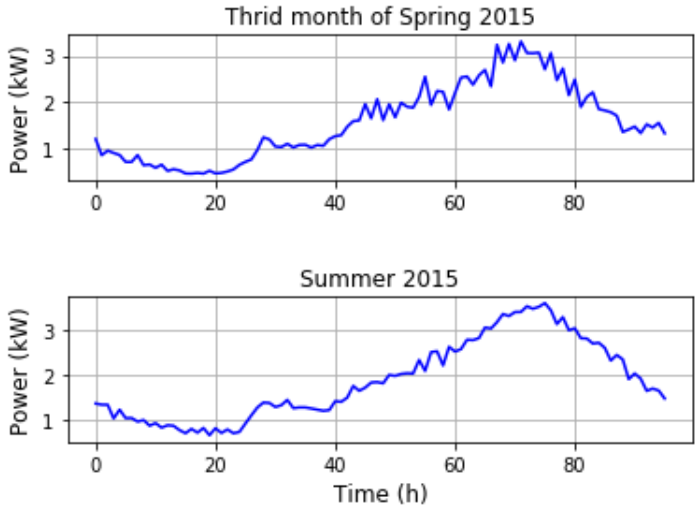


Figura 33. Comparación de los perfiles diarios representativo del último mes de Primavera 2015 y de los tres meses de Verano 2015

Como se puede observar los perfiles son muy parecidos entre sí. De igual forma que se ha hecho para Primavera 2015, la Tabla 13 compara los dos algoritmos empleados, en este caso para la predicción.

Tabla 13. Resultados de la predicción para Verano 2015 empleando el último mes de primavera como input

Kmeans	CNNs			
Precisión	Precisión Test		Precisión Train	
0.5968	μ	s	μ	s
	0.7023	0.0371	0.7330	0.0345

En este caso el modelo de CNNs presenta un mejor desempeño que el algoritmo K-means alcanzando una mejora de alrededor del $18\,\%$ con respecto a la precisión del K-means. De los resultados obtenidos en la Tabla 13 se puede concluir que el modelo de CNNs se encuentra ligeramente overfitted debido al reducido número de clientes para entrenar. Adicionalmente a los resultados mostrados en la Tabla 13, la Tabla 14 muestra la precisión en la tareas de predicción del Verano 2015 según el número de clases que se utilizan para caracterizar a los consumidores en verano.

Tabla 14. Precisión en las predicciones para el Verano 2015 según el número de clases y los inputs utilizados

Predicciones Verano 2015							
Número de Clases	K-means Precisión			Convolutional Neural Networks			
				Precisión Test		Precisión Train	
	Raw TS	GAFs	GAFs+MTF	μ	s	μ	s
6	**0.6270**	0.5423	0.5947	**0.6464**	0.0439	0.6851	0.0387
7	**0.5967**	0.5242	0.5282	**0.7023**	0.0371	0.7330	0.0345
8	**0.6048**	0.4879	0.5202	**0.6060**	0.0755	0.6574	0.0240
9	**0.5383**	0.4697	0.4959	**0.5151**	0.0521	0.5667	0.0241

Tras los resultados de la Tabla 14 se puede concluir que el modelo de CNNs funciona mejor a la hora de hacer predicciones para verano. También se concluye a mayor número de clases, menos preciso es el algoritmo de predicción excepto para 7 clases.

9.3. Predicción según la estación del año

Por último se ha comparado la precisión en las predicciones según la estación del año que se pretenda predecir. La Tabla 15 muestra la precisión en las predicciones según la estación del año empleando como input el perfil diario representativo del mes previo a la estación que se pretende predecir. Para este estudio se ha reducido la base de datos inicial utilizando únicamente

a aquellos consumidores de Austin. Esta base de datos reducida ofrece unos clústeres más compactos ofreciendo mejores resultados para el resto de estaciones.

Tabla 15. Resultados de la predicción en función de la estación a predecir usando únicamente consumidores de Austin

Predicciones año 2015					
Estación	K-means	Convolutional Neural Networks			
	Precisión	Precisión Test		Precisión Train	
	Raw TS	μ	s	μ	s
Verano	**0.6368**	**0.6080**	0.4511	0.6830	0.0345
Otoño	**0.5890**	**0.5668**	0.0371	0.7319	0.0586
Invierno	**0.5398**	**0.5205**	0.0638	0.7072	0.0423
Primavera	**0.3800**	**0.3769**	0.057	0.6761	0.0661

De los resultados presentados en la Tabla 15 se puede concluir que la predicción de verano es más fácil de generalizar y funciona bien para cualquier dataset empleado a diferencia del resto de estaciones. De la misma manera que para verano, para las predicciones de primavera se ha empleado el último mes de invierno. Un buen o mal resultado en la predicción depende en gran medida en cómo de parecidas son las series temporales de ambos periodos. Como se puede observar en la Figura 34 el último mes de Invierno 2015 es muy diferente a Primavera 2015 mientras que en la Figura 33 se puede observar que los perfiles de primavera y verano son muy parecidos.

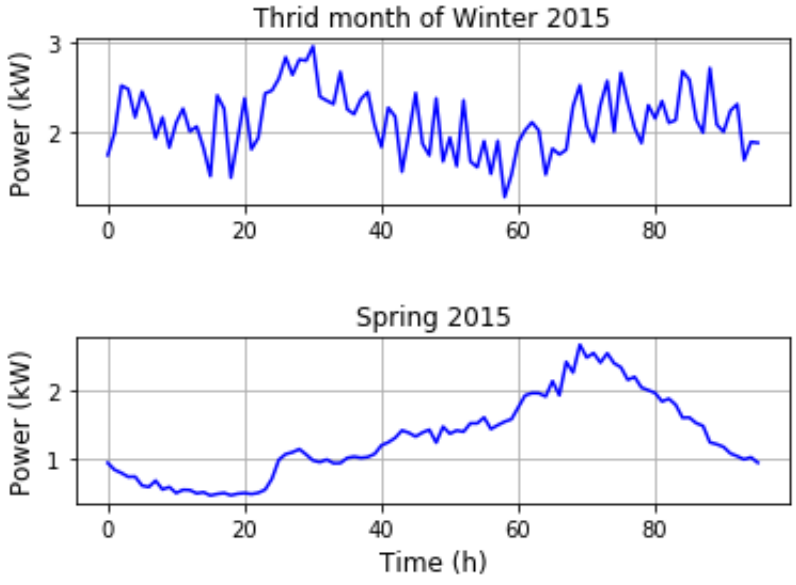


Figura 34. Comparación de los perfiles diarios representativo del último mes de Invierno 2015 y de los tres meses de Primavera 2015

Con esto se pretende justificar porque la precisión en las predicciones de verano son de alrededor del 70 % mientras que para primavera son del 38 %.

Capítulo 10

Aplicación de los resultados a la reducción de incertidumbre en la demanda eléctrica

En este capitulo se mostrará como se aplican las predicciones en las clases a la reducción de incertidumbre en las curvas de demanda individuales y totales. Se mostrarán las reconstrucciones de los perfiles individuales y se compararán con los perfiles reales. También se explicarán los procedimientos definidos en el código *UncertainReductionV3.py* que ofrece medidas para comparar cómo de buenas son las predicciones respecto a perfiles reales.

10.1. Reconstrucción de las predicciones Verano 2015

Para medir la reducción de la incertidumbre en la demanda eléctrica se han calculado una serie de medidas que cuantifican las diferencias entre la potencia real consumida por los usuarios y las predicciones para el periodo de Verano 2015. El problema es que cuando se predicen las clases de verano se produce una pérdida de información que es clave para reconstruir las curvas de demanda. El problema se encuentra en que los datos han sido normalizados utilizando la media y la desviación típica del perfil diario representativo. Por lo tanto cuando hacemos una predicción el perfil que obtenemos es el perfil representativo del clúster normalizado pero al no disponer de los valores de la media y la desviación típica para Verano 2015 no podemos de-normalizar las curvas y obtener los valores de potencia real. Es decir se puede predecir en que clúster estará un cliente y ofrecer una tarifa que se ajuste a ese clúster pero no se podrá reproducir fielmente su curva de demanda. Se tiene la opción de usar los valores de la media y la desviación típica del perfil representativo del último mes de la estación de primavera el cual ha sido usado como input para llevar a cabo la predicción. En el caso de mantener estos valores de media y desviación típica, la Figura 35 compara los valores reales de los perfiles diarios representativos con las predicciones para 4 consumidores diferentes. En el caso el que el algoritmo K-means y

el modelo CNNs coinciden a la hora de atribuir un consumidor a una clase, la predicción de la curva será la misma y por lo tanto aparecerán superpuestas.

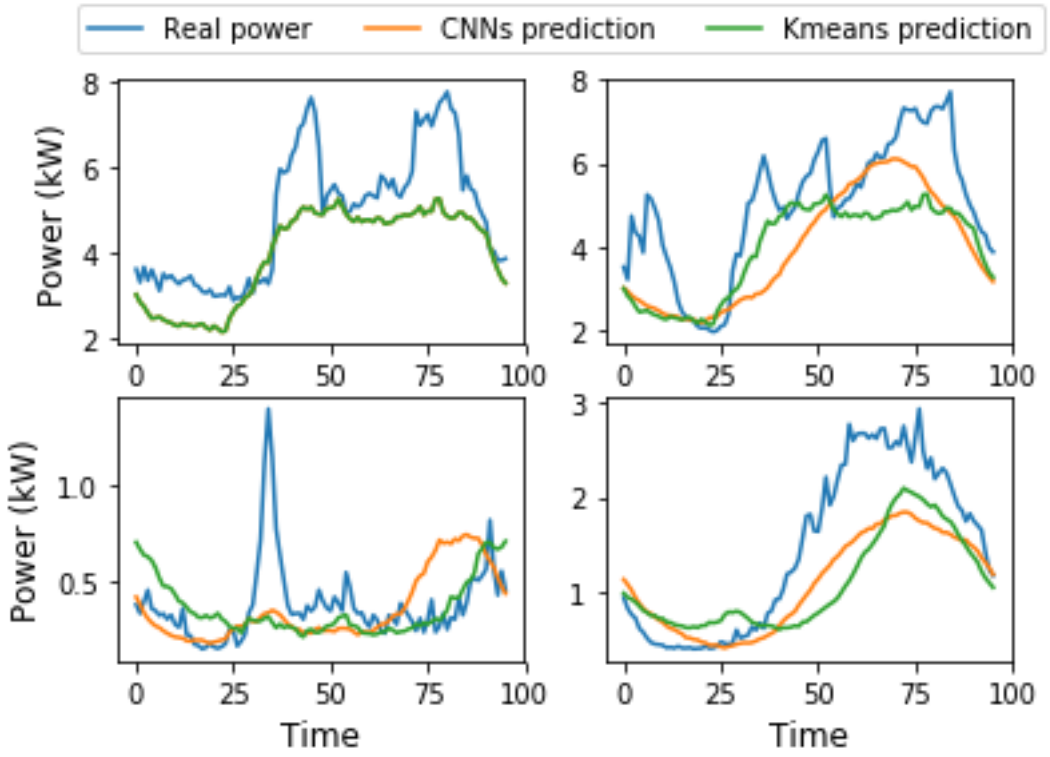


Figura 35. Comparativa de los perfiles diarios representativos para Verano 2015 y las predicciones de los algortimos K-means y CNNs usando la media y desviación típica de Primavera 2015

Si calculamos la diferencia entre los valores reales de consumo y las predicciones a largo de la dimensión temporal, es decir, el consumo de todos los clientes para la hora 0, para la hora 1, etc.., se obtienen unas diferencias del $22,33\,\%$ para las predicciones de K-means y un $23,18\,\%$ para las predicciones del modelo CNNs. Por otro lado, si se tuvieran los valores reales de la media y la desviación típica para Verano 2015, las predicciones serían mucho más precisas. La Figura 36 compara los perfiles diarios representativos reales con las predicciones para 4 consumidores usando los valores reales de las medias y desviaciones típicas para Verano 2015.

En este caso las diferencias se han reducido significativamente alcanzando una diferencia del $3{,}65\,\%$ para el algoritmo K-means y una diferencia del $0{,}65\,\%$ para las predicciones del modelo de CNNs.

Es interesante medir las diferencias con respecto a los valores reales a largo del eje temporal puesto que cuando una comercializadora acude al mercado mayorista debe estimar que demanda necesita el conglomerado de todos sus clientes para cada hora. Por lo tanto no es necesario hacer una predicción precisa del consumo instantánea individual de cada cliente. Se ha de tener en cuenta que los grandes consumidores de potencia tendrán un mayor impacto por lo

que una incorrecta clasificación de estos consumidores reportará un mayor perjuicio a las comercializadoras.

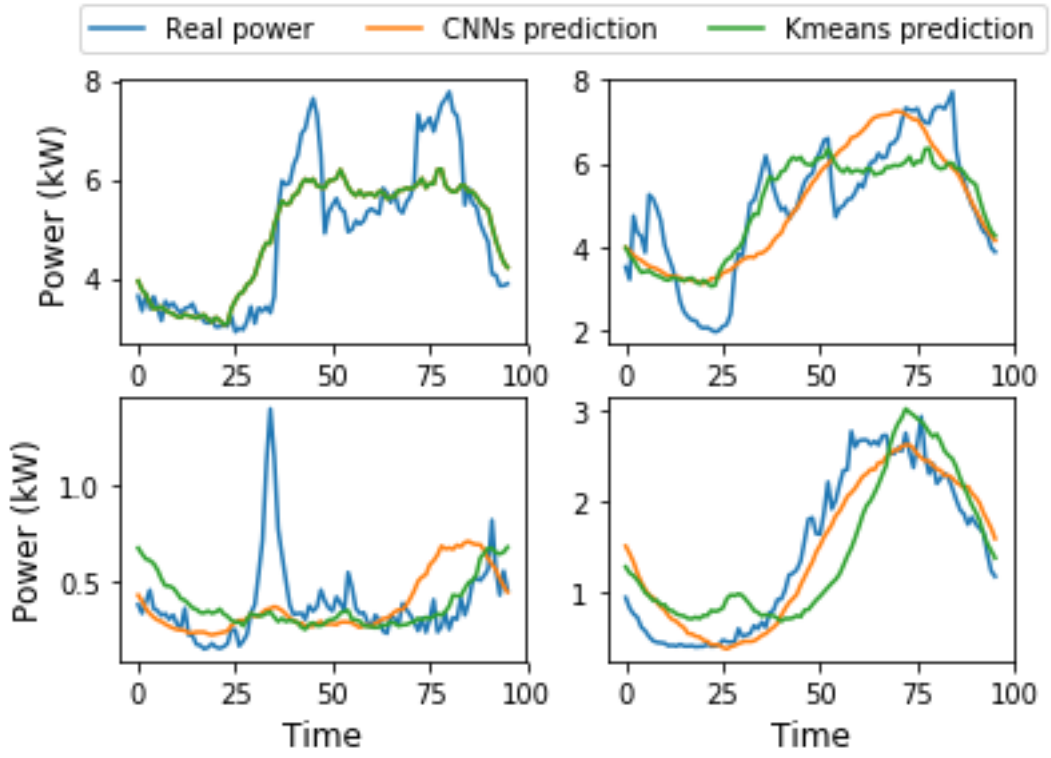


Figura 36. Comparativa de los perfiles diarios representativos para Verano 2015 y las predicciones de los algoritmos K-means y CNNs usando la media y desviación típica de Verano 2015

10.2. Implantación en Python

A continuación se explica cómo se han implementado todas las operaciones descritas anteriormente en Python. La figura 37 recoge los procedimientos definidos en el archivo *UncertainReductionV3.py*. El primer paso es como en los demás archivos anteriores la activación de variables. Tras la activación de variables se cargan los datos de los periodos seleccionados, se reducen los datos a los elementos comunes a ambos periodos y se normalizan los datos. Tras estos procedimientos es clave comprobar si la variable FAKE se encuentra activa o no. La variable FAKE activada indica que se utilizarán los valores reales de la media y la desviación típica de los consumidores para Verano 2015. En caso no estar activada se utilizará la media y desviación del último mes la Primavera 2015. Tras esto se definen el siguiente conjunto de variables: *real_energy_day, CNNs_energy_day, Kmeans_energy_day, real_power_15min, CNNs_power_15min, Kmeans_power_15min, Kmeans_power_15min* y *K_means_percentage_instant_power_np*. Las variables *XX_energy_day* recogen el consumo total de energía para un día según el algoritmo de predicción. Las variables *XX_power_15min* recogen los valores de potencia cada 15-minutos y por lo tanto constituyen la reconstrucción de los perfiles diarios representativos. Por último las variables *XX_percentage_instant_power_np*

recogen la diferencia en tanto por ciento de las predicciones de los valores de potencia con respecto a los valores reales cada 15-minutos.

El bloque de precisión en la clasificación hace referencia al cálculo porcentual de la precisión de cada algoritmo a la hora de clasificar a los clientes en el periodo de verano. Puesto que se están utilizando como prueba los clientes que han sido utilizados para definir las clases de Verano podemos saber cuándo los algoritmos de clasificación aciertan o no en sus predicciones. A continuación se presenta un bucle for en el que se calculan diferentes valores para cada cliente de forma individual. Por lo tanto no se sale del bucle hasta que se han analizado todos los clientes. Entre la operaciones destacables se encuentra la reconstrucción del perfil diario representativo o la diferencia entre la potencia instantánea real y las predicciones. El Código 9 muestra como se reconstruyen los perfiles diarios representativos a partir de la predicción del modelo de CNNs.

```
power_2 = average_1D_np[i]+(centers_season1[labels_CNNs[i]]*std_dev_1D_np
    [i])
```

Codigo 9. Reconstrucción perfil diario representativo a partir de las predicciones del modelo de CNNs

Finalmente se presentan valores medios como la diferencia media de potencia a lo largo de las 96 dimensiones (cada uno de los datos recogidos cada 15 minutos) o la diferencia media en la potencia instantánea de las predicciones respecto a los valores reales.

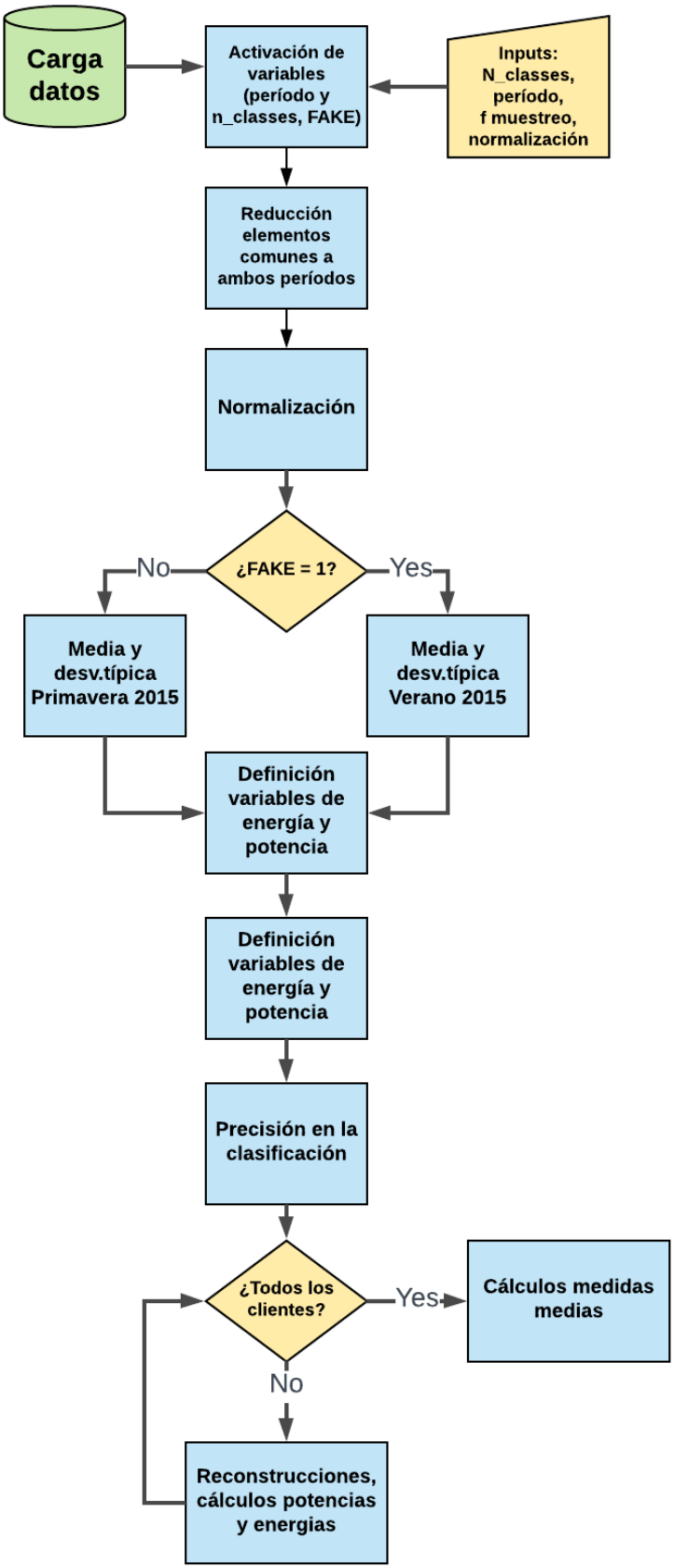


Figura 37. Estructura del archivo *UncertainReductionV3.py*

Capítulo 11

Conclusiones y trabajo futuro

En este trabajo se ha mostrado la utilidad de los más novedosos algoritmos de machine learning en aplicaciones reales. La alta precisión alcanzada por modelos de CNNs en tareas de clasificación de series temporales presentadas en otras trabajos fue la inspiración para aplicar estos conceptos en la predicción de demanda eléctrica. La definición de los GAFs y los MTFs con la combinación de las Redes Neuronales Convolucionales, aporta nuevas posibilidades en el análisis de series temporales de una manera totalmente diferente a como se ha hecho tradicionalmente.

Entre otras cosas este trabajo explica como extraer información significativa a partir de las formas de los perfiles de demanda obtenidos a partir de grandes cantidades de datos. Además se explica como transformar las series temporales en imágenes que pueden ser clasificadas por modelos de redes neuronales. Este trabajo también desarrolla un procedimiento generalizado para obtener los perfiles de consumidores típicos presentes en cualquier base de datos y usar esta generalización para predecir futuras demandas. Estas predicciones ayudarán a reducir la incertidumbre en la demanda eléctrica los que ayudará reducir los costes a las comercializadoras.

Pese a todo esto los resultados obtenidos no son del todo satisfactorios puesto que los resultados obtenidos no mejoran el resultado de los algoritmos empleados actualmente para la predicción de demanda. Los siguientes pasos a este trabajo, debe de ser probar estos conceptos sobre una base de datos real más grande proporcionada por una comercializadora. Datasets de mayor tamaño darán mejores resultados en términos de precisión en la clasificación y en la generalización de consumidores.

Bibliografía

[1] **Zardoya Javier, Chocarro Aritz, (2016)**, *"Análisis viabilidad comercialización eléctrica y posibilidades", versión 7, Ayuntamiento de Pamplona,*

[2] **Figueiredo V, Rodrigues F, Vale Z, Gouveia JB (2005)**, *"An electric energy consumer characterization framework based on data mining techniques", IEEE in Trans.Power Syst., vol. 20, no.2, 2005.*

[3] **G. Chicco, R. Napoli, P. Postulache, M. Scutariu, C. Toader (Feb 2003)**, *"Customer characterization options for improving the tariff offer", IEEE in Trans. Power Syst., vol. 18, no. 1, pp. 381387, 2003.*

[4] **Mill Steven (2016)**, *Electric load forecasting: advantages and challenges*.
Última consulta: 23/05/2018
http://engineering.electrical-equipment.org

[5] **Wang Zhiguang, Oates Tim (2015)**, *"Encoding Time Series as Images for Visual Inspection and Classification Using Tiled Convolutional Neural Networks", Association for the Advancement of Artificial Intelligence, 2015 AAAI Workshop*

[6] **PecanStreet (2018)**, *Dataport, Base de datos*.
Última consulta: 25/02/2018
http://dataport.cloud

[7] **Johann Faouzi (mayo 2018)**, *pyts: a Python package for time series transformation and classification*.
Última consulta: 23/05/2018
https://doi.org/10.5281/zenodo.1244152

[8] **Scikit-Learn (mayo 2018)**, *Scikit-Learn: a Python package for Machine Learning*.
Última consulta: 23/05/2018
https://scikit-learn.org

[9] **AustinEnergy (mayo 2018)**, *Time-Of-Use Pilot program*.
Última consulta: 23/04/2018
http://austinenergy.com

[10] **Iberdrola S.A. (mayo 2018)**, *Planes de luz para tu hogar*.
Última consulta: 12/05/2018
https://www.iberdrola.es/luz

[11] **Keras**, *Chollet, François and others*.
Última consulta: 26/06/2018
https://keras.io

[12] **Wikipedia**, *Redes neuronales convolucionales*.
Última consulta: 26/06/2018
https://es.wikipedia.org/wiki/Redes_neuronales_convolucionales

[13] **Texas Advanced Computing Center**, *The University of Texas at Austin*.
Última consulta: 26/06/2018
https://www.tacc.utexas.edu/

[14] **What is a tiled convolutional neural network?**, *Charles Moyes*.
https://www.quora.com/What-is-a-tiled-convolutional-neural-network

[15] **A Beginner's Guide To Understanding Convolutional Neural Networks** , *Adit Deshpande*.
https://adeshpande3.github.io/A-Beginner%27s-Guide-To-Understanding-Convolutional-Neural-Networks-Part-2/

Paper

Electric Energy Consumer Characterization, Classification and Demand Forecasting using Convolutional Neural Networks

Ignacio Aguirre[1], Stepan V Ulyanin[2], Jose Vazquez-Canteli[2], Zoltan Nagy[2]
[1]Universidad Pontificia de Comillas ICAI, Madrid, Spain
[2]University of Texas at Austin, TX USA

Abstract

This paper presents a procedure to support the retail and distribution companies on the extraction of knowledge from electricity consumption data. Our final objective is to forecast individual load profiles of consumers for a particular season of the year using the historical information gathered during the months of the previous season. The algorithm classifies consumers in one of the 7 clusters, with 70% accuracy in the best case, which gives retail companies an excellent point to start in tariff customization. We encode the power use (time series) in GAFs and MTFs to represent the images that are processed by the CNNs to perform classification tasks.

Introduction

Electric utilities are currently experiencing an age of changes that are transforming the traditional business of the power industry. One of these changes is the extensive amount of data that they have but do not know how to manage in a useful manner to reduce consumer costs and improve the overall system efficiency. This paper presents a procedure to support the retail and distribution companies on the extraction of knowledge from electricity consumption data.

One of the significant consequences of the electricity markets liberalization is the freedom that all customers will have in choosing their particular electricity supplier. This open market creates an environment where retail companies compete for the electricity supply of end users. The knowledge of how and when consumers consume electricity is the key to be competitive in the electric retail industry Figueiredo V (2005). From historical data of different consumer demand, it is possible to derive this kind of knowledge. It is essential to classify every consumer in one of the classes represented by its load profiles. The number of classes in which we separate the data is an input that we must introduce when assigning the labels to consumers, and it must be based on the level of compactness needed and the results obtained in the forecasts.

Once we characterize new consumers, there is a better chance to offer tariffs that will benefit them in the long term. Furthermore, this characterization provides several advantages:

1. Accurate load forecasting for better system expansion planning to achieve good service quality Figueiredo V. and et al. (2003)

2. It might help to reduce the costs associated with an excess of power capacity in the design of electric equipment required for distribution such as lines and transformers Mill (2016).

3. It might help retail companies to determine which consumers will be benefited by Demand Respond Programs such as Real Time Pricing tariffs M. H. Albadi (2007) Valero S (2004).

This paper bases on the application of clustering techniques for the determination and characterization of a set of load profiles, representing the different consumption patterns of a sample of consumers. The final objective is to forecast individual load patterns for a season of the year using the information gathered during the previous season. To achieve this, we use Convolutional Neural Networks and K-means algorithm to link for example spring inputs of consumers with their summer clustering classes. Wang Zhiguang (2015) shows the techniques to encode time series as images that we can directly apply to our data. In his conference paper, he proposes a framework to encode time series as different types of images, namely, Gramian Angular Fields (GAF) and Markov Transition Fields (MTF). The definition of these fields allows the use of Convolutional Neural Networks as the algorithm to classify new consumers to the specified classes. The prediction of the behavior of new consumers one season ahead will help retail companies in personalizing electric tariffs and in planning the amount of electricity they need to buy. The reduction of the unpredictability associated with electrical consumption will lead electricity providers to reduce costs. These reductions in costs must affect consumers in the way of savings in their electricity bills.

The paper is organized as follows: in the first part, the characterization section explains the methodology to

prepare and transform the data. The forecasting section explains the conversion process of time series into MTFs and GAFs. Then, we present a case study with real data to analyze the performance of our method. Finally, the results and conclusions of the work will close the document.

Methodology

For this work, we developed a procedure to reduce the uncertainty in residential electricity demand to help electricity providers in offering personalized electric tariffs to new clients. Figure 1 depicts this process using a block diagram to show the procedure flow. The first step is to carry out a characterization study of the different classes of electric behaviors among the consumers. Once we define the classes, it is possible to customize electric tariffs that will adapt to each of the classes defined. A good approach with new consumers would be to initially offer them a fixed price tariff until electricity providers gather enough information about their consumption patterns. After collecting information for the first month, we can classify consumers in one of the classes defined by the clustering. At this point we will compare the performance of the CNNs model and the K-means algorithm in the classification task. Having consumers classified, it is possible to offer them the tariff that will better fit their needs. Once we collect one season of data, it is possible to forecast in which cluster class the client will be the next season. Classifying consumers for the next season is a more difficult task because the algorithms now must classify consumers using information from a different period. For the forecasting, instead of using three months of data averaged to construct a representative daily profile, the data will be averaged per month with the purpose of losing the least information possible. But after doing some simulations, it can be stated that the data that have more predicting power is the most recent one to the season that wants to be predicted. An explanation of this fact is that recent data will be more similar in shape to the profiles that we pretend predict. For this reason, we decided to use just the third month of the previous season to predict consumer classes of the next season. Therefore, electric tariffs will be reviewed once every three months. Our case study will focus on the seasons of spring and summer which give the best results. More results concerning the rest of the year will also be provided at the simulation and results section.

The characterization section summarizes the steps to prepare the data to obtain the different classes of customers. It explains which parameters are the key to achieve an accurate generalization of the typical consumers presented in our data. The forecasting section summarizes how to encode time

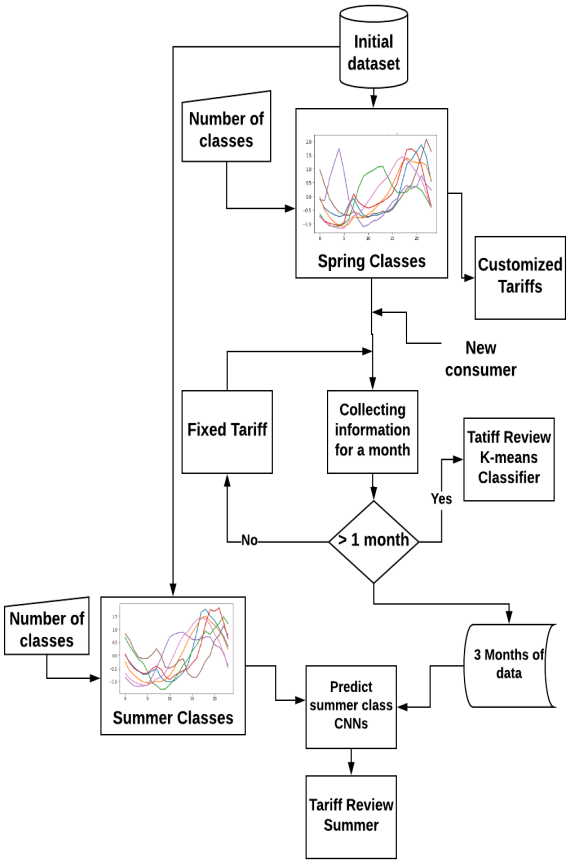


Figure 1: Classification procedure

series as images that the CNN model uses as inputs. This section explains why Markov Transition Fields and Gramian Angular Fields can be combined as 3 channels to form an image and why this combination is effective in predicting consumer patterns.

Characterization section

The characterization section is organized in the following steps:

Data selection and cleaning: the first step is the selection of the data with significance to the process. It is necessary to conduct different studies depending on the voltage level of consumers. In this case, this work focuses on the residential level. The power use is measured as the total amount of power consumed by a home in an instant, and therefore it includes the sum of all the different electrical devices and appliances of a house. In the cleaning phase, we look for inconsistencies in the data and we remove consumers with null elements from the initial dataset.

Data reduction: the season of the year and the type of day (working days or weekends) affect electricity consumption patterns. We use these characteristics to separate the data into smaller datasets. Two

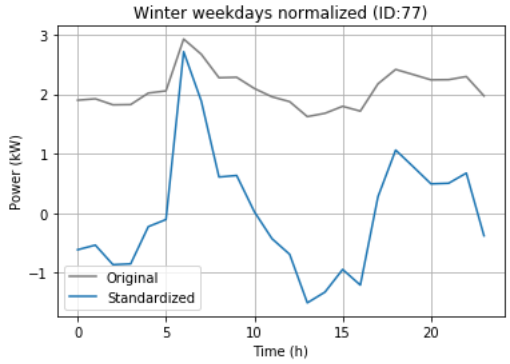


Figure 2: Comparison of the original and the normalized winter weekdays representative load diagram for ID77

datasets represent each season of the year, one for working days and another for weekends. The consumer data of each season of the year is reduced to one daily load profile which is known as the representative load profile. Representative load profiles are obtained by averaging the load profiles of the whole season to one day. The data is averaged along its dimensions which are the 24 hours of a day. Each consumer is then described by one single representative load profile in each dataset, for the different loading conditions. For example, Consumer X will have a representative daily load diagram for winter weekdays and a different one for summer weekdays.

Data normalization: we construct the load profiles using the field-measurement values, so they need to be brought together to a similar scale for their comparison. The representative load diagram of each consumer was normalized using Z-normalization. This procedure ensures that all elements of the input vector are transformed into the output vector whose mean is approximately 0 while the standard deviation is in a range close to 1. This kind of normalization allows maintaining the shape of the curve and comparing the consumption patterns. Figure 2 illustrates the differences between the raw and Z-normalized time series. The normalization formula is shown below:

$$x_i^{'} = \frac{x_i - \mu}{\sigma} \tag{1}$$

Load profiling: here the goal is the partition of the data sample into a set of classes defined according to the load shape of the representative load diagrams of each consumer. The data-mining model for consumer characterization is based on the unsupervised learning technique clustering K-means to form the clusters. The representative load diagram of the mth consumer is the vector $p^{(m)} = \{p_1^m, ..., p_h^m, ..., p_H^m\}$ where p_h^m describes the normalized values of the power consumed at the instant h and $h = 1, ..., H$ with $H = 24$ or $H = 96$ depending of the use of 1-hour interval or 15-min interval resolution. The use of 1-hour interval or 15-min will depend on the application of the classification. K-means algorithm compares the power use of all consumer at each of the 96 dimensions to minimize the distance between elements in the same cluster. The centroids represent the average of all those elements grouped in the same cluster. K-means algorithm would classify the same two time series in two different classes if they are lagged, as it considers every dimension independently (every hour, every 15 min). 15-min interval time series are more sensitive to this issue and thereby are a worse option if the objective is to find the class that shares similar metadata characteristics, such as, the size of houses or the number of members in a house. On the other hand, CNNs take into account temporal relationships between dimensions (this information is provided by GAFs and MTFs) in a similar way as an ACF does with time series. 15-min interval time series are more precise for studying the simultaneity between consumers and tariff designing. They also have a better performance on CNNs and this is why they are our final choice.

The classes obtained in the clustering represent the different consumption patterns among the sample in the study. The load profiles of the centroids represent each of these classes. A good consumer characterization will have a great impact on the success of the forecasting module. A functional characterization must ensure well-separated classes (a proper distinction between class representative load diagrams) and compacted classes (the load diagrams included in each class must be similar to their representative centroids). To evaluate the performance of the algorithm we use two measures of adequacy: a measure of cluster compactness (MIA) and another measure of cluster separation (CDI). G. Chicco (2003) presented both measures in his work . Inputs with smaller MIA and CDI values are considered to have sharper discriminating properties. Considering a set of X load diagrams separated in K classes with k=1,...,K and each class is formed by a subset $C^{(k)}$ of load diagrams, where $r^{(k)}$ is a pattern assigned to cluster k (cluster centroid), G. Chicco (2003) defined the following performance measures:

a) Mean Index Adequacy (MIA):

$$MIA = \sqrt{\frac{1}{K}\sum_{k=1}^{K} d^2(r^{(k)}, C^{(k)})} \tag{2}$$

b) Clustering Dispersion Indicator (CDI):

$$CDI = \frac{1}{d(R)}\sqrt{\frac{1}{K}\sum_{k=1}^{K} d^2(C^{(k)})} \tag{3}$$

The Mean Index Adequacy (MIA) depends on the average of the mean distances between each pattern assigned to the cluster and its center. The Clustering Dispersion Indicator (CDI) is directly proportional to the distance between the load diagrams in the same cluster and inversely proportional to the distance between the class representative load diagrams. The number of classes selected for our K-means algorithm must follow a balance between compactness and the purpose of the clustering. A reasonable number of classes will be between six and nine. Below six, the classes are too general while above nine the classes are too specific. The idea is to let electricity providers design the tariffs that will adapt to their consumers. Therefore, offering more than 6-7 different tariffs are too many as most of them only offer two currently, a fixed rate tariff and a Time-Of-Use rate tariff.

Forecasting section

The effectiveness of Gramian Angular Fields and Markov Transition Fields in classification problems become clear after the publication of "Encoding Time Series as Images for Visual Inspection and Classification Using Tiled Convolutional Neural Networks" by Wang Zhiguang (2015). Wang's work considers the problem of encoding time series as images to allow machines to "visually" recognize, classify and learn structures and patterns. Here we summarized the three representations introduced by Wang Zhiguang (2015). They are called Gramian Angular Summation/Difference Fields (GASF/GADF) and Markov Transition Fields (MTF). Table 4, shows an example of these three images for the representative profile of the last month of spring 2015. We generated these images using Faouzi (2018)

Gramian Angular Field

Gramian Angular Fields (GAF), proposed by Wang Zhiguang (2015), are a way to encode angular information about the time series into an image. The image resulting from a GAF transformation is a quasi-Gramian matrix, computed on one of the two inner-product spaces,

$$< x, y >= x \cdot y - \sqrt{1 - x^2}\sqrt{1 - y^2} \tag{4}$$

$$< x, y >= y \cdot \sqrt{1 - x^2} - x \cdot \sqrt{1 - y^2} \tag{5}$$

for Gramian Angular Summation Fields (GASF) and Gramian Angular Difference Fields (GADF) respectively. These Gramian matrices are equivalent to first normalize the time series data $X = \{x_1, x_2, ..., x_n\}$ to fall within the interval $[0, 1]$ and convert them to polar coordinates such that $\phi_n = \arccos(x_n)$. After transforming the re-scaled time series into the polar coordinate system, it is easy to exploit the angular perspective by considering the trigonometric sum/difference between each point to identify the temporal correlation within different time intervals. The Gramian Angular Summation Field (GASF) and the Gramian Angular Difference Field (GADF) are matrices whose elements are defined as follows:

$$GASF_{ij} = [cos(\phi_i + \phi_j)] \tag{6}$$

$$GADF_{ij} = [sin(\phi_i - \phi_j)] \tag{7}$$

The encoding map explained by Wang and Oates has two important properties. First it is bijective as $cos(\phi)$ is monotonic when $\phi \in [0, \pi]$. This means that given a time series, the proposed map produces one and only one result in the polar coordinate system with a unique inverse map. Therefore, a time series will have its unique GAF representation. It is important to re-scale the time series so that all values fall in the interval $[0, 1]$ as this will permit to reconstruct the time series from the GASF as follows. The main diagonal of the GASF, i.e. $\{G_{ii}\}$ allow us to precisely reconstruct the original time series by:

$$cos(\phi) = \sqrt{\frac{cos(2\phi) + 1}{2}} \tag{8}$$

According to Wang Zhiguang (2015), there are several advantages of GAF representation method: the bottom right element in the matrix contained the last information in the time series, and the old information may be placed on the upper and left parts. The timestamp of each data element in this matrix will be increased with rightward and downward axes. This attribute means that the generated graph can retain the temporal dependency, making the images without losing the property of the original data. GAFs give at least as much information as the time series itself with the elements of its diagonal. But it also gives additional information not available explicitly in the time series as the temporal correlations between elements separated a time period of $|i-j| = k$. Another import feature of this encoding map is that polar coordinates preserve absolute temporal relations. These features might have an important impact because the Neural Network might learn these feature in the training process.

Markov Transition Field

The MTFs encode dynamical transition probabilities sequentially to preserve information in the time domain. The steps to encode time series into MTFs can be summarized in three parts. First, identify the Q quantile bins of a time series and assign each x_i to the corresponding bins q ($q \in [1, Q]$). Then construct the Markov transition matrix $[W] = [QxQ]$ by counting transitions among quantile bins. $\{W_{ii}\}$ accounts for self-transition probability whereas $\{W_{ij}\}$ shows the frequency with which a point in quantile q_j follows a point in quantile q_i. Finally, the Markov Transition Field (MTF) is a $[NxN]$ matrix where N represents the number of elements of the

time series. $\{M_{ii}\}$ represents the self transition probability of the quantile q_i at time t_i whereas $\{M_{ij}\}$ represents the transition probability from the quantile at t_i to the quantile at t_j. The map functions of MTF will produce one and only one image with fixed size and fixed number of quantiles Q for each given time series . Because its mapping functions are surjective, the inverse image of the mapping functions is not fixed, which means that it is not possible to recover the raw time series from its MTF.

Wang Zhiguang (2015) stated that G_{ij} denotes the superposition/difference of the directions at t_i and t_j whereas M_{ij} is the transition probability from the quantile at t_i to the quantile at t_j. GAF encodes static information because it contains temporal relations similarly to an autoregressive model (AR) while MTF depicts information about dynamics in a similar way to a logistic regression does. From this point of view, we consider them as three orthogonal channels, like different colors in the RGB image space. Thus, we can combine GAFs and MTF images of the same size (i.e. $S_{GAFs} = S_{MTF}$) to construct a triple-channel image (GASF-GADF-MTF).

Simulation: case study on a data base of electric consumers

In this case study, we focus on the months of spring and summer to give some significant results. The database contains information from 496 residential consumers between Texas, California and Colorado. This database is provided by PecanStreet (2018).

Following the procedures presented in the *characterization section*, we reduce and normalize the data. Each consumer is then described by a normalized representative daily load curve as Figure 3 illustrates. Figure 3 depicts the minimal differences between the representative load profile for the consumer ID77 depending on the sampling size.

Table 1 shows MIA and CDI compactness measures depending on the number of classes from six to nine. In general, as the number of classes increases, the MIA decreases while the CDI increases except for nine clusters. Nine classes seem to be the best answer regarding compactness, but it will make the forecasting task more challenging as we discuss later. Eight classes are even worse than nine because the clusters are less compact and eight classes are still challenging for the forecasting task. Seven classes was the final decision over six because an additional class will help in collecting all those consumers that have very particular profiles and are difficult to classify in other classes.

Figure 4 illustrates the cluster centroids for Spring

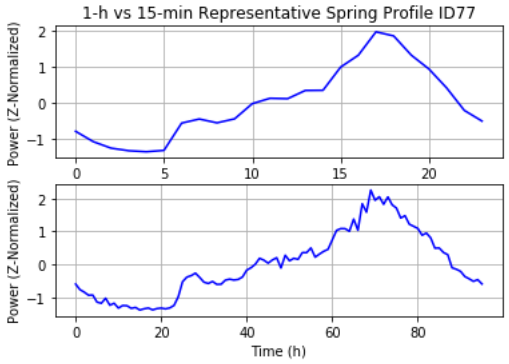


Figure 3: Comparison of the representative load profile of ID77 for spring with a resolution of 1-hour and 15-min

Table 1: MIA and CDI of the summer and winter clusters depending on the number of clusters

Consumer characterization				
	Spring		Summer	
clusters	MIA	CDI	MIA	CDI
6	0.5637	0.9236	0.5555	0.9277
7	0.5541	0.9335	0.5491	0.9471
8	0.5402	0.9475	0.5260	0.9448
9	0.5345	0.9246	0.5208	0.8776

2015. At this point, all the consumer have been labeled from 0 to 6 depending on their spring consumption patterns. With a month of data, there is enough accuracy to classify new consumers in one of the classes that Figure 4 describes.

Due to the small size of the dataset (496 consumers), the accuracy of the neural network depends heavily on which elements were used for the training and which for the testing. Therefore, we trained the neural network model 20 different times to present its average test accuracy values and its standard deviation. The performance in the classification

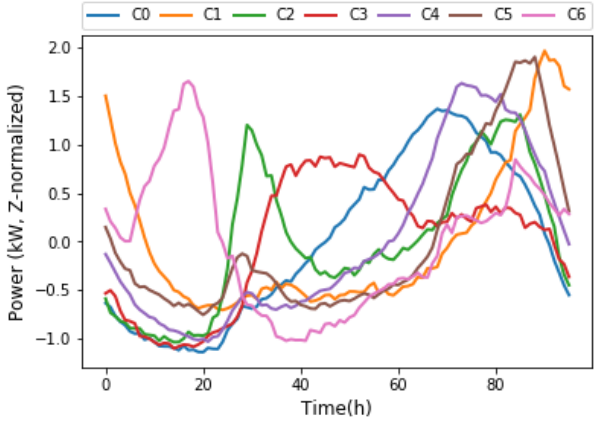


Figure 4: Cluster centroids for Spring 2015 workdays with the data collected in 15-min intervals

Table 2: Classification results for Spring15 using the first month of Spring as input

Kmeans	CNNs			
TS Accuracy	Test Accuracy		Train Accuracy	
0.7056	μ	s	μ	s
	0.6464	0.0608	0.6523	0.0595

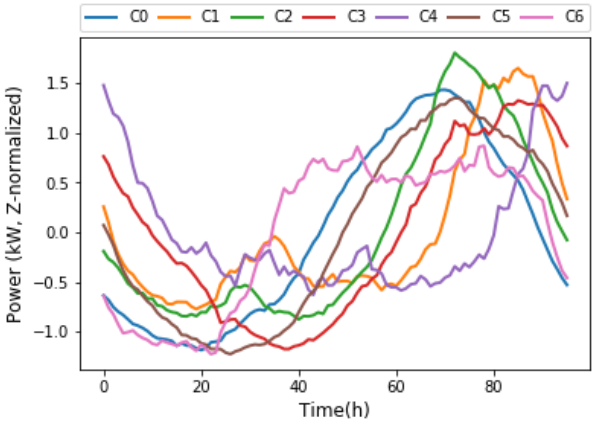


Figure 5: Cluster centroids for summer 2015 workdays with the data collected in 15-min intervals

task of the K-means algorithm using the raw time and the CNNs model using the MTFs and GAFs is compared in Table 2. In this case, Table 2 shows that the K-means classifier works better.

Figure 5 shows the different classes in summer. Cluster number 4 collects 15 elements with unusual behavior that could not be grouped in other clusters. These elements must be removed from the study.
Figure 6 shows the representative daily profiles of consumer ID77 for the last month of Spring and for the whole summer while Table 4 shows the MTF, GASF, and GADF of the representative daily profile of the last month of spring 2015.
As we did in spring, two different classification algorithms are compared in Table 3. In this case, the CNNs model performs significantly better than K-means algorithm, achieving around an 18% improvement. From the results presented in Table 3, we can stated that the CNNs model is slightly overffited due to the reduce number of consumers used in the training.

Table 3: Classification results for summer 2015 using the last month of spring as input

Kmeans	CNNs			
Accuracy	Test Accuracy		Train Accuracy	
0.5968	μ	s	μ	s
	0.7023	0.0371	0.7330	0.0345

Additionally, to measure the reduction of unpre-

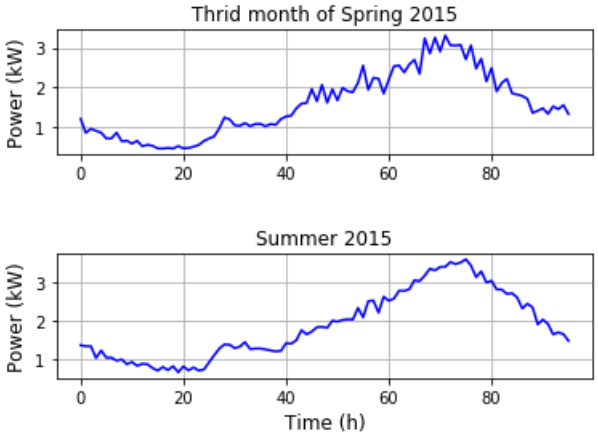


Figure 6: Comparison between the representative profile of the last month of spring and the three months of summer 2015 for consumer ID77

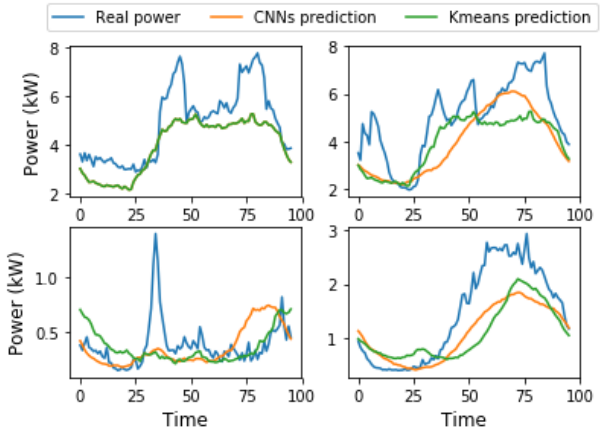


Figure 7: Comparison between the representative profile of the summer and the CNNs and K-means predictions using data only from spring for 4 consumers

dictability in the demand, we have calculated different indicators to quantify the differences in power between the real values and the forecasts for summer 2015. The problem is that when we forecast summer classes, we loss information that is key to reconstruct the power profiles. In this case, we do not have the values for the mean and the standard deviation of each representative power profile that are used to denormalize the cluster centroids. We have the option of using the mean and standard deviation extracted from the last month of spring data. Figure 7 depicts the comparison between the real representative profiles and the predictions for 4 consumers. In the case in which CNNs and K-means agree in the same label, the profiles are overlapped. If we compute the difference along the same dimension (hour 0, hour 1, ...) between the real values and the predictions, we obtain a difference of the 22.33% for the K-means prediction and a difference of the 23.18% for the CNNs prediction.
On the other hand, if we had the real values of the

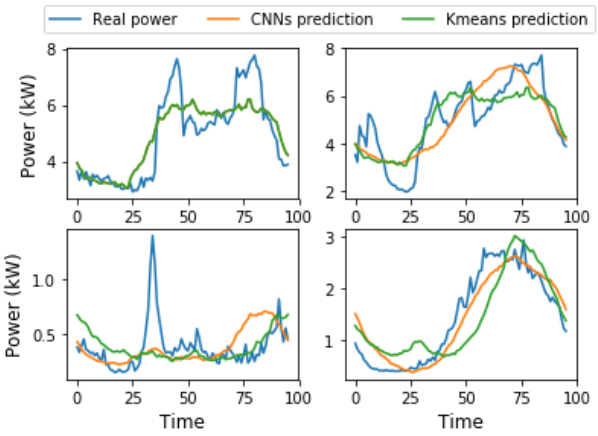


Figure 8: Comparison between the representative profile of the summer and the CNNs and K-means predictions using the real values of the average and standard deviation for 4 consumers

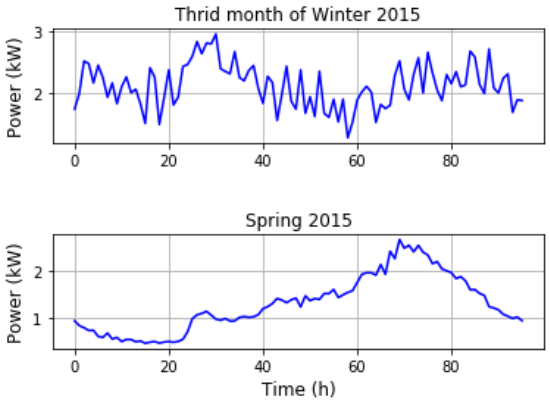


Figure 9: Comparison between the representative profile of the last month of winter and the Spring 2015 for consumer ID77

average and the standard deviation the results would be much more accurate. Figure 8 depicts the comparison between the real representative profiles and the predictions for 4 consumers using the real values of the average and standard deviation for summer. In this case the differences are significantly reduced reaching a 3.65% for the K-means prediction and a difference of 0.65% for the CNNs prediction.

Discussion and result analysis

Regarding to the classification task, Table 5 shows the results of the spring classification depending on the numbers of classes with the objective of generalizing the results of the case study. Some variations were introduced to extract some relevant conclusions. For example, these variations include a K-means classification using the 3 channel images (MTF + GAFs) and using only the GAFs (2 channels). Raw TS means using the traditional time serie without transformations. In general, the K-means algorithm has a better performance than the CNNs. The reason of this is that the spring data have been labeled using a K-means algorithm and therefore the K-means algorithm for classification will give the best results possible. The K-means do not achieve an accuracy near to 100% because the labels have been created using the information of the three months of spring whereas the classification algorithm uses the data of the first month of spring. From Table 5 can be stated that CNNs have worse performance when the number of classes increases. This seems fair because the more classes are, the more difficult is for the CNNs to differentiate between very similar classes. Additionally, it appears that the MTFs have a smaller contribution to the K-means accuracy at all. The differences between using the three channels or using just the GAFs are very small. This is because the K-means algorithm better understands the GAFs than the MTFs. GAFs accurately represent the time serie in a polar coordinate system while the MTFs provide information regarding transition probabilities in a time serie that the K-means algorithm does not know how to interpret.

With respect to the forecasting task, Table 6 shows the results of the summer forecasting depending on the numbers of classes with the objective of obtaining general results of the case study. It seems that the results obtained by the CNNs are significantly better. The more number of classes, the less accurate the forecasting algorithm is except for the CNNs model with 7 classes. Table 7 shows the results of forecasting every season of the year using the last month of the previous season to the one that is predicted. For this study we have used a reduced dataset of our initial data using only electricity consumers from Austin. This dataset gives us more compact clusters obtaining better results for the rest of the seasons. From the results, it seems that summer predictions are easier to generalize and work better for any dataset than the rest of the seasons. As we did for summer, the last month of winter tries to predict the classes of spring. It seems that a good or a bad performance has a lot to do with how similar are the time series of both periods. As it can be seen in Figure 9, the last month of Winter 2015 is very different than Spring 2015 while in Figure 6 the profiles are quite similar. This explains why the accuracy of forecasting summer is about a 70% while for spring is about a 38%.

Regarding to Figure 7 and Figure 8, we can state that the most accurate way to measure the performance of our forecasts is using the 15-min power use differences as it gives us the differences of power in real-time. A good point to take into account is that consumers with higher power levels will have a bigger impact in our total energy results and therefore a bad classification of these elements will lead to worse results.

Conclusion

In this work, we showed how state-of-the-art machine learning algorithms can be used in real-world applications. The high accuracy of CNNs models in classifying time series presented in other works was the inspiration to try these concepts in electrical demand forecasting. The definition of the GAFs and the MTF with the combination of Convolutional Neural Networks, gives new possibilities in time series analysis in a different way as it has been done traditionally. This work has covered how to extract significant information of load profiles shapes from large quantities of data and how to encode this data in images that can be used in classification tasks. This work also covers a procedure to use this data to generalize the typical electric consumers and to use this generalization to predict future demands. This forecasts will help to reduce the uncertainty in electric demand which will reduce the costs of electricity providers. The most significant point of this work is the high accuracy obtained in the prediction of future consumer behaviours. On the other hand, this works falls in obtaining an accurate clustering model that will correctly represent the general behaviour of our consumers. The next steps of our work, will be to prove these concepts in a real dataset provided by an electricity supplier to see how this procedure works and how it helps in tariff customization. Bigger datasets should give better results in terms of classification accuracy and consumer generalization.

References

Faouzi, J. (2018, May). pyts: a Python package for time series transformation and classification. `https://doi.org/10.5281/zenodo.1244152`. [Online; accessed May-2018].

Figueiredo V., F. J. Duarte, F. R. Z. V. and J. G. et al. (2003). Electric energy customer characterization by clustering. Greece.

Figueiredo V, Rodrigues F, V. Z. G. J. (2005). An electric energy consumer characterization framework based on data mining techniques.

G. Chicco, R. Napoli, P. P. M. S. C. T. (Feb 2003). Customer characterization options for improving the tariff offer. In *Trans. Power Syst., vol. 18, no. 1, pp. 381387*. IEEE.

M. H. Albadi, E. F. E.-S. (2007). Demand response in electricity markets: An overview. IEEE.

Mill, S. (2016). Electric load forecasting: advantages and challenges. `http://engineering.electrical-equipment.org`. [Online; accessed January-2018].

PecanStreet (2018). Dataport. `http://dataport.cloud`. [Online; accessed January-2018].

Valero S, Ortiz M, G. F.-E. N. G. A. M. A. G. E. (2004). Characterization and identification of electrical customers through the use of self-organizing maps and daily load parameters. IEEE.

Wang Zhiguang, O. T. (2015). Encoding time series as images for visual inspection and classification using tiled convolutional neural networks. Association for the Advancement of Artificial Intelligence.

Table 4: MTF, GASF, and GADF of the representative daily profile of the last month of spring 2015 for ID77

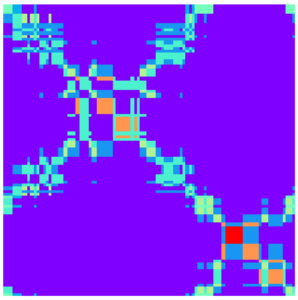
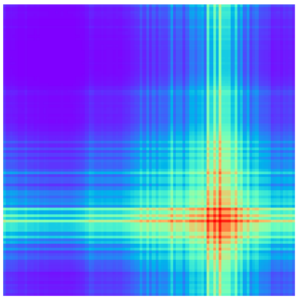
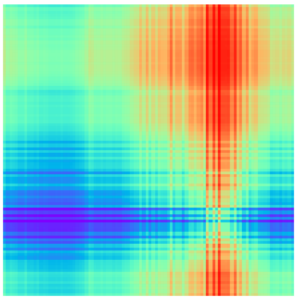
MTF (Q=16)	GASF	GADF
		

Table 5: Classification task results depending on the number of classes and the inputs used for K-means and CNNs

Spring 2015 Classification task							
Number of Clusters	K-means Accuracy			Convolutional Neural Networks			
				Test Accuracy		Train Accuracy	
	Raw TS	GAFs	GAFs+MTF	μ	s	μ	s
6	**0.7096**	0.6108	0.6189	**0.7171**	0.0451	0.7330	0.0525
7	**0.7056**	0.5887	0.6108	**0.6464**	0.0608	0.6523	0.0595
8	**0.6995**	0.5987	0.6230	**0.6969**	0.0607	0.6973	0.0726
9	**0.6673**	0.5947	0.5987	**0.6363**	0.0412	0.6398	0.0241

Table 6: Summer 2015 Forecasting task results depending on the number of classes and the inputs used for K-means and CNNs

Summer 2015 Forecasting task							
Number of Clusters	K-means Accuracy			Convolutional Neural Networks			
				Test Accuracy		Train Accuracy	
	Raw TS	GAFs	GAFs+MTF	μ	s	μ	s
6	**0.6270**	0.5423	0.5947	**0.6464**	0.0439	0.6851	0.0387
7	**0.5967**	0.5242	0.5282	**0.7023**	0.0371	0.7330	0.0345
8	**0.6048**	0.4879	0.5202	**0.6060**	0.0755	0.6574	0.0240
9	**0.5383**	0.4697	0.4959	**0.5151**	0.0521	0.5667	0.0241

Table 7: Forecasting task results depending on the season of the year and using a reduced dataset with consumers from Austin only

Year 2015 Forecasting task					
Season	K-means Accuracy	Convolutional Neural Networks			
		Test Accuracy		Train Accuracy	
	Raw TS	μ	s	μ	s
Summer	**0.6368**	**0.6080**	0.4511	0.6830	0.0345
Fall	**0.5890**	**0.5668**	0.0371	0.7319	0.0586
Winter	**0.5398**	**0.5205**	0.0638	0.7072	0.0423
Spring	**0.3800**	**0.3769**	0.057	0.6761	0.0661

PARTE II

CÓDIGO FUENTE

DataAcquisitionV8.py

DataAcquisitionV8.py

```
"""
Created on Tue May  8 12:52:31 2018
@author: Ignacio Aguirre

DATA ADQUISITION, CLEANING AND PREPROCESSING:

The objective of this script is to obtain the data from Pecan Street, clean it
and compute averages to obtain representative load profile. Dirty ELements wil
l be cleaned
if the loss of data does not reach more than a specified percetage of the whol
e data interval.
There are different type of dirty elemnts:

    Type 1. Elements with smaller size that the data interval required. Some c
onsumers finish
    the agreement with Pecan Street and there is no more data after this date.
 We
    can still average the data available. Some loss of information

    Type 2. Elements with null values in the middle of the season. There might
 be one or
    more than one null value in a row. If there is just one, we average the i+
1h and the i-1h data point.
    If there are more than one in row, we average i+1day and i-
1day data point.

    Type 3. ELements with null values but also with a smaller size.

    Type 4. Elements missing just the last value, we just keep the previous on
e


Once the data is cleaned, the next step is to compute the averages. We split t
he data in two types:

    1. Elements with size=data interval, those are the perfect elements+elemen
ts will null problems that have been cleaned.
    These elements hava a data size = data interval, there we can compute the
averages together

    2. Elements with smaller size than the data interval. We need to compute i
ts averages one at a time.

"""


#Libraries
import psycopg2
from datetime import date
from functionsV2 import daily_profile_V2, consecutive
import numpy as np
import pandas as pd


#Constants
CONST_SAMPLE_SIZE = 96#    24 or 96 depending on the the sample size (1h or 15
-min)
AUSTIN = 0 #If 1 only collects austinians data, 0 collects all data including
austinians

```

```
#Index use vectors positioning
INDEX_CONST_SAMPLE_SIZE = CONST_SAMPLE_SIZE-1

#Percentage of the data missed and accepted to be cleaned
CLEAN_PERCENTAGE = 0.8

#Activate with 1 if you want to obtains weekdays or weekends. Both can be sele
cted but it will take more time
WEEKDAYS = 1
WEEKENDS = 0


#Select the season. Please just select one per time.
WINTER = 0
SPRING = 1
SUMMER = 0
FALL = 0

#The March 11 we loss 1h hour due to the Daylight Saving Time, then we need to
 adjust this issue.
#Also for Fall we gain 1h.
#Just activate it, if your interval of time includes March 11--
> then winter and fall
Winter_DaylightSavingTime = 0

#Do not modify it
timelag = 0

#Dates of study--
> the interval works as [start date, end date] so it does include the first an
d the last day.
if WINTER == 1:

    start_date = date(2014,12,21)
    end_date = date(2015,3,20)

    if Winter_DaylightSavingTime == 1:
        #It will take into account the hour due to Daylight Saving Time, it wo
uld be 1 if sample size is 24 or 4 if it is 96
        timelag = int((CONST_SAMPLE_SIZE)/24)*(-1)


elif SPRING == 1:

    start_date = date(2015,3,21)
    end_date = date(2015,6,20)


elif SUMMER == 1:
    start_date = date(2015,6,21)
    end_date = date(2015,9,20)


elif FALL == 1:

    start_date = date(2015,9,21)
    end_date = date(2015,12,20)
    if Winter_DaylightSavingTime == 1:

                timelag = int((CONST_SAMPLE_SIZE)/24)*(1) #It will take into ac
count the hour due to Daylight Saving Time, it would be 1 if sample size is 24
 or 4 if it is 96


       #String depending on the data interval size
       if CONST_SAMPLE_SIZE == 24:
```

```
            q_table = "university.electricity_egauge_hours"
            q_column_datetime = "localhour"

        elif CONST_SAMPLE_SIZE == 96:
            q_table ="university.electricity_egauge_15min"
            q_column_datetime = "local_15min"

        else:

            print("Sample size not selected")

        #It measures the size of time interval
        days_interval = ((end_date-start_date).days)+1
        data_interval = (CONST_SAMPLE_SIZE*days_interval) + timelag


        #Conection to the database using psycopg2
        conn = psycopg2.connect(host="67.78.67.93",database="postgres", user=""
    , password="", port="5434")
        cur = conn.cursor()


        #Cleaning 1: eliminate all the elements with a different size of the ex
    pected.
        #This includes elements with null elements and elements that we do not
    have data for the full period
        #(they left PecanStreet)

        #Here we just collect ID information of not clean elements
        #Query to check query size
        q_beg = "SELECT dataid, COUNT(use) FROM "
        q_midd_1=" WHERE "+q_column_datetime+" BETWEEN "
        start_date_str = "'""'"+str(start_date)+"'"
        end_date_str = "'""'"+str(end_date)+" 23:45:00'"
        q_end = start_date_str+" AND "+end_date_str+" GROUP BY dataid "
        q_having ="HAVING COUNT(use)<"+str(data_interval)
        q_order =" ORDER BY dataid ASC"

        query = q_beg+q_table+q_midd_1+q_end+q_having+q_order+";"
        # ================================================
        # SELECT  dataid, COUNT(use) FROM university.electricity_egauge_15min
        # WHERE local_15min BETWEEN 'startdate' AND 'end date'
        # GROUP BY dataid
        # HAVING COUNT(use)<= data interval
        # ORDER BY dataid ASC;
        # ================================================


        #Positioning in the database, and query extraction
        cur.execute(query)
        imperfect_consumers = cur.fetchall()

        #Consumer with a size < data interval
        imperfect_consumers_np = np.array(imperfect_consumers)
        imperfect_consumers_np_datainterval = imperfect_consumers_np[:,1].astyp
    e(int)


        #Download the whole datbase, it includes clean and not clean elements.
        q_beg = "SELECT "
        q_midd_1 = ", use, dataid FROM "
        q_midd_12 = " WHERE "+q_column_datetime+" BETWEEN "
        q_end = start_date_str+" AND "+end_date_str
        q_order =" ORDER BY dataid ASC, local_15min ASC"

```

```
        query = q_beg+q_column_datetime+q_midd_1+q_table+q_midd_12+q_end+q_order+";"
        # ================================================
        # SELECT local_15min, use, dataid
        # FROM university.electricity_egauge_15min
        # WHERE local_15min BETWEEN 'startdate' AND 'end date'
        # ORDER BY dataid ASC, local_15min ASC;
        # ================================================
        cur.execute(query)
        consumers = cur.fetchall()
        consumers_np = np.array(consumers)

        #Disposal consumer is an array where False are the elements with a different size. These elements are removed
        disposal_consumers_bool = np.logical_not(np.in1d(consumers_np[:,2],imperfect_consumers_np[:,0]))

        #We apply the mask. Clean_consumers have all the data.
        perfect_consumers = consumers_np[disposal_consumers_bool]

        #It is neccesary to transform the power use from decimal to float to do numpy operations.
        perfect_consumers_use = perfect_consumers[:,1].astype(float)

        #Final consumer is the matrix where we add the complete elements(perfect elements + elements that have been cleaned (null numbers) and have the data interval size):
        Final_consumers = np.vstack((perfect_consumers[:,0],perfect_consumers_use,perfect_consumers[:,2]))
        Final_consumers = Final_consumers.transpose()


        #Let's clean elements that have at least the PERCENTAGE% (this can be changed) of the data:
        #First creat the mask
        cleaning_elements_bool = imperfect_consumers_np_datainterval >= int(data_interval*CLEAN_PERCENTAGE)
        #Apply the mask. These are elemets that are cleaned
        cleaning_consumers = imperfect_consumers_np[cleaning_elements_bool]


        # Cleaning 2: Here the purpose is to recover elemnts with null values. T
        # --> Null values will be average using the values of i+1h and i-1h or just keeping i-1h if we do not have the next point


        #Now let's recover consumer with null elements:
        #Query to check null rows


        q_beg = "SELECT "
        q_midd_1 = ", use, dataid FROM "
        q_midd_2 = " AND "+q_column_datetime+" BETWEEN "
        q_where = " WHERE dataid="
        q_midd_12 = " AND "+q_column_datetime+" BETWEEN "
        start_date_str = "'"+str(start_date)+"'"
        end_date_str = "'"+str(end_date)+" 23:45:00'"
        q_end = start_date_str+" AND "+end_date_str
        q_order =" ORDER BY dataid ASC, local_15min ASC"


        M_cleaned_size = []


```

```
223.       #Matrix where we put the dirty elements type 1 and 3.
224.       M_1 = []
225.       M_3 = []
226.       M_daily_profile = np.array([])
227.
228.
229.       #I use H and L, to know in what position of [0,95] my point is. 0 corre
    sponds to 00:00:00 and 95 to 23:45:00
230.       H = np.arange(0,CONST_SAMPLE_SIZE)
231.       L = np.tile(H,days_interval) #It concatenates vector from 0 to 95 with
    the size of the data interval
232.
233.
234.       #We run this code once for every element that must be cleaned.
235.       for i in range(0,len(cleaning_consumers)):
236.
237.
238.           #Run a with only one dirty consumer at a time
239.           identification = str(cleaning_consumers[i,0])
240.           query = q_beg+q_column_datetime+q_midd_1+q_table+q_where+identifica
    tion+q_midd_12+q_end+q_order+";"
241.           # ================================================
242.           # SELECT local_15min, use, dataid
243.           # FROM university.electricity_egauge_15min
244.           # WHERE dataid=identification
245.           # AND local_15min BETWEEN '2015-03-21' AND '2015-03-23 23:45:00'
246.           # ORDER BY dataid ASC, local_15min ASC;
247.           # ================================================
248.           cur.execute(query)
249.           consumer_in_process = cur.fetchall()
250.           consumer_in_process_np = np.array(consumer_in_process)
251.
252.           #1D vectors
253.           consumer_in_process_time = consumer_in_process_np[:,0]
254.           consumer_in_process_use = consumer_in_process_np[:,1].astype(float)

255.           consumer_in_process_ID = consumer_in_process_np[:,2].astype(int)
256.
257.           #Let's create a mask to find null elements
258.           consumer_in_process_bool = np.isnan(consumer_in_process_use)
259.           #It gives you the true positions where there are null elements
260.           indexes = np.where(consumer_in_process_bool)[0]
261.
262.           #We will use this variable to average the values that form the null
     value
263.           values = np.array([], dtype=float)
264.
265.
266.           #It does not have null elements just elements missing at the end:
267.           if len(indexes)==0:
268.
269.               #if the consumer is only missing the last element, we copy the
    previous number
270.               if cleaning_consumers[i,1] == (data_interval-1):
271.
272.                   consumer_in_process_time = np.append(consumer_in_process_ti
    me,consumer_in_process_time[-1])
273.                   consumer_in_process_use = np.append(consumer_in_process_use
    ,consumer_in_process_use[-1])
274.                   consumer_in_process_ID = np.append(consumer_in_process_ID,c
    onsumer_in_process_ID[-1])
275.
276.                   consumer_in_process_transform = np.vstack((consumer_in_proc
    ess_time,consumer_in_process_use,consumer_in_process_ID))
277.                   consumer_in_process_transform = consumer_in_process_transfo
    rm.transpose()
```

```
278.
279.                     #Final_consumers = np.concatenate((Final_consumers, consume
    r_in_process_transform), axis=0)
280.
281.                 #if consumers have smaller size but not null elements. We elimi
    ntate the elemnts of the last day
282.                 elif (len(consumer_in_process_use) < (data_interval-1)):
283.
284.                     index = cleaning_consumers[i,1]-1
285.                     mask = np.ones(len(consumer_in_process_use), dtype=bool)
286.                     mask_index = index-L[index]
287.                     mask[mask_index:] = False
288.                     #check = L[mask_index-1]
289.
290.                     consumer_in_process_use = consumer_in_process_use[mask]
291.                     consumer_in_process_np = consumer_in_process_np[mask]
292.
293.                     M_daily_profile  = daily_profile_V2(consumer_in_process_np,
     len(consumer_in_process_use), timelag, CONST_SAMPLE_SIZE, WEEKDAYS, WEEKENDS)

294.                     M_prueba = M_daily_profile[0]
295.                     M_1.append(M_daily_profile)
296.
297.
298.
299.             #There are also elemnts that at the end have null values, in real i
    s that they have a smaller size
300.             elif (indexes[0] == cleaning_consumers[i,1]):
301.                 mask = np.ones(len(consumer_in_process_use), dtype=bool)
302.                 mask[indexes[0]:] = False
303.
304.                 consumer_in_process_np = consumer_in_process_np[mask]
305.                 consumer_in_process_use = consumer_in_process_use[mask]
306.
307.                 #The last no null element is in the middle of a day, we elimina
    te that day
308.                 if L[cleaning_consumers[i,1]-1] != (INDEX_CONST_SAMPLE_SIZE):
309.
310.                     index = cleaning_consumers[i,1]-1
311.                     mask = np.ones(len(consumer_in_process_use), dtype=bool)
312.                     mask_index = index-L[index]
313.                     mask[mask_index:] = False
314.
315.                     consumer_in_process_np = consumer_in_process_np[mask]
316.
317.                     consumer_in_process_use = consumer_in_process_use[mask]
318.                     consumer_in_process_time = consumer_in_process_np[:,0]
319.                     consumer_in_process_ID = consumer_in_process_np[:,2].astype
    (int)
320.
321.                     Final_consumer_in_cleaned = np.vstack((consumer_in_process_
    time,consumer_in_process_use,consumer_in_process_ID))
322.                     Final_consumer_in_cleaned = Final_consumer_in_cleaned.trans
    pose()
323.
324.                 M_daily_profile  = daily_profile_V2(Final_consumer_in_cleaned,
    len(Final_consumer_in_cleaned), timelag, CONST_SAMPLE_SIZE, WEEKDAYS, WEEKENDS
    )
325.                 M_1.append(M_daily_profile)
326.
327.             #If the consumer has only one null element. We used the value of 1h
     before and after.
328.             elif len(indexes) == 1:
329.
330.                 prev_hour =  indexes-1
331.                 next_hour =  indexes+1
```

```
                values = np.append(values,[consumer_in_process_use[prev_hour],consumer_in_process_use[next_hour]])
                average = np.average(values)
                np.put(consumer_in_process_use,indexes,average)



            #If the consumer has null elements and maybe smaller size. We used the value of 1h before and after.
            else:

                #distribution of null elements, created function to measure the number of consecutive elements
                distribution = consecutive(indexes)
                #Indexes are referenced to the original array
                mask = np.ones(len(consumer_in_process_use), dtype=bool)

                #If there are more than 2 consecutive hours without data, we eliminate the whole day
                for j in range (0, len(distribution)):

                    #If it is an isolated elemnt lets compute average between i+1h and i-1h
                    if len(distribution[j]) == 1:

                        prev_hour =  int(distribution[j][0]-1)
                        next_hour =  int(distribution[j][0]+1)

                        values = np.append(values,[consumer_in_process_use[prev_hour],consumer_in_process_use[next_hour]])
                        average = np.average(values)
                        consumer_in_process_use[distribution[j][0]] = average

                        #np.put(consumer_in_process_use,indexes[j],average)

                    #We maintain the previous number a maximum of two hours
                    elif 9 > len(distribution[j]) > 1:

                        prev_hour = int(distribution[j][0]-1)
                        value = consumer_in_process_use[prev_hour]
                        consumer_in_process_use[distribution[j][0]:distribution[j][-1]+1] = value

                    elif len(distribution[j]) >= 9:

                        if ((L[distribution[j][0]])==0) and ((L[distribution[j][-1]]) == INDEX_CONST_SAMPLE_SIZE):

                            mask[distribution[j][0]:distribution[j][-1]+1] = False

                        else:

                            #if I take out an intermediate block, I need to take into account what i leave at both sides of the block
                            start = distribution[j][0]-L[distribution[j][0]]
                            finish = distribution[j][-1]+(INDEX_CONST_SAMPLE_SIZE-L[distribution[j][-1]])
                            #We can reconstruct the first part, if need lees than 2h of data
                            if (L[distribution[j][0]] >= (CONST_SAMPLE_SIZE-8)) :


                                prev_hour = int(distribution[j][0]-1)
                                value = consumer_in_process_use[prev_hour]
```

```
                                index_max = INDEX_CONST_SAMPLE_SIZE - L[distrib
    ution[j][0]]
                                consumer_in_process_use[distribution[j][0]:dist
    ribution[j][index_max]+1] = value
                                check = L[distribution[j][index_max+1]]

                                start = distribution[j][index_max+1]


                            if (L[distribution[j][-1]] <= 7):

                                next_hour = int(distribution[j][-1]+1)
                                value = consumer_in_process_use[next_hour]
                                index_min = -1-L[distribution[j][-1]]
                                consumer_in_process_use[ (distribution[j][index
    _min]) : ((distribution[j][-1])+1)] = value
                                check = L[distribution[j][index_min-1]]

                                finish = distribution[j][index_min-1]


                            mask[start:finish+1] = False

                #out of the for loop

                consumer_in_process_np = consumer_in_process_np[mask]
                consumer_in_process_use = consumer_in_process_use[mask]
                consumer_in_process_time = consumer_in_process_np[:,0]
                consumer_in_process_ID = consumer_in_process_np[:,2].astype(int
    )

                Final_consumer_in_cleaned = np.vstack((consumer_in_process_time
    ,consumer_in_process_use,consumer_in_process_ID))
                Final_consumer_in_cleaned = Final_consumer_in_cleaned.transpose
    ()


                if (len(consumer_in_process_use) < (data_interval-1)):


                    M_daily_profile  = daily_profile_V2(Final_consumer_in_clean
    ed, len(Final_consumer_in_cleaned), timelag, CONST_SAMPLE_SIZE, WEEKDAYS, WEEK
    ENDS)
                    M_prueba = M_daily_profile[0]
                    M_1.append(M_daily_profile)

                else:

                    consumer_in_process_time = consumer_in_process_np[:,0]
                    consumer_in_process_ID = consumer_in_process_np[:,2].astype
    (int)

                    consumer_in_process_transform = np.vstack((consumer_in_proc
    ess_time,consumer_in_process_use,consumer_in_process_ID))
                    consumer_in_process_transform = consumer_in_process_transfo
    rm.transpose()

                    Final_consumers = np.concatenate((Final_consumers, consumer
    _in_process_transform), axis=0)
```

```
        M_1_np = np.array(M_1)
        M_2_np = M_1_np[:,0,:,:]
        #daily_profile_V2 compute the averages to obtain the representative pro
    files
        M_daily_profile  = daily_profile_V2(Final_consumers, data_interval, tim
    elag, CONST_SAMPLE_SIZE, WEEKDAYS, WEEKENDS)
        M_3_np = np.array(M_daily_profile)


        #This is the final matrix
        M_4_np = np.concatenate((M_3_np,M_2_np),axis=0)

        # ================================================
        # #t prints False if there aren't null elements
        # for i in range (0, len(M_4_np)):
        #     for j in range (0,96):
        #         print(np.isnan(M_4_np[i,j,1]).any())
        # ================================================


        #It just takes into accounto consumers from Austin
        if AUSTIN ==1:
            #Update the Austinians, the excel file must be in the same folder
            df = pd.read_excel(open('AustiniansID.xlsx','rb'), sheet_name='Shee
    t1')
            dataID_austinians = df.values

            ID_M_4 = M_4_np[:,0,2]
            dataID_np = np.intersect1d(ID_M_4,dataID_austinians)
            mask_austinian = np.in1d(ID_M_4,dataID_np)
            M_4_np = M_4_np[mask_austinian]


        #Uncomment it in case you want to save the results

        #np.save('AV_consumers_XXXX_15min.npy',M_4_np)
```

ClusteringV2.py

ClusteringV2.py

```
"""
Created on Thu Mar 15 11:39:45 2018
@author: Ignacio Aguirre Panadero

DESCRIPTION:

This script will generate the labels for the clustering k-means. It will also
depict the cluster centroids and the elements of each class. It will also provides
compactness measures and statistics. It is necessary to select to seasons because the code
will only take into account elements that appear in both periods as the final objective is to classify them.

"""

#Libraries
import numpy as np
from sklearn.cluster import KMeans
import matplotlib.pyplot as plt
from pyts.transformation import StandardScaler
from functionsV2 import  MIA,CDI


#Constants
CONST_SAMPLE_SIZE = 24
N_CLUSTERS = 7



#Activation variables
Z_NORM = 1 #1 Z-Normalization, 0 for peak normalization

AUSTIN = 0 #1 if you want to the use data of only austininians

#The clustering will consider the season with a 1. Select two seasons when running

SUMMER = 1
SPRING = 1
FALL = 0
WINTER = 0


#Initialization figures
f=0



#Load the data
if AUSTIN == 1:

    if SUMMER == 1:
        if CONST_SAMPLE_SIZE == 96:
            Consumer_summer15 = np.load('C:/Users/aguir/AnacondaProjects/Summer/AV_Austin_consumer_Wdays_15min_Summer15V2.npy')
        else:
            Consumer_summer15 = np.load('C:/Users/aguir/AnacondaProjects/Summer/AV_Austin_consumer_Wdays_hours_Summer15V1.npy')

    if SPRING == 1:
```

```
        if CONST_SAMPLE_SIZE == 96:
            Consumer_spring15 = np.load('C:/Users/aguir/AnacondaProjects/Spring/AV_Austin_consumer_Wdays_15min_Spring15V1.npy')
        else:
            Consumer_spring15 = np.load('C:/Users/aguir/AnacondaProjects/Spring/AV_Austin_consumer_Wdays_hours_Spring15V1.npy')


    if FALL == 1:
        if CONST_SAMPLE_SIZE == 96:
            Consumer_fall15 = np.load('C:/Users/aguir/AnacondaProjects/Fall/AV_Austin_consumer_Wdays_15min_Fall15V1.npy')
        else:
            Consumer_fall15 = np.load('C:/Users/aguir/AnacondaProjects/Fall/AV_Austin_consumer_Wdays_hours_Fall15V1.npy')

    if WINTER == 1:
        if CONST_SAMPLE_SIZE == 96:
            Consumer_winter15 = np.load('C:/Users/aguir/AnacondaProjects/Winter/AV_Austin_consumer_Wdays_15min_Winter15V2.npy')
        else:
            Consumer_winter15 = np.load('C:/Users/aguir/AnacondaProjects/Winter/AV_Austin_consumer_Wdays_hours_Winter15V2.npy')

else:

    if SUMMER ==1:
        if CONST_SAMPLE_SIZE == 96:
            Consumer_summer15 = np.load('C:/Users/aguir/AnacondaProjects/Summer/AV_consumer_Wdays_15min_Summer15V2.npy')
        else:
            Consumer_summer15 = np.load('C:/Users/aguir/AnacondaProjects/Summer/AV_consumer_Wdays_hours_Summer15V2.npy')

    if SPRING ==1:
        if CONST_SAMPLE_SIZE == 96:
            Consumer_spring15 = np.load('C:/Users/aguir/AnacondaProjects/Spring/AV_consumer_Wdays_15min_Spring15V1.npy')
        else:
            Consumer_spring15 = np.load('C:/Users/aguir/AnacondaProjects/Spring/AV_consumer_Wdays_hours_Spring15V1.npy')


    if FALL ==1:
        if CONST_SAMPLE_SIZE == 96:
            Consumer_fall15 = np.load('C:/Users/aguir/AnacondaProjects/Fall/AV_consumer_Wdays_15min_Fall15V1.npy')
        else:
            Consumer_fall15 = np.load('C:/Users/aguir/AnacondaProjects/Fall/AV_consumer_Wdays_hours_Fall15V1.npy')

    if WINTER ==1:
        if CONST_SAMPLE_SIZE == 96:
            Consumer_winter15 = np.load('C:/Users/aguir/AnacondaProjects/Winter/AV_consumer_Wdays_15min_Winter15V2.npy')
        else:
            Consumer_winter15 = np.load('C:/Users/aguir/AnacondaProjects/Winter/AV_consumer_Wdays_hours_Winter15V2.npy')


       #It re-order the seasons in season1 and season2
       if SUMMER == 1:
           season1 = Consumer_summer15

           if SPRING == 1:
```

```
                season2 = Consumer_spring15

            elif FALL == 1:
                season2 = Consumer_fall15
            elif WINTER ==1:
                season2 = Consumer_winter15

        elif SPRING == 1:
            season1 = Consumer_spring15

            if FALL == 1:
                season2 = Consumer_fall15

            elif WINTER ==1:
                season2 = Consumer_winter15

        elif FALL == 1:
            season1 = Consumer_fall15
            season2 = Consumer_winter15



        #Let's generate a vector with dataID
        season1_ID = season1[:,0,2]
        season2_ID = season2[:,0,2]

        #Vector with the common consumer of both period.
        #There are consumer that finishes the agreement with Pecan Street in th
    e middle of the spring or season2 season.
        Common_consumer = np.intersect1d(season1_ID,season2_ID)


        #It gives you a boolean vector, where the TRUE positions are common ele
    ments of both periods
        Common_consumer_season1_bool = np.in1d(season1_ID,Common_consumer)
        Common_consumer_season2_bool = np.in1d(season2_ID,Common_consumer)

        #Let's eliminate the consumer that are not common for both periods from
     the data
        season1 = season1[Common_consumer_season1_bool]
        season2 = season2[Common_consumer_season2_bool]



        #Vector with the client's power use
        season1_use =  np.vstack(season1[:,:,1]).astype(float)
        season2_use =  np.vstack(season2[:,:,1]).astype(float)



        #Z-Normalization
        if Z_NORM == 1 :

            standardscaler = StandardScaler(epsilon=1e-2)
            season1_use = standardscaler.transform(season1_use)
            season2_use = standardscaler.transform(season2_use)


        #Peak-Normalization
        else:
            for i in range (0, len(season1_use)):
                p_peak = np.amax(season1_use[i,:])
                a = season1_use[i,:]/p_peak
                season1_use[i,:] = a

                p_peak = np.amax(season2_use[i,:])
```

```
171.                a = season2_use[i,:]/p_peak
172.                season2_use[i,:] = a
173.
174.
175.
176.
177.        #Number of classes that give the best solution
178.        n_clusters = []
179.        inertia = []
180.        j = 1
181.        for i in range (0,10):
182.            n_clusters.append(j)
183.            kmeans = KMeans(n_clusters= n_clusters[i], random_state=0).fit(seas
    on1_use)
184.            inertia.append(kmeans.inertia_)
185.            j+=1
186.
187.
188.        X = []
189.        Y = []
190.        plt.figure(f)
191.        f+=1
192.        for n in range(0,10):
193.            X.insert(n,n_clusters[n])
194.            Y.insert(n,inertia[n])
195.            plt.plot(X, Y, 'ro')
196.        plt.title("Selection of the number of classes")
197.        plt.ylabel("Error")
198.        plt.xlabel("k number of cluster")
199.
200.
201.
202.
203.
204.        #Computing the clustering
205.        kmeans = KMeans(n_clusters=N_CLUSTERS, random_state=0).fit(season1_use)

206.        labels_season1 = kmeans.labels_
207.        centers_season1 = kmeans.cluster_centers_
208.        kmeans = KMeans(n_clusters=N_CLUSTERS, random_state=0).fit(season2_use)

209.        labels_season2 = kmeans.labels_
210.        centers_season2 = kmeans.cluster_centers_
211.
212.        #Clustering compactness
213.        MIA_per_cluster_season1= np.array([])
214.        MIA_per_cluster_season2 = np.array([])
215.
216.
217.        #Functions to measure compactness
218.        mia_season1, MIA_per_cluster_season1 = MIA(labels_season1, centers_seas
    on1, season1_use, N_CLUSTERS, CONST_SAMPLE_SIZE)
219.        mia_season2, MIA_per_cluster_season2 = MIA(labels_season2, centers_seas
    on2, season2_use, N_CLUSTERS, CONST_SAMPLE_SIZE)
220.
221.        cdi_season1 = CDI(labels_season1, centers_season1, season1_use, N_CLUST
    ERS, CONST_SAMPLE_SIZE)
222.        cdi_season2 = CDI(labels_season2, centers_season2, season2_use, N_CLUST
    ERS, CONST_SAMPLE_SIZE)
223.
224.
225.
226.        #PLot centroids
227.        plt.figure(f)
228.        f+=1
229.
```

```
230.
231.       plt.title('Centroides Verano 2015 ')
232.       for g in range (0,N_CLUSTERS):
233.           plt.plot(centers_season1[g], label="C%s" % (g), linewidth=2.0)
234.
235.       #plt.legend(bbox_to_anchor=(0., 1.02, 1., .102), loc=3,
236.                  #ncol=7, mode="expand", borderaxespad=0.)
237.       plt.legend(bbox_to_anchor=(1.05, 1), loc=2, borderaxespad=0.)
238.       plt.ylabel('Power (kW, Peak-normalized)', fontsize=12)
239.       plt.xlabel('Time(h)', fontsize=12)
240.
241.
242.
243.       plt.figure(f)
244.       f+=1
245.       plt.title('Centroides Primavera 2015')
246.       for g in range (0,N_CLUSTERS):
247.
248.           plt.plot(centers_season2[g], label="C%s" % (g), linewidth=2.0)
249.
250.       #plt.legend(bbox_to_anchor=(0., 1.02, 1., .102), loc=3,
251.                  #ncol=7, mode="expand", borderaxespad=0.)
252.       plt.legend(bbox_to_anchor=(1.05, 1), loc=2, borderaxespad=0.)
253.       plt.ylabel('Power (kW, Peak-normalized)', fontsize=12)
254.       plt.xlabel('Time(h)', fontsize=12)
255.       plt.show()
256.
257.
258.
259.       M_avg = [] #array to collect the average of every cluster centroid
260.       M_std = []
261.
262.       M_clusters_season1 = [] #arrray to collect the season1 classes
263.       M_clusters_season2 = [] #arrray to collect the season2 classes
264.       #Size of each cluster
265.       for i in range(0,N_CLUSTERS):
266.           cluster_s1 = []
267.           clusterID_s1 = [] #It colects the ID of each cluster for season1
268.           fila_s1 = []
269.           cluster_s2 = []
270.           clusterID_s2 = [] #It colects the ID of each cluster for season2
271.           fila_s2 = []
272.
273.
274.           for j in range (0, len(labels_season1)):
275.
276.               if labels_season1[j] == i:
277.                   clusterID_s1.append(Common_consumer[j])
278.                   for k in range(0,CONST_SAMPLE_SIZE):
279.                       fila_s1.append(season1_use[j][k])
280.                   cluster_s1.append(fila_s1)
281.                   fila_s1 = []
282.
283.               if labels_season2[j] == i:
284.                   clusterID_s2.append(Common_consumer[j])
285.                   for k in range(0,CONST_SAMPLE_SIZE):
286.                       fila_s2.append(season2_use[j][k])
287.                   cluster_s2.append(fila_s2)
288.                   fila_s2 = []
289.
290.
291.           cluster_s1_np = np.array(cluster_s1)
292.           cluster_s2_np = np.array(cluster_s2)
293.           Std = np.std(cluster_s1_np, axis=0)
294.           Avg = np.average(cluster_s1_np, axis=0)
295.           M_std.append(Std)
```

```
            M_avg.append(Avg)
            M_clusters_season1.append(cluster_s1_np)
            M_clusters_season2.append(cluster_s2_np)

            #The comand exec creates variables. Uncomment the variables required,
            exec("clusterID_W%s=clusterID_s1" % (i))
            exec("clusterID_S%s=clusterID_s2" % (i))
            #exec("Cluster_W%s=cluster_s1_np" % (i))
            #exec("Cluster_S%s=cluster_s2_np" % (i))
            #exec("Std_S1%s=Std" % (i))
            #exec("Avg_S1%s=Avg" % (i))

        #Plotting the clusters

        X = []
        H = np.arange(96)


        for i in range (0,len(M_clusters_season1)):
            plt.figure(f)
            f+=1
            plt.title(('Cluster%s Verano 2015')%(i))
            for j in range(0,len(M_clusters_season1[i])):
                Y= M_clusters_season1[i][j,:]
                plt.plot(H,Y, linestyle='-', color='b',)

            plt.plot(H,centers_season1[i], linestyle='--', color='r', label="average")
            plt.legend(bbox_to_anchor=(0.4, 0.95), loc=2, borderaxespad=0.)
            plt.ylabel("Power (kW (Peak-normalized))")
            plt.xlabel("Time (h)")
            plt.grid(True)

        for i in range (0,len(M_clusters_season2)):
            plt.figure(f)
            f+=1
            plt.title(('Cluster%s Primavera 2015')%(i))
            for j in range(0,len(M_clusters_season2[i])):
                Y= M_clusters_season2[i][j,:]
                plt.plot(H,Y, linestyle='-', color='b',)


            plt.plot(H,centers_season2[i], linestyle='--', color='r', label="average")
            plt.legend(bbox_to_anchor=(0.4, 0.95), loc=2, borderaxespad=0.)
            plt.ylabel("Power (kW (Z-normalized))")
            plt.xlabel("Time (h)")
            plt.grid(True)



        #Here we check if the classes of season1 and season2 have same elements or if the classes depends on each season.
        #The accuracy give us the percentage of elements that are in the same cluster for both seasons
        accuracy_v = []
        accuracy = 0

        for w in range (0,len(Common_consumer)):

            dataid_cluster_season115 = []
            dataid_cluster_season215 = []

            for e in range (0,len(Common_consumer)):

```

```
357.
358.                 if (labels_season1[w]==labels_season1[e]) & (w!=e):
359.                     dataid_cluster_season115.append(Common_consumer[e])
360.                 if (labels_season2[w]==labels_season2[e]) & (w!=e):
361.                     dataid_cluster_season215.append(Common_consumer[e])
362.
363.             Common_consumer_cluster = np.intersect1d(dataid_cluster_season115,d
    ataid_cluster_season215)
364.
365.             accuracy += len(Common_consumer_cluster)/(len(dataid_cluster_season
    115))
366.             #accuracy_v.append(accuracy)
367.
368.
369.
370.         Average_accuracy = (accuracy)/float(len(Common_consumer))
371.         print(Average_accuracy)
372.
373.
374.         ###RESULTS TO SAVE--> labels
375.         #Uncomment it if you want to save values
376.
377.         #np.save('labels_XXXX_15min,labels_season1)
378.         #np.save('labels_XXXX_15min,labels_season2)
```

MTF-GAF_V6.py

MTF-GAF_V6.py

```
"""
Created on Sun Mar 18 19:17:03 2018
@author: Ignacio Aguirre Panadero

Description:

With this script we can generate the MTF and GAFs for the month use to predict

the next season. It uses the library pyts to generate the MTF and GAFs, then i
t must be installed
(https://github.com/johannfaouzi/pyts).

MTF_1--> it will refer to the MTF of season 1
MTF_2--> it will refer to the MTF of season 2

MTF_3--
> it will refer to the MTF of the last month month of season 2 used to forecas
t.


"""




#Libraries
import numpy as np
from pyts.transformation import MTF
from pyts.visualization import plot_mtf
from pyts.transformation import GASF, GADF
from pyts.visualization import plot_gasf
from pyts.visualization import plot_gadf
from pyts.transformation import StandardScaler
from pyts.visualization import plot_standardscaler
import matplotlib.pyplot as plt


#Constants
CONST_SAMPLE_SIZE = 96


#Select number of cluster and quantiles bins for the MTF
N_CLUSTERS = 7
N_BINS = 16

#Activation variables
Z_NORM = 1 #1 Z-Normalization, 0 for peak normalization
AUSTIN = 0

#Select 1 season. If you select season we will be predicting summer using spri
ng. If you select winter, we will be predicting winter using fall.
SUMMER = 1
SPRING = 0
FALL = 0
WINTER = 0


#Initialization figures
f=0


```

```
#Load the data

#Load the data. Season 1 is always the one to be predicted whereas season2 is the one we use to predict.

if AUSTIN == 1:

    if SUMMER ==1:
        if CONST_SAMPLE_SIZE == 96:

            season1 = np.load('C:/Users/aguir/AnacondaProjects/Summer/AV_Austin_consumer_Wdays_15min_Summer15V2.npy')
            season2 = np.load('C:/Users/aguir/AnacondaProjects/Spring/AV_Austin_consumer_Wdays_15min_Spring15V1.npy')


            season2_3rdmonth = np.load('C:/Users/aguir/AnacondaProjects/Spring/AV_Austin_consumer_Wdays_15min_3rdmonth_Spring15V1.npy')
        else:

            season1 = np.load('C:/Users/aguir/AnacondaProjects/Summer/AV_Austin_consumer_Wdays_hours_Summer15V2.npy')
            season2 = np.load('C:/Users/aguir/AnacondaProjects/Spring/AV_Austin_consumer_Wdays_hours_Spring15V1.npy')


            season2_3rdmonth = np.load('C:/Users/aguir/AnacondaProjects/Spring/AV_Austin_consumer_Wdays_15min_3rdmonth_Spring15V1.npy')

    if SPRING ==1:
        if CONST_SAMPLE_SIZE == 96:
            season1 = np.load('C:/Users/aguir/AnacondaProjects/Spring/AV_Austin_consumer_Wdays_15min_Spring15V1.npy')
            season2 = np.load('C:/Users/aguir/AnacondaProjects/Winter/AV_Austin_consumer_Wdays_15min_Winter15V2.npy')


            season2_3rdmonth = np.load('C:/Users/aguir/AnacondaProjects/Winter/AV_Austin_consumer_Wdays_15min_3rdmonth_Winter15V1.npy')
        else:
            season1 = np.load('C:/Users/aguir/AnacondaProjects/Spring/AV_Austin_consumer_Wdays_hours_Spring15V1.npy')
            season2 = np.load('C:/Users/aguir/AnacondaProjects/Winter/AV_Austin_consumer_Wdays_hours_Winter15V2.npy')


            season2_3rdmonth = np.load('C:/Users/aguir/AnacondaProjects/Winter/AV_Austin_consumer_Wdays_hours_3rdmonth_Winter15V1.npy')

    if FALL ==1:
        if CONST_SAMPLE_SIZE == 96:
            season1 = np.load('C:/Users/aguir/AnacondaProjects/Fall/AV_Austin_consumer_Wdays_15min_Fall15V1.npy')
            season2 = np.load('C:/Users/aguir/AnacondaProjects/Summer/AV_Austin_consumer_Wdays_15min_Summer15V2.npy')


                    season2_3rdmonth = np.load('C:/Users/aguir/AnacondaProjects/Summer/AV_Austin_consumer_Wdays_15min_3rdmonth_Summer15V1.npy')
                else:
                    season1 = np.load('C:/Users/aguir/AnacondaProjects/Fall/AV_Austin_consumer_Wdays_hours_Fall15V1.npy')
                    season2 = np.load('C:/Users/aguir/AnacondaProjects/Summer/AV_Austin_consumer_Wdays_hours_Summer15V2.npy')


```

```
                    season2_3rdmonth = np.load('C:/Users/aguir/AnacondaProjects/Summer/AV_Austin_consumer_Wdays_hours_3rdmonth_Summer15V1.npy')
            if WINTER ==1:
                if CONST_SAMPLE_SIZE == 96:
                    season1 = np.load('C:/Users/aguir/AnacondaProjects/Winter/AV_Austin_consumer_Wdays_15min_Winter15V2.npy')
                    season2 = np.load('C:/Users/aguir/AnacondaProjects/Fall/AV_Austin_consumer_Wdays_15min_Fall15V1.npy')


                    season2_3rdmonth = np.load('C:/Users/aguir/AnacondaProjects/Fall/AV_Austin_consumer_Wdays_15min_3rdmonth_FAll15V1.npy')
                else:
                    season1 = np.load('C:/Users/aguir/AnacondaProjects/Winter/AV_Austin_consumer_Wdays_hours_Winter15V2.npy')
                    season2 = np.load('C:/Users/aguir/AnacondaProjects/Fall/AV_Austin_consumer_Wdays_hours_Fall15V1.npy')

                    season2_3rdmonth = np.load('C:/Users/aguir/AnacondaProjects/Fall/AV_Austin_consumer_Wdays_hours_3rdmonth_FAll15V1.npy')

        else:

            if SUMMER == 1:
                if CONST_SAMPLE_SIZE == 96:
                    season1 = np.load('C:/Users/aguir/AnacondaProjects/Summer/AV_consumer_Wdays_15min_Summer15V2.npy')
                    season2 = np.load('C:/Users/aguir/AnacondaProjects/Spring/AV_consumer_Wdays_15min_Spring15V1.npy')


                    season2_3rdmonth = np.load('C:/Users/aguir/AnacondaProjects/Spring/AV_consumer_Wdays_15min_3rdmonth_Spring15V1.npy')
                else:
                    season1 = np.load('C:/Users/aguir/AnacondaProjects/Summer/AV_consumer_Wdays_hours_Summer15V2.npy')
                    season2 = np.load('C:/Users/aguir/AnacondaProjects/Spring/AV_consumer_Wdays_hours_Spring15V1.npy')


                    season2_3rdmonth = np.load('C:/Users/aguir/AnacondaProjects/Spring/AV_consumer_Wdays_hours_3rdmonth_Spring15V1.npy')

            if SPRING ==1:
                if CONST_SAMPLE_SIZE == 96:
                    season1 = np.load('C:/Users/aguir/AnacondaProjects/Spring/AV_consumer_Wdays_15min_Spring15V1.npy')
                    season2 = np.load('C:/Users/aguir/AnacondaProjects/Winter/AV_consumer_Wdays_15min_Winter15V1.npy')


                    season2_3rdmonth = np.load('C:/Users/aguir/AnacondaProjects/Winter/AV_consumer_Wdays_15min_3rdmonth_Winter15V1.npy')
                else:
                    season1 = np.load('C:/Users/aguir/AnacondaProjects/Spring/AV_consumer_Wdays_hours_Spring15V1.npy')
                    season2 = np.load('C:/Users/aguir/AnacondaProjects/Winter/AV_consumer_Wdays_hours_Winter15V1.npy')


                    season2_3rdmonth = np.load('C:/Users/aguir/AnacondaProjects/Winter/AV_consumer_Wdays_hours_3rdmonth_Winter15V1.npy')


            if FALL ==1:
                if CONST_SAMPLE_SIZE == 96:
```

```
                    season1 = np.load('C:/Users/aguir/AnacondaProjects/Fall/AV_consumer_Wdays_15min_Fall15V1.npy')
                    season2 = np.load('C:/Users/aguir/AnacondaProjects/Summer/AV_consumer_Wdays_15min_Summer15V2.npy')


                    season2_3rdmonth = np.load('C:/Users/aguir/AnacondaProjects/Summer/AV_consumer_Wdays_15min_3rdmonth_Summer15V1.npy')
                else:
                    season1 = np.load('C:/Users/aguir/AnacondaProjects/Fall/AV_consumer_Wdays_hours_Fall15V1.npy')
                    season2 = np.load('C:/Users/aguir/AnacondaProjects/Summer/AV_consumer_Wdays_hours_Summer15V2.npy')


                    season2_3rdmonth = np.load('C:/Users/aguir/AnacondaProjects/Summer/AV_consumer_Wdays_15min_3rdmonth_Summer15V1.npy')

            if WINTER ==1:
                if CONST_SAMPLE_SIZE == 96:
                    season1 = np.load('C:/Users/aguir/AnacondaProjects/Winter/AV_consumer_Wdays_15min_Winter15V1.npy')
                    season2 = np.load('C:/Users/aguir/AnacondaProjects/Fall/AV_consumer_Wdays_15min_Fall15V1.npy')


                    season2_3rdmonth = np.load('C:/Users/aguir/AnacondaProjects/Fall/AV_consumer_Wdays_15min_3rdmonth_FAll15V1.npy')
                else:
                    season1 = np.load('C:/Users/aguir/AnacondaProjects/Winter/AV_consumer_Wdays_hours_Winter15V1.npy')
                    season2 = np.load('C:/Users/aguir/AnacondaProjects/Fall/AV_consumer_Wdays_hours_Fall15V1.npy')


                    season2_3rdmonth = np.load('C:/Users/aguir/AnacondaProjects/Fall/AV_consumer_Wdays_15min_3rdmonth_Fall15V1.npy')


        #1D vector with the iDs of each season
        season1_ID = season1[:,0,2]
        season2_ID = season2[:,0,2]
        season2_3rdmonth_ID = season2_3rdmonth[:,0,2]


        #Computamos un vector con los clientes comunes a ambos periodos
        #Clients from both periods
        Common_consumer = np.intersect1d(season1_ID,season2_ID)
        Common_consumer_1 = np.intersect1d(season2_3rdmonth_ID,Common_consumer)


        #Te da un array de True y False, donde las posiciones true son elementos comunes
        #It gives you a boolean array. True position are common elements while FALSE are not common elements.
        Common_consumer_season1_bool = np.in1d(season1_ID,Common_consumer)
        Common_consumer_season2_bool = np.in1d(season2_ID,Common_consumer)
        Common_consumer_season2_3rdmonth_bool = np.in1d(season2_3rdmonth_ID,Common_consumer_1)


        #Eliminamos los consumidores no comunes a ambos periodos
        #We eliminate not common consumers

        season1 = season1[Common_consumer_season1_bool]
```

```
204.        season2 = season2[Common_consumer_season2_bool]
205.
206.
207.        season2_3rdmonth = season2_3rdmonth[Common_consumer_season2_3rdmonth_bo
    ol]
208.
209.        #Generamos un vector con los power use de los clientes de cada estación

210.        #1D array with the power use
211.        season1_use =  np.vstack(season1[:,:,1]).astype(float)
212.        season2_use =  np.vstack(season2[:,:,1]).astype(float)
213.        season2_3rdmonth_use = np.vstack(season2_3rdmonth[:,:,1]).astype(float)

214.
215.
216.        #Z-Normalization
217.        if Z_NORM == 1 :
218.            standardscaler = StandardScaler(epsilon=1e-2)
219.            season1_use = standardscaler.transform(season1_use)
220.            season2_use = standardscaler.transform(season2_use)
221.            season2_3rdmonth_use = standardscaler.transform(season2_3rdmonth_us
    e)
222.
223.        #Peak Normalization
224.        else:
225.            for i in range (0, len(season1_use)):
226.                p_peak = np.amax(season1_use[i,:])
227.                a = season1_use[i,:]/p_peak
228.                season1_use[i,:] = a
229.
230.                p_peak = np.amax(season2_use[i,:])
231.                a = season2_use[i,:]/p_peak
232.                season2_use[i,:] = a
233.
234.
235.
236.        #Some plots
237.        plt.figure(f)
238.        f+=1
239.        plot_standardscaler(season1_use[5])
240.        plt.figure(f)
241.        f+=1
242.        plot_mtf(season1_use[0], image_size=CONST_SAMPLE_SIZE, n_bins=N_BINS, q
    uantiles='empirical', overlapping=False)
243.
244.
245.
246.        #MARKOV TRANSITION FIELD (includes blurring kernel):
247.        mtf = MTF(image_size=CONST_SAMPLE_SIZE, n_bins=N_BINS, quantiles='empir
    ical', overlapping=False) #quantile can be empirical or gaussian, empirical ma
    kes sure same number of elements in each bin
248.        X_mtf_1 = mtf.transform(season1_use)
249.        X_mtf_2 = mtf.transform(season2_use)
250.        X_mtf_3 = mtf.transform(season2_3rdmonth_use)
251.
252.        ##Uncoment if you want to save the MTF
253.        #np.save('MTF96_Q=16_XXXX.npy',X_mtf_3)
254.
255.
256.        #GRAMIAN ANGULAR FIELDS (incluyen PAA):
257.        gasf = GASF(image_size=CONST_SAMPLE_SIZE, overlapping=False, scale='0')
     #Scale = 0 to normalize between 0 and 1. Scale = -1 to normalize between -
    1 and 1
258.        X_gasf_1 = gasf.transform(season1_use)
259.        X_gasf_2 = gasf.transform(season2_use)
260.        X_gasf_3 = gasf.transform(season2_3rdmonth_use)
```

```
261.        #np.save('GASF96_XXXX',X_gasf_3)
262.
263.        plt.figure(f)
264.        f+=1
265.        plot_gasf(season1_use[0], image_size=CONST_SAMPLE_SIZE, overlapping=Fal
    se, scale='0')
266.
267.
268.        gadf = GADF(image_size=CONST_SAMPLE_SIZE, overlapping=False, scale='0')

269.        X_gadf_1 = gadf.transform(season1_use)
270.        X_gadf_2 = gadf.transform(season2_use)
271.        X_gadf_3 = gadf.transform(season2_3rdmonth_use)
272.        #np.save('GADF96_XXXX',X_gadf_3)
273.
274.        plt.figure(f)
275.        f+=1
276.        plot_gadf(season1_use[0], image_size=CONST_SAMPLE_SIZE, overlapping=Fal
    se, scale='0')
```

DemandForecasting.py

DemandForecasting.py

```
#Libraries
import numpy as np
import matplotlib.pyplot as plt
get_ipython().run_line_magic('matplotlib', 'inline')

import keras
from keras.models import Model
from keras.layers import LocallyConnected2D
from keras.layers import Input, Conv2D, MaxPooling2D, Flatten, Dense, Dropout

from keras.utils import to_categorical




# Select 1 per run:
SUMMER_FORECASTING_496 = 1
SUMMER_FORECASTING_358 = 0
WINTER_FORECASTING = 0
SPRING_FORECASTING = 0
FALL_FORECASTING = 0
SPRING_CLASSIFICATION = 0

# Select 1 for Convolutional or Tiled Convolutional:

CONV = 1
TILED = 0


n_classes=7


# ## Load and process the data and the targets




# load the channels

#SUMMER FORECASTING: 496 consumers (California+Texas+Colorado)
if SUMMER_FORECASTING_496 == 1:

    mtf_3 = np.load("./Data12/SummerForecasting/POSTPAPER/MTF96_SPRING_Q=16_3rdMonth.npy")
    gadf_3 = np.load("./Data12/SummerForecasting/POSTPAPER/GADF96_SPRING_3rdmonth.npy")
    gasf_3 = np.load("./Data12/SummerForecasting/POSTPAPER/GASF96_SPRING_3rdmonth.npy")

    targets = to_categorical(np.load("./Data12/SummerForecasting/POSTPAPER/labels_Summer_postpaper_N=7.npy"), num_classes=n_classes)


#SUMMER FORECASTING: 358 consumers (Only Austin)
elif SUMMER_FORECASTING_358 == 1:
    mtf_3 = np.load("./Data12/SummerForecasting/96/MTF_Summer_Q=15_3.npy")
    gadf_3 = np.load("./Data12/SummerForecasting/96/GADF_96_3.npy")
    gasf_3 = np.load("./Data12/SummerForecasting/96/GASF_96_3.npy")

```

```
    targets = to_categorical(np.load("./Data12/SummerForecasting/96/Labels_SummerForecasting_N=7_96.npy"), num_classes=n_classes)


#WINTER FORECASTING
elif WINTER_FORECASTING == 1:
    mtf_3 = np.load("./Data12/WinterForecasting/MTF_Fall_Q=4_3.npy")
    gadf_3 = np.load("./Data12/WinterForecasting/GADF_Fall_3.npy")
    gasf_3 = np.load("./Data12/WinterForecasting/GASF_Fall_3.npy")

    targets = to_categorical(np.load("./Data12/WinterForecasting/Labels_WinterForecasting_N=7_96.npy"), num_classes=n_classes)

#SPRING FORECASTING
elif SPRING_FORECASTING == 1:
    mtf_3 = np.load("./Data12/SpringForecasting/MTF_Winter_Q=4_3.npy")
    gadf_3 = np.load("./Data12/SpringForecasting/GADF_Winter_3.npy")
    gasf_3 = np.load("./Data12/SpringForecasting/GASF_Winter_3.npy")

    targets = to_categorical(np.load("./Data12/SpringForecasting/Labels_SpringForecasting_N=7_96.npy"), num_classes=n_classes)

#FALL FORECASTING
elif FALL_FORECASTING == 1:
    mtf_3 = np.load("./Data12/FallForecasting/MTF_Summer_Q=4_3.npy")
    gadf_3 = np.load("./Data12/FallForecasting/GADF_Summer_3.npy")
    gasf_3 = np.load("./Data12/FallForecasting/GASF_Summer_3.npy")

    targets = to_categorical(np.load("./Data12/FallForecasting/Labels_FallForecasting_N=7_96.npy"), num_classes=n_classes)

#SPRING CLASSIFICATION
elif SPRING_CLASSIFICATION == 1:
    mtf_3 = np.load("./Data12/SpringClassification/96/MTF96_Q=5_S15.npy")
    gadf_3 = np.load("./Data12/SpringClassification/96/GADF96_S15.npy")
    gasf_3 = np.load("./Data12/SpringClassification/96/GASF96_S15.npy")

    targets = to_categorical(np.load("./Data12/SpringClassification/96/labels_SpringClasification_N=7_96.npy"), num_classes=n_classes)









# Append the channels
# empty list
m = []

# append along 3rd axis
        for i in range(gadf_3.shape[0]):
            #m.append([ gadf_1[i], gasf_1[i], gadf_2[i], gasf_2[i],gadf_3[i], gasf_3[i]])
            m.append([mtf_3[i], gadf_3[i], gasf_3[i]])
            #m.append([gadf_1[i], gadf_2[i], gadf_3[i]])

        # define the data array
        data = np.array(m)
        #data = gasf_3




        data.shape

```

```
115.
116.
117.        # check an image
118.        plot = plt.imshow(data[0][0], cmap='gray')
119.
120.
121.
122.
123.        # global parameters
124.        _SHAPE = data.shape
125.        _KERNEL_SIZE = (3, 3, 3)
126.        _FC_NODES = (128, 512, n_classes)
127.        _FILTERS = (16, 64, 256)
128.        #_FILTERS = (64, 128, 512)
129.
130.
131.        # ## Build the model
132.
133.        # **Fully Connected CNN**
134.
135.
136.
137.        _SHAPE
138.
139.
140.
141.
142.
143.        def setup_fully_connected() -> Model:
144.            '''''
145.                Builds the model architecture:
146.                    1) Fully connected CNN - 3 Convolutional layers, 3 fully co
    nnected layers
147.                    2) FC CNN - 3 convolutioonal layers and 3 fully connected l
    ayers
148.
149.                :return:
150.                    keras functional model
151.            '''
152.
153.            # build the fully connected model
154.
155.
156.            # input layer
157.            input_layer = Input(shape=(_SHAPE[1], _SHAPE[2], _SHAPE[2]), name="
    Input")
158.            #input_layer = Input(shape=(1, 96, 96), name="Input")
159.
160.            # first convolution
161.            input_layer_conv = Conv2D(filters=_FILTERS[0], kernel_size=(_KERNEL
    _SIZE[0],_KERNEL_SIZE[0]),
162.                                      strides=(1,1), padding="same", data_forma
    t="channels_first",
163.                                      name="input_conv", activation="relu")(inp
    ut_layer)
164.            # max pool
165.            input_max = MaxPooling2D(pool_size=(2,2), strides=None, padding="sa
    me", data_format="channels_first",
166.                                              name="input_max")(input_layer_conv)

167.
168.
169.            # second convolution
170.            hidden_conv_1 = Conv2D(filters=_FILTERS[1], kernel_size=(_KERNEL_SI
    ZE[1],_KERNEL_SIZE[1]),
```

```
                                    strides=(1,1), padding="same", data_format="channels_first",
                                    name="hidden_conv_1", activation="relu")(input_max)


            hidden_max_1 = MaxPooling2D(pool_size=(2,2), strides=None, padding="same", data_format="channels_first",
                                        name="hidden_max_1")(hidden_conv_1)


            # Third convolution
            hidden_conv_2 = Conv2D(filters=_FILTERS[2], kernel_size=(_KERNEL_SIZE[2],_KERNEL_SIZE[2]),
                                    strides=(1,1), padding="same", data_format="channels_first",
                                    name="hidden_conv_2", activation="relu")(hidden_max_1)


            hidden_max_2 = MaxPooling2D(pool_size=(2,2), strides=None, padding="same", data_format="channels_first",
                                        name="hidden_max_2")(hidden_conv_2)


            # flatten the image
            fc_1 = Flatten()(hidden_max_2)

            fc_2 = Dense(_FC_NODES[0], activation="relu")(fc_1)

            fc_3 = Dense(_FC_NODES[1], activation="relu")(fc_2)

            fc_4 = Dense(_FC_NODES[2], activation="softmax")(fc_3)

            model = Model(inputs=input_layer, outputs=fc_4)


            return Model(inputs=input_layer, outputs=fc_4)


        # **Locally Connected CNN**




        def setup_locally_connected() -> Model:
            '''''
                Builds the model architecture:
                    1) Fully connected CNN - 3 Convolutional layers, 3 fully connected layers
                    2) Tiled CNN - 3 convolutioonal layers and 3 locally connected layers

                :return:
                    keras functional model
            '''

            # input layer
            input_layer = Input(shape=(_SHAPE[1], _SHAPE[2], _SHAPE[2]), name="Input")
            #input_layer = Input(shape=(1, 96, 96), name="Input")
            # first convolution
            input_layer_conv = LocallyConnected2D(filters=_FILTERS[0], kernel_size=(_KERNEL_SIZE[0],_KERNEL_SIZE[0]),
```

```
                                        strides=(1,1), padding="valid", data_format="channels_first",
                                        name="input_conv", activation="relu")(input_layer)

            # max pool
            input_max = MaxPooling2D(pool_size=(2,2), strides=None, padding="valid", data_format="channels_first",
                                                name="input_max")(input_layer_conv)


            # second convolution
            hidden_conv_1 = LocallyConnected2D(filters=_FILTERS[1], kernel_size=(_KERNEL_SIZE[1],_KERNEL_SIZE[1]),
                                        strides=(1,1), padding="valid", data_format="channels_first",
                                        name="hidden_conv_1", activation="relu")(input_max)

            hidden_max_1 = MaxPooling2D(pool_size=(2,2), strides=None, padding="valid", data_format="channels_first",
                                            name="hidden_max_1")(hidden_conv_1)


            # second convolution
            hidden_conv_2 = LocallyConnected2D(filters=_FILTERS[2], kernel_size=(_KERNEL_SIZE[2],_KERNEL_SIZE[2]),
                                        strides=(1,1), padding="valid", data_format="channels_first",
                                        name="hidden_conv_2", activation="relu")(hidden_max_1)

            hidden_max_2 = MaxPooling2D(pool_size=(2,2), strides=None, padding="valid", data_format="channels_first",
                                            name="hidden_max_2")(hidden_conv_2)


            # flatten the image
            fc_1 = Flatten()(hidden_max_2)

            fc_2 = Dense(_FC_NODES[0], activation="relu")(fc_1)

            fc_3 = Dense(_FC_NODES[1], activation="relu")(fc_2)

            fc_4 = Dense(_FC_NODES[2], activation="softmax")(fc_3)

            return Model(inputs=input_layer, outputs=fc_4)


        # ### Prepare The Data




        def prepare_data(data, targets, test_split):
            '''''
                Splits the data into training and test sets; also, reshapes the image data for CNN input - 1 channel

                :param data:
                    data in the shape of (i, nX, nY, nCh)

                :param test_split:
                    desired train-test split ratio

```

```
279.                :return:
280.                    training set, test set, training targets and test targets
281.            '''
282.
283.            # split the dataset into training and testing sets
284.            # shuffle the data
285.            array = np.arange(data.shape[0])  # get the indices of the set
286.            np.random.shuffle(array)  # shuffle the indices
287.            shuffled_data, shuffled_targets = data[array], targets[array]  # sh
    uffle the data
288.
289.            # define the separation boundary
290.            bound = int(np.floor(data.shape[0] * test_split))
291.
292.            # separate the data
293.            test_set, training_set = shuffled_data[:bound,:], shuffled_data[bou
    nd:,:]
294.            test_targets, training_targets = shuffled_targets[:bound,:], shuffl
    ed_targets[bound:,:]
295.
296.            # reshape the data to fit the input to the CNN
297.            # channels-last
298.            training_set = training_set.reshape((training_set.shape[0], test_se
    t.shape[1], test_set.shape[2], test_set.shape[3]))
299.            test_set = test_set.reshape((test_set.shape[0], test_set.shape[1],
    test_set.shape[2], test_set.shape[3]))
300.            #training_set = training_set.reshape((training_set.shape[0], 1, tes
    t_set.shape[1], test_set.shape[2]))
301.            #test_set = test_set.reshape((test_set.shape[0], 1, test_set.shape[
    1], test_set.shape[2]))
302.            return training_set, test_set, training_targets, test_targets
303.
304.
305.        # ### Define The Training Function
306.
307.
308.
309.
310.        def train(data, batch_size=128, epochs = 25, val_split = 0.20, test_spl
    it = 0.20):
311.            '''''
312.                Compiles and trains the model, prints out the train and test er
    rors
313.
314.                :param data:
315.                    data in the shape of (i, nX, nY, nCh)
316.
317.                :param epochs:
318.                    number of epochs for training
319.
320.                :param val_split:
321.                    validation split ratio
322.
323.                :param test_split:
324.                    train - test split ratio
325.            '''
326.
327.            # define the aoptimizer and the loss function
328.            # TODO: Tune Adam, architecture
329.
330.            optimizer = keras.optimizers.adam()
331.            loss = "categorical_crossentropy"
332.
333.            # define the early stopping callback
334.            early_stopping = keras.callbacks.EarlyStopping(monitor="val_categor
    ical_accuracy",
```

```
                                                    patience=2, min_del
    ta=0.01, verbose=1)

            # get the model

            if CONV == 1:
                model = setup_fully_connected()
            elif TILED == 1:
                model = setup_locally_connected()

            print(model.summary())

            # compile the model
            model.compile(loss=loss, optimizer=optimizer, metrics=["categorical
    _accuracy"])

            # print the model summary
            # print(model.summary())

            # get the training, test sets and targets
            training_set, test_set, training_targets, test_targets = prepare_da
    ta(data, targets, test_split)

            # fit the model
            model.fit(x=training_set, batch_size=batch_size, y=training_targets
    , epochs=epochs, verbose=1,
                      callbacks=[early_stopping], validation_split=val_split)

            # print the metrics
            test_loss, test_accuracy = model.evaluate(x=test_set, y=test_target
    s, verbose=0)
            training_loss, training_accuracy = model.evaluate(x=training_set, y
    =training_targets, verbose=0)
            print("Test Loss: {loss:.5f}, Test Accuracy: {accuracy:.5f}".format
    (loss=test_loss, accuracy=test_accuracy))
            print("Train Loss: {loss:.5f}, Train Accuracy: {accuracy:.5f}".form
    at(loss=training_loss, accuracy=training_accuracy))

            # return the trained model
            return model

            # write the model to a file
            #model.save("./models/TESTACC-{:.4f}.h5".format(test_accuracy))


        # ## Train the Model

        # **Check The Devices**



        from tensorflow.python.client import device_lib
        print(device_lib.list_local_devices())


        # **Train**



        # train the model and return it to the global scope
        model = train(data, batch_size=128, val_split = 0.20, test_split = 0.20
    )

```

```
392.
393.
394.        y_prob = model.predict(x=data)
395.        print(y_prob)
396.
397.
398.
399.
400.
401.        y_classes = y_prob.argmax(axis=-1)
402.        print(y_classes)
403.
404.
405.
406.
407.
408.        y_prob[0]
409.
410.
411.
412.
413.
414.        #np.save("./Data12/predicted_labels.npy",y_classes)
```

SpringSummer_Kmeans_Classification.py

SpringSummer_Kmeans_Classification_ImagesV2.py

```
"""
Created on Wed Apr 18 16:38:05 2018
@author: Ignacio Aguirre

Description:
This script has the objective of providing the accuracy of kmeans algorithm in
 classification
and forecasting tasks. The scrip is divided in 2 parts. The first part is the
classification section
where we use the first month of a season to classify the consumer for the whol
e season.
We compute this clasification using 3 different methods:
    1. Using raw time series of the first month
    2. Using the GAFs images of the first month
    3. Using the GAFs+MTF images of the first month

In the second part we use the last month of the season (3rd month) to make a p
rediction
of the next season. We compute this forecasting using 3 different methods:
    1. Using raw time series of the third month
    2. Using the GAFs images of the third month
    3. Using the GAFs+MTF images of the third month

"""



#Libraries
import numpy as np
from sklearn.cluster import KMeans
from pyts.transformation import StandardScaler
from pyts.transformation import MTF
from pyts.transformation import GASF, GADF



#Constants
CONST_SAMPLE_SIZE = 96
N_CLUSTERS = 7
N_bins_MTF = 16


#Activation variables
Z_NORM = 1 #Hacemos la Z normalizacion o por pico
AUSTIN = 0

#Variable para las figuras;
f=0

#Select 1 season. If you select season we will be predicting summer using spri
ng. If you select winter, we will be predicting winter using fall.
WINTER = 0
SPRING = 0
SUMMER = 1
FALL = 0


#Load the data. Season 1 is always the one to be predicted whereas season2 is
the one we use to predict.

```

```
58. if AUSTIN == 1:
59.
60.     if SUMMER ==1:
61.         if CONST_SAMPLE_SIZE == 96:
62.
63.             season1 = np.load('C:/Users/aguir/AnacondaProjects/Summer/AV_Austi
    n_consumer_Wdays_15min_Summer15V2.npy')
64.             season2 = np.load('C:/Users/aguir/AnacondaProjects/Spring/AV_Austi
    n_consumer_Wdays_15min_Spring15V1.npy')
65.
66.             season2_1stmonth = np.load('C:/Users/aguir/AnacondaProjects/Spring
    /AV_Austin_consumer_Wdays_15min_1stmonth_Spring15V1.npy')
67.             season2_3rdmonth = np.load('C:/Users/aguir/AnacondaProjects/Spring
    /AV_Austin_consumer_Wdays_15min_3rdmonth_Spring15V1.npy')
68.         else:
69.
70.             season1 = np.load('C:/Users/aguir/AnacondaProjects/Summer/AV_Austi
    n_consumer_Wdays_hours_Summer15V2.npy')
71.             season2 = np.load('C:/Users/aguir/AnacondaProjects/Spring/AV_Austi
    n_consumer_Wdays_hours_Spring15V1.npy')
72.
73.             season2_1stmonth = np.load('C:/Users/aguir/AnacondaProjects/Spring
    /AV_Austin_consumer_Wdays_15min_1stmonth_Spring15V1.npy')
74.             season2_3rdmonth = np.load('C:/Users/aguir/AnacondaProjects/Spring
    /AV_Austin_consumer_Wdays_15min_3rdmonth_Spring15V1.npy')
75.
76.     if SPRING ==1:
77.         if CONST_SAMPLE_SIZE == 96:
78.             season1 = np.load('C:/Users/aguir/AnacondaProjects/Spring/AV_Austi
    n_consumer_Wdays_15min_Spring15V1.npy')
79.             season2 = np.load('C:/Users/aguir/AnacondaProjects/Winter/AV_Austi
    n_consumer_Wdays_15min_Winter15V2.npy')
80.
81.             season2_1stmonth = np.load('C:/Users/aguir/AnacondaProjects/Winter
    /AV_Austin_consumer_Wdays_15min_1stmonth_Winter15V1.npy')
82.             season2_3rdmonth = np.load('C:/Users/aguir/AnacondaProjects/Winter
    /AV_Austin_consumer_Wdays_15min_3rdmonth_Winter15V1.npy')
83.         else:
84.             season1 = np.load('C:/Users/aguir/AnacondaProjects/Spring/AV_Austi
    n_consumer_Wdays_hours_Spring15V1.npy')
85.             season2 = np.load('C:/Users/aguir/AnacondaProjects/Winter/AV_Austi
    n_consumer_Wdays_hours_Winter15V2.npy')
86.
87.             season2_1stmonth = np.load('C:/Users/aguir/AnacondaProjects/Winter
    /AV_Austin_consumer_Wdays_hours_1stmonth_Winter15V1.npy')
88.             season2_3rdmonth = np.load('C:/Users/aguir/AnacondaProjects/Winter
    /AV_Austin_consumer_Wdays_hours_3rdmonth_Winter15V1.npy')
89.
90.     if FALL ==1:
91.         if CONST_SAMPLE_SIZE == 96:
92.             season1 = np.load('C:/Users/aguir/AnacondaProjects/Fall/AV_Austin_
    consumer_Wdays_15min_Fall15V1.npy')
93.             season2 = np.load('C:/Users/aguir/AnacondaProjects/Summer/AV_Austi
    n_consumer_Wdays_15min_Summer15V2.npy')
94.
95.             season2_1stmonth = np.load('C:/Users/aguir/AnacondaProjects/Summer
    /AV_Austin_consumer_Wdays_15min_1stmonth_Summer15V1.npy')
96.             season2_3rdmonth = np.load('C:/Users/aguir/AnacondaProjects/Summer
    /AV_Austin_consumer_Wdays_15min_3rdmonth_Summer15V1.npy')
97.         else:
98.             season1 = np.load('C:/Users/aguir/AnacondaProjects/Fall/AV_Austin_
    consumer_Wdays_hours_Fall15V1.npy')
99.             season2 = np.load('C:/Users/aguir/AnacondaProjects/Summer/AV_Austi
    n_consumer_Wdays_hours_Summer15V2.npy')
100.
```

```
                    season2_1stmonth = np.load('C:/Users/aguir/AnacondaProjects/Summer/AV_Austin_consumer_Wdays_hours_1stmonth_Summer15V1.npy')
                    season2_3rdmonth = np.load('C:/Users/aguir/AnacondaProjects/Summer/AV_Austin_consumer_Wdays_hours_3rdmonth_Summer15V1.npy')
            if WINTER ==1:
                if CONST_SAMPLE_SIZE == 96:
                    season1 = np.load('C:/Users/aguir/AnacondaProjects/Winter/AV_Austin_consumer_Wdays_15min_Winter15V2.npy')
                    season2 = np.load('C:/Users/aguir/AnacondaProjects/Fall/AV_Austin_consumer_Wdays_15min_Fall15V1.npy')

                    season2_1stmonth = np.load('C:/Users/aguir/AnacondaProjects/Fall/AV_Austin_consumer_Wdays_15min_1stmonth_Fall15V1.npy')
                    season2_3rdmonth = np.load('C:/Users/aguir/AnacondaProjects/Fall/AV_Austin_consumer_Wdays_15min_3rdmonth_FAll15V1.npy')
                else:
                    season1 = np.load('C:/Users/aguir/AnacondaProjects/Winter/AV_Austin_consumer_Wdays_hours_Winter15V2.npy')
                    season2 = np.load('C:/Users/aguir/AnacondaProjects/Fall/AV_Austin_consumer_Wdays_hours_Fall15V1.npy')
                    season2_1stmonth = np.load('C:/Users/aguir/AnacondaProjects/Fall/AV_Austin_consumer_Wdays_hours_1stmonth_Fall15V1.npy')
                    season2_3rdmonth = np.load('C:/Users/aguir/AnacondaProjects/Fall/AV_Austin_consumer_Wdays_hours_3rdmonth_FAll15V1.npy')

        else:

            if SUMMER == 1:
                if CONST_SAMPLE_SIZE == 96:
                    season1 = np.load('C:/Users/aguir/AnacondaProjects/Summer/AV_consumer_Wdays_15min_Summer15V2.npy')
                    season2 = np.load('C:/Users/aguir/AnacondaProjects/Spring/AV_consumer_Wdays_15min_Spring15V1.npy')

                    season2_1stmonth = np.load('C:/Users/aguir/AnacondaProjects/Spring/AV_consumer_Wdays_15min_1stmonth_Spring15V1.npy')
                    season2_3rdmonth = np.load('C:/Users/aguir/AnacondaProjects/Spring/AV_consumer_Wdays_15min_3rdmonth_Spring15V1.npy')
                else:
                    season1 = np.load('C:/Users/aguir/AnacondaProjects/Summer/AV_consumer_Wdays_hours_Summer15V2.npy')
                    season2 = np.load('C:/Users/aguir/AnacondaProjects/Spring/AV_consumer_Wdays_hours_Spring15V1.npy')

                    season2_1stmonth = np.load('C:/Users/aguir/AnacondaProjects/Spring/AV_consumer_Wdays_hours_1stmonth_Spring15V1.npy')
                    season2_3rdmonth = np.load('C:/Users/aguir/AnacondaProjects/Spring/AV_consumer_Wdays_hours_3rdmonth_Spring15V1.npy')

            if SPRING ==1:
                if CONST_SAMPLE_SIZE == 96:
                    season1 = np.load('C:/Users/aguir/AnacondaProjects/Spring/AV_consumer_Wdays_15min_Spring15V1.npy')
                    season2 = np.load('C:/Users/aguir/AnacondaProjects/Winter/AV_consumer_Wdays_15min_Winter15V1.npy')

                    season2_1stmonth = np.load('C:/Users/aguir/AnacondaProjects/Winter/AV_consumer_Wdays_15min_1stmonth_Winter15V1.npy')
                    season2_3rdmonth = np.load('C:/Users/aguir/AnacondaProjects/Winter/AV_consumer_Wdays_15min_3rdmonth_Winter15V1.npy')
                else:
                    season1 = np.load('C:/Users/aguir/AnacondaProjects/Spring/AV_consumer_Wdays_hours_Spring15V1.npy')
                    season2 = np.load('C:/Users/aguir/AnacondaProjects/Winter/AV_consumer_Wdays_hours_Winter15V1.npy')

```

```
                    season2_1stmonth = np.load('C:/Users/aguir/AnacondaProjects/Winter/AV_consumer_Wdays_hours_1stmonth_Winter15V1.npy')
                    season2_3rdmonth = np.load('C:/Users/aguir/AnacondaProjects/Winter/AV_consumer_Wdays_hours_3rdmonth_Winter15V1.npy')


            if FALL ==1:
                if CONST_SAMPLE_SIZE == 96:
                    season1 = np.load('C:/Users/aguir/AnacondaProjects/Fall/AV_consumer_Wdays_15min_Fall15V1.npy')
                    season2 = np.load('C:/Users/aguir/AnacondaProjects/Summer/AV_consumer_Wdays_15min_Summer15V2.npy')

                    season2_1stmonth = np.load('C:/Users/aguir/AnacondaProjects/Summer/AV_consumer_Wdays_15min_1stmonth_Summer15V1.npy')
                    season2_3rdmonth = np.load('C:/Users/aguir/AnacondaProjects/Summer/AV_consumer_Wdays_15min_3rdmonth_Summer15V1.npy')
                else:
                    season1 = np.load('C:/Users/aguir/AnacondaProjects/Fall/AV_consumer_Wdays_hours_Fall15V1.npy')
                    season2 = np.load('C:/Users/aguir/AnacondaProjects/Summer/AV_consumer_Wdays_hours_Summer15V2.npy')

                    season2_1stmonth = np.load('C:/Users/aguir/AnacondaProjects/Summer/AV_consumer_Wdays_15min_1stmonth_Summer15V1.npy')
                    season2_3rdmonth = np.load('C:/Users/aguir/AnacondaProjects/Summer/AV_consumer_Wdays_15min_3rdmonth_Summer15V1.npy')

            if WINTER ==1:
                if CONST_SAMPLE_SIZE == 96:
                    season1 = np.load('C:/Users/aguir/AnacondaProjects/Winter/AV_consumer_Wdays_15min_Winter15V1.npy')
                    season2 = np.load('C:/Users/aguir/AnacondaProjects/Fall/AV_consumer_Wdays_15min_Fall15V1.npy')

                    season2_1stmonth = np.load('C:/Users/aguir/AnacondaProjects/Fall/AV_consumer_Wdays_15min_1stmonth_Fall15V1.npy')
                    season2_3rdmonth = np.load('C:/Users/aguir/AnacondaProjects/Fall/AV_consumer_Wdays_15min_3rdmonth_FAll15V1.npy')
                else:
                    season1 = np.load('C:/Users/aguir/AnacondaProjects/Winter/AV_consumer_Wdays_hours_Winter15V1.npy')
                    season2 = np.load('C:/Users/aguir/AnacondaProjects/Fall/AV_consumer_Wdays_hours_Fall15V1.npy')

                    season2_1stmonth = np.load('C:/Users/aguir/AnacondaProjects/Fall/AV_consumer_Wdays_15min_1stmonth_Fall15V1.npy')
                    season2_3rdmonth = np.load('C:/Users/aguir/AnacondaProjects/Fall/AV_consumer_Wdays_15min_3rdmonth_Fall15V1.npy')



        #1D vector with the IDs
        #Generamos un vector con los dataID de los clientes de cada estación
        season1_ID = season1[:,0,2]
        season2_ID = season2[:,0,2]


        #1D vector with the IDs
        #Generamos un vector con los dataID de los clientes de cada estación
        season2_ID_1month = season2_1stmonth[:,0,2]
        season2_ID_3month = season2_3rdmonth[:,0,2] #el tercer mes es el que tiene menos elementos y por tanto más restrictivo


```

```
        #It gives us a vector with common consumer for both periods
        #Computamos un vector con los clientes comunes a ambos periodos
        Common_consumer = np.intersect1d(season1_ID,season2_ID)

        #It gives us a vector with common consumer for both periods
        #Computamos un vector con los clientes comunes a ambos periodos
        Common_consumer_season2_1stmonth = np.intersect1d(season2_ID_1month,Com
    mon_consumer)
        Common_consumer_season2_3rdmonth = np.intersect1d(season2_ID_3month,Com
    mon_consumer)


        #It gives a boolean array where the TRUE position are the common elemen
    ts for both periods
        #Te da un array de True y False, donde las posiciones true son elemento
    s comunes
        Common_consumer_season1_bool = np.in1d(season1_ID,Common_consumer)
        Common_consumer_season2_bool = np.in1d(season2_ID,Common_consumer)

        #It gives a boolean array where the TRUE position are the common elemen
    ts for both periods
        #Te da un array de True y False, donde las posiciones true son elemento
    s comunes
        Common_consumer_1month_season2_bool = np.in1d(season2_ID_1month,Common_
    consumer)
        Common_consumer_3month_season2_bool = np.in1d(season2_ID_3month,Common_
    consumer)


        #Here we eliminate the elements that are not common for both periods
        season1 = season1[Common_consumer_season1_bool]
        season2 = season2[Common_consumer_season2_bool]

        season2_1stmonth = season2_1stmonth[Common_consumer_1month_season2_bool
    ]
        season2_3rdmonth = season2_3rdmonth[Common_consumer_3month_season2_bool
    ]


        #Generamos un vector con los power use de los clientes de cada estación

        season1_use =  np.vstack(season1[:,:,1]).astype(float)
        season2_use =  np.vstack(season2[:,:,1]).astype(float)

        #Generamos un vector con los power use de los clientes de cada estación

        season2_1stmonth_use =  np.vstack(season2_1stmonth[:,:,1]).astype(float
    )
        season2_3rdmonth_use =  np.vstack(season2_3rdmonth[:,:,1]).astype(float
    )


        if Z_NORM == 1 :

            standardscaler = StandardScaler(epsilon=1e-2)
            season1_use = standardscaler.transform(season1_use)
            season2_use = standardscaler.transform(season2_use)

            season2_1stmonth_use = standardscaler.transform(season2_1stmonth_us
    e)
            season2_3rdmonth_use = standardscaler.transform(season2_3rdmonth_us
    e)


        #Classification after 1month gathering info. Here we are not forecastin
    g yet.
```

```
239.
240.       #Here we create the images for the season 2, it will be used to classif
    y using images
241.       mtf = MTF(image_size= CONST_SAMPLE_SIZE, n_bins=N_bins_MTF, quantiles='
    empirical', overlapping=False)
242.       X_mtf_0 = mtf.transform(season2_use)
243.
244.       gasf = GASF(image_size= CONST_SAMPLE_SIZE, overlapping=False, scale='0'
    )
245.       X_gasf_0 = gasf.transform(season2_use)
246.
247.       gadf = GADF(image_size= CONST_SAMPLE_SIZE, overlapping=False, scale='0'
    )
248.       X_gadf_0 = gadf.transform(season2_use)
249.
250.       #Construct the three channels
251.
252.       n_channels = 3
253.
254.       #Size of the vector of the three images in a row
255.       size = int(CONST_SAMPLE_SIZE*CONST_SAMPLE_SIZE*n_channels)
256.
257.
258.       # Append the channels
259.       # To use images for classification we need to consider them as a 1D vec
    tor where all the pixeles are appended to 1D vector
260.       m_0 = np.array([], dtype=float).reshape(0,size)
261.
262.
263.       for i in range(X_mtf_0.shape[0]):
264.
265.           fila = np.array([], dtype=float).reshape(0,size)
266.           fila = np.append(fila, [X_mtf_0[i,:,:].flatten(),X_gadf_0[i,:,:].fl
    atten(),X_gasf_0[i,:,:].flatten()])
267.           m_0 = np.r_[m_0, [fila]]
268.
269.
270.
271.
272.       #First month images
273.       mtf = MTF(image_size= CONST_SAMPLE_SIZE, n_bins=N_bins_MTF, quantiles='
    empirical', overlapping=False)
274.       X_mtf_1 = mtf.transform(season2_1stmonth_use)
275.
276.       gasf = GASF(image_size= CONST_SAMPLE_SIZE, overlapping=False, scale='0'
    )
277.       X_gasf_1 = gasf.transform(season2_1stmonth_use)
278.
279.       gadf = GADF(image_size= CONST_SAMPLE_SIZE, overlapping=False, scale='0'
    )
280.       X_gadf_1 = gadf.transform(season2_1stmonth_use)
281.
282.       #Construct the three channels
283.       # Append the channels
284.       m_1 = np.array([], dtype=float).reshape(0,size)
285.
286.       for i in range(X_mtf_1.shape[0]):
287.
288.           fila = np.array([], dtype=float).reshape(0,size)
289.           fila = np.append(fila, [X_mtf_1[i,:,:].flatten(),X_gadf_1[i,:,:].fl
    atten(),X_gasf_1[i,:,:].flatten()])
290.           m_1 = np.r_[m_1, [fila]]
291.
292.       #Kmeans for images
293.       kmeans = KMeans(n_clusters=N_CLUSTERS, random_state=0).fit(m_0)
294.       labels_spring_Images = kmeans.labels_
```

```
        labels_spring_classified_Images = kmeans.predict(m_1)

        #Kmeans for Time Series
        kmeans = KMeans(n_clusters=N_CLUSTERS, random_state=0).fit(season2_use)

        labels_spring_TimeSeries = kmeans.labels_
        labels_spring_classified_TimeSeries = kmeans.predict(season2_1stmonth_use)


        #K-means Accuracy
        Common_labels_Images = np.sum(labels_spring_Images == labels_spring_classified_Images)
        Common_labels_TimeSeries = np.sum(labels_spring_TimeSeries == labels_spring_classified_TimeSeries)

        ACC_Spring_Images = Common_labels_Images/len(labels_spring_Images)
        ACC_Spring_TimeSeries = Common_labels_TimeSeries/len(labels_spring_TimeSeries)

        print("Accuracy Classification Time Series: %.4f" % (ACC_Spring_TimeSeries))
        print("Accuracy Classification images MTF+GAFs: %.4f" % (ACC_Spring_Images))


        #Probemos clustering con imagenes sin usar la MTF
        n_channels = 2

        #Size of the vector of the three images in a row
        size = int(CONST_SAMPLE_SIZE*CONST_SAMPLE_SIZE*n_channels)

        m_2 = np.array([], dtype=float).reshape(0,size)


        # append along 3rd axis
        for i in range(X_gadf_0.shape[0]):

            fila = np.array([], dtype=float).reshape(0,size)
            fila = np.append(fila, [X_gadf_0[i,:,:].flatten(),X_gasf_0[i,:,:].flatten()])
            m_2 = np.r_[m_2, [fila]]


        m_3 = np.array([], dtype=float).reshape(0,size)
        # append along 3rd axis
        for i in range(X_gadf_1.shape[0]):

            fila = np.array([], dtype=float).reshape(0,size)
            fila = np.append(fila, [X_gadf_1[i,:,:].flatten(),X_gasf_1[i,:,:].flatten()])
            m_3 = np.r_[m_3, [fila]]

        kmeans = KMeans(n_clusters=N_CLUSTERS, random_state=0).fit(m_2)
        labels_spring_Images_GAF = kmeans.labels_
        labels_spring_classified_Images_GAF = kmeans.predict(m_3)

        Common_labels_Images_GAF = np.sum(labels_spring_Images_GAF == labels_spring_classified_Images_GAF)
        ACC_Spring_Images_GAF = Common_labels_Images_GAF/len(labels_spring_Images_GAF)
        print("Accuracy Classification images GAFs: %.4f" % (ACC_Spring_Images_GAF))

        #Forecastin using last month of data:
```

```
349.
350.        #With time series
351.        kmeans = KMeans(n_clusters=N_CLUSTERS, random_state=0).fit(season1_use)

352.        labels_summer_TimeSeries = kmeans.labels_
353.        labels_spring_classified_TimeSeries_Summer = kmeans.predict(season2_3rd
    month_use)
354.
355.        ##Uncomment if you want to save the K-means predictions
356.        #np.save('labels_predicted_Kmeans.npy',labels_spring_classified_TimeSer
    ies_Summer)
357.        Common_labels_TimeSeries_Summer = np.sum(labels_summer_TimeSeries == la
    bels_spring_classified_TimeSeries_Summer)
358.        ACC_Summer_TimeSeries = Common_labels_TimeSeries_Summer/len(labels_summ
    er_TimeSeries)
359.        print("Accuracy Forecasting Time series: %.4f" % (ACC_Summer_TimeSeries
    ))
360.
361.        #With Images:
362.
363.        #Obtain de images
364.        #Spring Images
365.        mtf = MTF(image_size= CONST_SAMPLE_SIZE, n_bins=N_bins_MTF, quantiles='
    empirical', overlapping=False)
366.        X_mtf_0 = mtf.transform(season1_use)
367.
368.        gasf = GASF(image_size= CONST_SAMPLE_SIZE, overlapping=False, scale='0'
    )
369.        X_gasf_0 = gasf.transform(season1_use)
370.
371.        gadf = GADF(image_size= CONST_SAMPLE_SIZE, overlapping=False, scale='0'
    )
372.        X_gadf_0 = gadf.transform(season1_use)
373.
374.        #Construct the three channels
375.        n_channels = 3
376.
377.        #Size of the vector of the three images in a row
378.        size = int(CONST_SAMPLE_SIZE*CONST_SAMPLE_SIZE*n_channels)
379.
380.
381.        # Append the channels
382.        # empty list
383.        m_4 = np.array([], dtype=float).reshape(0,size)
384.
385.
386.        for i in range(X_mtf_0.shape[0]):
387.
388.            fila = np.array([], dtype=float).reshape(0,size)
389.            fila = np.append(fila, [X_mtf_0[i,:,:].flatten(),X_gadf_0[i,:,:].fl
    atten(),X_gasf_0[i,:,:].flatten()])
390.            m_4 = np.r_[m_4, [fila]]
391.
392.
393.
394.
395.        #First month images
396.        mtf = MTF(image_size= CONST_SAMPLE_SIZE, n_bins=N_bins_MTF, quantiles='
    empirical', overlapping=False)
397.        X_mtf_3 = mtf.transform(season2_3rdmonth_use)
398.
399.        gasf = GASF(image_size= CONST_SAMPLE_SIZE, overlapping=False, scale='0'
    )
400.        X_gasf_3 = gasf.transform(season2_3rdmonth_use)
401.
```

```
        gadf = GADF(image_size= CONST_SAMPLE_SIZE, overlapping=False, scale='0'
    )
        X_gadf_3 = gadf.transform(season2_3rdmonth_use)


        # Append the channels
        # empty list
        m_5 = np.array([], dtype=float).reshape(0,size)

        for i in range(X_mtf_3.shape[0]):

            fila = np.array([], dtype=float).reshape(0,size)
            fila = np.append(fila, [X_mtf_3[i,:,:].flatten(),X_gadf_3[i,:,:].fl
    atten(),X_gasf_3[i,:,:].flatten()])
            m_5 = np.r_[m_5, [fila]]


        kmeans = KMeans(n_clusters=N_CLUSTERS, random_state=0).fit(m_4)
        labels_Summer_Images = kmeans.labels_
        labels_Summer_classified_Images = kmeans.predict(m_5)

        Common_labels_Images = np.sum(labels_Summer_Images == labels_Summer_cla
    ssified_Images)
        ACC_Summer_Images = Common_labels_Images/len(labels_Summer_Images)
        print("Accuracy Forecasting images MTF+GAFs: %.4f" % (ACC_Summer_Images
    ))

        #WITHOUT MTF
        #Probemos clustering con imagenes sin usar la MTF
        n_channels = 2

        #Size of the vector of the three images in a row
        size = int(CONST_SAMPLE_SIZE*CONST_SAMPLE_SIZE*n_channels)

        m_6 = np.array([], dtype=float).reshape(0,size)
        m_7 = np.array([], dtype=float).reshape(0,size)


        # append along 3rd axis
        for i in range(X_gadf_0.shape[0]):

            fila = np.array([], dtype=float).reshape(0,size)
            fila = np.append(fila, [X_gadf_0[i,:,:].flatten(),X_gasf_0[i,:,:].f
    latten()])
            m_7 = np.r_[m_7, [fila]]


        for i in range(X_gadf_3.shape[0]):

            fila = np.array([], dtype=float).reshape(0,size)
            fila = np.append(fila, [X_gadf_3[i,:,:].flatten(),X_gasf_3[i,:,:].f
    latten()])
            m_6 = np.r_[m_6, [fila]]

        kmeans = KMeans(n_clusters=N_CLUSTERS, random_state=0).fit(m_7)
        labels_Summer_Images_GAF = kmeans.labels_
        labels_Summer_classified_Images_GAF = kmeans.predict(m_6)

        Common_labels_Images_GAF = np.sum(labels_Summer_Images_GAF == labels_Su
    mmer_classified_Images_GAF)
        ACC_Summer_Images_GAF = Common_labels_Images_GAF/len(labels_Summer_Imag
    es_GAF)

        print("Accuracy Forecasting GAFs: %.4f" % (ACC_Summer_Images_GAF))
```

UncertainReductionV3.py

UncertainReductionV3.py

```
"""
Created on Tue May 15 22:49:34 2018

@author: Ignacio Aguirre Panadero

Description:
In this script we compare the forecasting algorithms with the real values.
The predicition labels of both algorithm must be an input. To reconstruct the

predictions we can use data (average and std.deviation) from the previous
month to the season that we pretend to predict or we can use the real values o
f
the predicted month (this is a bit deceptive because we are using info of the
season
that we are predicting). This two options are change with the variable FAKE.
"""



#Libraries
import numpy as np
import matplotlib.pyplot as plt
from pyts.transformation import StandardScaler
from sklearn.cluster import KMeans


#Global constants
N_CLUSTERS = 7
CONST_SAMPLE_SIZE = 96 #24 for 1h cadence, 96 for 15 min cadence

#Activation variables
Z_NORM = 1 #Z_NORM=1 for Z-Normalization, Z_NORM=0 for peak Normalization
AUSTIN = 0

FAKE = 0 #IF 1 we use the real values of the average and std deviation to reco
nstruct the load profile. If 0 we use the data form the previous month
#Constant for figures
f=0

#Factor for energy calculations
FACTOR = (24/CONST_SAMPLE_SIZE)

#Select which season want to be predicted. Select only one per run. For exampl
e if we select summer, we will use the last month of spring to predict summer.

WINTER = 0
SPRING = 0
SUMMER = 1
FALL = 0



#Load the data. Season 1 is always the one to be predicted whereas season2 is
the one we use to predict.
if AUSTIN == 1:

    if SUMMER ==1:
        if CONST_SAMPLE_SIZE == 96:

```

```
            season1 = np.load('C:/Users/aguir/AnacondaProjects/Summer/AV_Austin_consumer_Wdays_15min_Summer15V2.npy')
            season2 = np.load('C:/Users/aguir/AnacondaProjects/Spring/AV_Austin_consumer_Wdays_15min_Spring15V1.npy')
        else:

            season1 = np.load('C:/Users/aguir/AnacondaProjects/Summer/AV_Austin_consumer_Wdays_hours_Summer15V2.npy')
            season2 = np.load('C:/Users/aguir/AnacondaProjects/Spring/AV_Austin_consumer_Wdays_hours_Spring15V1.npy')

    if SPRING ==1:
        if CONST_SAMPLE_SIZE == 96:
            season1 = np.load('C:/Users/aguir/AnacondaProjects/Spring/AV_Austin_consumer_Wdays_15min_Spring15V1.npy')
            season2 = np.load('C:/Users/aguir/AnacondaProjects/Winter/AV_Austin_consumer_Wdays_15min_Winter15V2.npy')

        else:
            season1 = np.load('C:/Users/aguir/AnacondaProjects/Spring/AV_Austin_consumer_Wdays_hours_Spring15V1.npy')
            season2 = np.load('C:/Users/aguir/AnacondaProjects/Winter/AV_Austin_consumer_Wdays_hours_Winter15V2.npy')

    if FALL ==1:
        if CONST_SAMPLE_SIZE == 96:
            season1 = np.load('C:/Users/aguir/AnacondaProjects/Fall/AV_Austin_consumer_Wdays_15min_Fall15V1.npy')
            season2 = np.load('C:/Users/aguir/AnacondaProjects/Summer/AV_Austin_consumer_Wdays_15min_Summer15V2.npy')
        else:
            season1 = np.load('C:/Users/aguir/AnacondaProjects/Fall/AV_Austin_consumer_Wdays_hours_Fall15V1.npy')
            season2 = np.load('C:/Users/aguir/AnacondaProjects/Summer/AV_Austin_consumer_Wdays_hours_Summer15V2.npy')

    if WINTER ==1:
        if CONST_SAMPLE_SIZE == 96:
            season1 = np.load('C:/Users/aguir/AnacondaProjects/Winter/AV_Austin_consumer_Wdays_15min_Winter15V2.npy')
            season2 = np.load('C:/Users/aguir/AnacondaProjects/Fall/AV_Austin_consumer_Wdays_15min_Fall15V1.npy')
        else:
            season1 = np.load('C:/Users/aguir/AnacondaProjects/Winter/AV_Austin_consumer_Wdays_hours_Winter15V2.npy')
            season2 = np.load('C:/Users/aguir/AnacondaProjects/Fall/AV_Austin_consumer_Wdays_hours_Fall15V1.npy')

else:

    if SUMMER == 1:
        if CONST_SAMPLE_SIZE == 96:
            season1 = np.load('C:/Users/aguir/AnacondaProjects/Summer/AV_consumer_Wdays_15min_Summer15V2.npy')
            season2 = np.load('C:/Users/aguir/AnacondaProjects/Spring/AV_consumer_Wdays_15min_Spring15V1.npy')
            if FAKE == 0:
                season2_3rdmonth = np.load('C:/Users/aguir/AnacondaProjects/Spring/AV_consumer_Wdays_15min_3rdmonth_Spring15V1.npy')
        else:
            season1 = np.load('C:/Users/aguir/AnacondaProjects/Summer/AV_consumer_Wdays_hours_Summer15V2.npy')
            season2 = np.load('C:/Users/aguir/AnacondaProjects/Spring/AV_consumer_Wdays_hours_Spring15V1.npy')
                if FAKE == 0:
```

```
                        season2_3rdmonth = np.load('C:/Users/aguir/AnacondaProjects/Spring/AV_consumer_Wdays_hours_3rdmonth_Spring15V1.npy')

            if SPRING ==1:
                if CONST_SAMPLE_SIZE == 96:
                    season1 = np.load('C:/Users/aguir/AnacondaProjects/Spring/AV_consumer_Wdays_15min_Spring15V1.npy')
                    season2 = np.load('C:/Users/aguir/AnacondaProjects/Winter/AV_consumer_Wdays_15min_Winter15V1.npy')
                    if FAKE == 0:
                        season2_3rdmonth = np.load('C:/Users/aguir/AnacondaProjects/Winter/AV_consumer_Wdays_15min_3rdmonth_Winter15V1.npy')
                else:
                    season1 = np.load('C:/Users/aguir/AnacondaProjects/Spring/AV_consumer_Wdays_hours_Spring15V1.npy')
                    season2 = np.load('C:/Users/aguir/AnacondaProjects/Winter/AV_consumer_Wdays_hours_Winter15V1.npy')
                    if FAKE == 0:
                        season2_3rdmonth = np.load('C:/Users/aguir/AnacondaProjects/Winter/AV_consumer_Wdays_hours_3rdmonth_Winter15V1.npy')


            if FALL ==1:
                if CONST_SAMPLE_SIZE == 96:
                    season1 = np.load('C:/Users/aguir/AnacondaProjects/Fall/AV_consumer_Wdays_15min_Fall15V1.npy')
                    season2 = np.load('C:/Users/aguir/AnacondaProjects/Summer/AV_consumer_Wdays_15min_Summer15V2.npy')
                    if FAKE == 0:
                        season2_3rdmonth = np.load('C:/Users/aguir/AnacondaProjects/Summer/AV_consumer_Wdays_15min_3rdmonth_Summer15V1.npy')
                else:
                    season1 = np.load('C:/Users/aguir/AnacondaProjects/Fall/AV_consumer_Wdays_hours_Fall15V1.npy')
                    season2 = np.load('C:/Users/aguir/AnacondaProjects/Summer/AV_consumer_Wdays_hours_Summer15V2.npy')
                    if FAKE == 0:
                        season2_3rdmonth = np.load('C:/Users/aguir/AnacondaProjects/Summer/AV_consumer_Wdays_hours_3rdmonth_Summer15V1.npy')

            if WINTER ==1:
                if CONST_SAMPLE_SIZE == 96:
                    season1 = np.load('C:/Users/aguir/AnacondaProjects/Winter/AV_consumer_Wdays_15min_Winter15V1.npy')
                    season2 = np.load('C:/Users/aguir/AnacondaProjects/Fall/AV_consumer_Wdays_15min_Fall15V1.npy')
                    if FAKE == 0:
                        season2_3rdmonth = np.load('C:/Users/aguir/AnacondaProjects/Fall/AV_consumer_Wdays_15min_3rdmonth_Fall15V1.npy')
                else:
                    season1 = np.load('C:/Users/aguir/AnacondaProjects/Winter/AV_consumer_Wdays_hours_Winter15V1.npy')
                    season2 = np.load('C:/Users/aguir/AnacondaProjects/Fall/AV_consumer_Wdays_hours_Fall15V1.npy')
                    if FAKE == 0:
                        season2_3rdmonth = np.load('C:/Users/aguir/AnacondaProjects/Fall/AV_consumer_Wdays_hours_3rdmonth_Fall15V1.npy')



        #1D array with the peak power of each consumer
        average_1D = []
        std_deviation_1D = []

```

```
148.
149.
150.        #1D vector with the IDs
151.        season1_ID = season1[:,0,2]
152.        season2_ID = season2[:,0,2]
153.
154.        #It gives us a vector with common consumer for both periods
155.        Common_consumer = np.intersect1d(season1_ID,season2_ID)
156.
157.        #Te da un array de True y False, donde las posiciones true son elemento
    s comunes
158.        #It gives a boolean array where the TRUE position are the common elemen
    ts for both periods
159.        Common_consumer_season1_bool = np.in1d(season1_ID,Common_consumer)
160.        Common_consumer_season2_bool = np.in1d(season2_ID,Common_consumer)
161.
162.        #Here we eliminate the elements that are not common for both periods
163.        season1 = season1[Common_consumer_season1_bool]
164.        season2 = season2[Common_consumer_season2_bool]
165.
166.        #1D vector with the power use of every period
167.        season1_use =  np.vstack(season1[:,:,1]).astype(float)
168.        season2_use =  np.vstack(season2[:,:,1]).astype(float)
169.
170.        #Z-Normalization
171.        if Z_NORM == 1 :
172.
173.            standardscaler = StandardScaler(epsilon=1e-2)
174.            season1_use_Znorm = standardscaler.transform(season1_use)
175.            season2_use_Znorm = standardscaler.transform(season2_use)
176.
177.        #If FAKE = 1, we use the actual values of the forecast season
178.        if FAKE == 1:
179.
180.            for i in range (0, len(season1_use)):
181.
182.                p_average = np.average(season1_use[i,:],axis = 0)
183.                average_1D.append(p_average)
184.                p_std = np.std(season1_use[i,:],axis = 0)
185.                std_deviation_1D.append(p_std)
186.
187.        elif FAKE == 0:
188.            season2_3rdmonth_use = season2_3rdmonth[:,:,1].astype(float)
189.            Common_consumer_3rdmonth_bool = np.in1d(season2_3rdmonth[:,0,2],Com
    mon_consumer)
190.            season2_3rdmonth_use = season2_3rdmonth_use[Common_consumer_3rdmont
    h_bool]
191.
192.            for i in range (0, len(season2_3rdmonth_use)):
193.
194.                    p_average = np.average(season2_3rdmonth_use[i,:],axis = 0)

195.                    average_1D.append(p_average)
196.                    p_std = np.std(season2_3rdmonth_use[i,:],axis = 0)
197.                    std_deviation_1D.append(p_std)
198.        else:
199.            print('FAKE HAS TO BE 0 OR 1')
200.
201.
202.
203.
204.        #Computing the clustering
205.        kmeans = KMeans(n_clusters=N_CLUSTERS, random_state=0).fit(season1_use_
    Znorm)
206.        labels_season1 = kmeans.labels_
207.        #np.save('labels_Spring_postpaper_N=9.npy',labels_spring )
```

```
        centers_season1 = kmeans.cluster_centers_
        kmeans = KMeans(n_clusters=N_CLUSTERS, random_state=0).fit(season2_use_Znorm)
        labels_season2 = kmeans.labels_

        labels_real = labels_season1
        centers_season2 = kmeans.cluster_centers_



        average_1D_np = np.array(average_1D)
        std_dev_1D_np = np.array(std_deviation_1D)

        #Variables to store the total erergy predicted by each algorithm for a representative day
        real_energy_day = np.array([], dtype = float)
        CNNs_energy_day = np.array([], dtype = float)
        Kmeans_energy_day = np.array([], dtype = float)

        #Variables to store the power at every interval. We will use this variables to compare the instant power predicted with the instant real values.
        real_power_15min =  np.array([], dtype = float)
        CNNs_power_15min =  np.array([], dtype = float)
        Kmeans_power_15min =  np.array([], dtype = float)

        #Variables to store the difference in percentage of the power use between the predictions and the real values at every interval.
        CNNs_percentage_instant_power_np = np.array([], dtype = float)
        K_means_percentage_instant_power_np = np.array([], dtype = float)

        #Load the the predictions data
        if SUMMER == 1:
            labels_CNNs = np.load('labels_predicted_Summer_CNNs.npy')
            labels_Kmeans = np.load('labels_predicted_Summer_Kmeans.npy')



        #Here we compare the accuracy in the forecasting task of both algorithms:
        Common_labels_CNNS = np.sum(labels_CNNs  == labels_real)
        ACC_season1_CNNs_classification = (Common_labels_CNNS)/len(labels_real)

        Common_labels_Kmeans = np.sum(labels_Kmeans  == labels_real)
        ACC_season1_kmeans_classification = (Common_labels_Kmeans)/len(labels_real)


        #Arrays to compare values along 24-
        96 dimensions. it contains the reconstruct predictions and the real representative daily profiles
        M_real = np.array([], dtype= float).reshape(0,CONST_SAMPLE_SIZE)
        M_CNN = np.array([], dtype= float).reshape(0,CONST_SAMPLE_SIZE)
        M_kmeans = np.array([], dtype= float).reshape(0,CONST_SAMPLE_SIZE)

        #H is a vector from 0 to 95 to plot profiles
        H = np.arange(0,96)

        #j is used to construct a 2x2 subplot
        j=0
        for i in range (0,len(labels_CNNs)):



            power_1 = season1_use[i]
            M_real = np.r_[M_real, [power_1]]
```

```
            day_energy_real = FACTOR * np.sum(power_1)
            real_energy_day = np.append(real_energy_day, day_energy_real)
            real_power_15min =   np.append(real_power_15min, power_1)


            power_2 = average_1D_np[i]+(centers_season1[labels_CNNs[i]]*std_dev_1D_np[i])
            M_CNN = np.r_[M_CNN, [power_2]]
            percentage_instant_power = (abs(power_1-power_2))/abs(power_1)
            CNNs_percentage_instant_power_np = np.append(CNNs_percentage_instant_power_np,percentage_instant_power)
            day_energy_CNN = FACTOR * np.sum(power_2)
            CNNs_energy_day = np.append(CNNs_energy_day, day_energy_CNN)
            CNNs_power_15min =   np.append(CNNs_power_15min, power_2)


            power_3 = average_1D_np[i]+(centers_season1[labels_Kmeans[i]]*std_dev_1D_np[i])
            M_kmeans = np.r_[M_kmeans, [power_3]]
            percentage_instant_power = (abs(power_1-power_3))/abs(power_1)
            Kmeans_percentage_instant_power_np = np.append(K_means_percentage_instant_power_np,percentage_instant_power)
            day_energy_Kmeans = FACTOR * np.sum(power_3)
            Kmeans_energy_day = np.append(Kmeans_energy_day, day_energy_Kmeans)

            Kmeans_power_15min =   np.append(Kmeans_power_15min, power_3)


            #We just draw the last 4 consumers predictions
            if i >=  (len(labels_CNNs)-4):

                j+=1

                if j == 1:
                    plt.figure(f)
                    f+=1
                    plt.subplot(2, 2, j)
                    plt.plot(H,power_1, label="Real power")
                    plt.plot(H,power_2, label="CNNs prediction")
                    plt.plot(H,power_3, label="Kmeans prediction")
                    plt.ylabel('Power (kW)', fontsize=12)

                elif j == 2:
                    plt.subplot(2, 2, j)
                    plt.plot(H,power_1, label="Real power")
                    plt.plot(H,power_2, label="CNNs prediction")
                    plt.plot(H,power_3, label="Kmeans prediction")

                elif j == 3:
                    plt.subplot(2, 2, j)
                    plt.plot(H,power_1, label="Real power")
                    plt.plot(H,power_2, label="CNNs prediction")
                    plt.plot(H,power_3, label="Kmeans prediction")
                    plt.ylabel('Power (kW)', fontsize=12)
                    plt.xlabel('Time', fontsize=12)

                else:
                   plt.subplot(2, 2, j)
                   plt.plot(H,power_1, label="Real power")
                   plt.plot(H,power_2, label="CNNs prediction")
                   plt.plot(H,power_3, label="Kmeans prediction")
                   plt.xlabel('Time', fontsize=12)

```

```
326.
327.       plt.legend(bbox_to_anchor=(-1.3,2.3), loc=3,
328.          ncol=3, borderaxespad=0.)
329.
330.       plt.show()
331.
332.
333.       #We are adding all the power use of all consumer along the same dimensi
    on.
334.       #To compare what is the total power at hour 0, 1 ...n
335.       M_sum_1 = np.sum(M_real, axis = 0)
336.       M_sum_2 = np.sum(M_CNN, axis=0)
337.       M_sum_3 = np.sum(M_kmeans, axis=0)
338.
339.       #We compute the difference at each dimension for the CCNs model. All co
    nsumer are added along dimensions.
340.       M_diff_CNN = (abs(M_sum_1-M_sum_2)/M_sum_1)
341.       ACC_average_CNN_dimension = np.average(M_diff_CNN)*100
342.       print('Average difference along 96 dimensions CNNs model: %f.4' % (ACC_
    average_CNN_dimension))
343.
344.       #We compute the difference at each dimension for the Kmeans algorithm
345.       M_diff_kmeans = (abs(M_sum_1-M_sum_3)/M_sum_1)
346.       ACC_average_kmeans_dimension = np.average(M_diff_kmeans)*100
347.       print('Average difference along 96 dimensions Kmeans model: %f.4' % (AC
    C_average_kmeans_dimension))
348.
349.       #These are the average error in following the real representative profi
    le for each consumer. All consumer are added along dimensions.
350.       Average_CNNs = np.average(CNNs_percentage_instant_power_np)
351.       print('Average difference instant power CNNs model: %f.4' % (Average_CN
    Ns))
352.       Average_Kmeans= np.average(Kmeans_percentage_instant_power_np)
353.       print('Average difference instant power Kmeans model: %f.4' % (Average_
    Kmeans))
```

FunctionsV2.py

FunctionsV2.py

```
"""
Created on Fri May  4 16:18:15 2018
@author: Ignacio Aguirre Panadero

DESCRIPTION:
Script with the functions
"""


import datetime
import matplotlib.pyplot as plt
import numpy as np

def day_of_week(consumer):

    consumer_Wdays = []
    consumer_Wends = []
    j=0
    k=0

    #Separación del consumidor entre Workdays y Weekends

    for i in range(0,len(consumer)-1): #-
1 porque cuando seleccionamos la query incluye un dato que no queremos

        if datetime.datetime.weekday(consumer[i][0]) < 5: #comprobamos que sea
 menor de 5 para weekdays Monday=0...

            consumer_Wdays.insert(j,consumer[i])
            j+= 1

        else:
           consumer_Wends.insert(k,consumer[i])
           k+= 1


    return consumer_Wdays, consumer_Wends

def average_profile(consumer, sample_size):

    AvConsumer = []  #creamos las listas de los valores medios representativos


    for h in range(0,sample_size): #h is the sameple size, 1h or 15 min
        suma = 0
        average = 0

        for l in range (0,int((len(consumer)/sample_size))): #it calculates th
e numbers of days of the input
            suma = suma + consumer[(l*sample_size)+h][1]


        average = suma/int((len(consumer)/sample_size))
        AvConsumer.insert(h,[consumer[h][0],average,consumer[h][2]])

    return AvConsumer

def daily_profile(consumer, sample_size):

    consumer_Wdays = []
    consumer_Wends = []
    AvConsumer_Wdays = []
    AvConsumer_Wends = []
```

```
    j=0
    k=0

    #Separación del consumidor entre Workdays y Weekends

    for i in range(0,len(consumer)-1): #-
1 porque cuando seleccionamos la query incluye un dato que no queremos

        if datetime.datetime.weekday(consumer[i][0]) < 5: #comprobamos que sea
 menor de 5 para weekdays Monday=0...

            consumer_Wdays.insert(j,consumer[i])
            j+= 1

        else:
           consumer_Wends.insert(k,consumer[i])
           k+= 1

#Cálculo de los valores medios
    for h in range(0,sample_size): #h is the sameple size, 1h (24) or 15 (96)
min
        suma = 0
        average = 0

        for l in range (0,int((len(consumer_Wdays)/sample_size))): #it calcula
tes the numbers of days of the input
            suma = suma + consumer_Wdays[(l*sample_size)+h][1]


        average = suma/int((len(consumer_Wdays)/sample_size))
        AvConsumer_Wdays.insert(h,[consumer_Wdays[h][0],average,consumer_Wdays
[h][2]])
        suma = 0
        average = 0

        for m in range (0,int((len(consumer_Wends)/sample_size))): #it calcula
tes the numbers of days of the input

            suma = suma + consumer_Wends[(m*sample_size)+h][1]


        average = suma/int((len(consumer_Wends)/sample_size))
        AvConsumer_Wends.insert(h,[consumer_Wends[h][0],average,consumer_Wends
[h][2]])


    return AvConsumer_Wdays, AvConsumer_Wends


      def daily_profile_V2(consumers, data_interval, timelag, CONST_SAMPLE_SI
ZE, WEEKDAYS, WEEKENDS):

           weekdays_bool = np.is_busday(consumers[:,0].astype('datetime64'))

           weekeends_bool = np.logical_not(weekdays_bool)

           clean_consumers_weekdays = consumers[weekdays_bool]
           clean_consumers_weekends = consumers[weekeends_bool]

           n_consumers = int(len(consumers)/data_interval)
           n_weekdays = int(len(clean_consumers_weekdays)/(n_consumers*CONST_S
AMPLE_SIZE))


           #The 4hour are due to saving daylight lag
```

```
116.           n_weekends = int(((len(clean_consumers_weekends)/n_consumers)-
    timelag)/CONST_SAMPLE_SIZE)
117.
118.
119.
120.           data_interval_weekdays = int (n_weekdays*CONST_SAMPLE_SIZE)
121.           data_interval_weekends = int (n_weekends*CONST_SAMPLE_SIZE)
122.
123.           M = []
124.           dates = consumers[0:int(CONST_SAMPLE_SIZE),0]
125.
126.           if WEEKDAYS == 1:
127.
128.               j=0
129.               for i in range (0,n_consumers):
130.                   j+=1
131.
132.                   start_1_weekdays = int(i*data_interval_weekdays)
133.                   end_1_weekdays = int (j*data_interval_weekdays)
134.
135.                   Weekdays_use = clean_consumers_weekdays[start_1_weekdays:en
    d_1_weekdays,1].astype(float)
136.
137.                   start_ID = int(i*data_interval)
138.                   end_ID = CONST_SAMPLE_SIZE+start_ID
139.                   consumer_ID = consumers[start_ID:end_ID,2]
140.
141.                   M_Weekdays = np.array([], dtype=float).reshape(0,CONST_SAMP
    LE_SIZE)
142.
143.                   u = 0
144.                   for z in range (0,n_weekdays):
145.                       u+=1
146.                       start_2 = int(z*CONST_SAMPLE_SIZE)
147.                       end_2 = int(u*CONST_SAMPLE_SIZE)
148.
149.                       day_use_weekday = Weekdays_use[start_2:end_2]
150.
151.
152.                       M_Weekdays = np.r_[M_Weekdays,[day_use_weekday]]
153.
154.
155.                   Average_weekdays = np.average(M_Weekdays,axis=0)
156.
157.
158.                   M_representative_Weekdays = np.array([], dtype=float).resha
    pe(CONST_SAMPLE_SIZE,0)
159.                   M_representative_Weekdays = np.c_[M_representative_Weekdays
    ,dates,Average_weekdays,consumer_ID]
160.
161.                   M.append(M_representative_Weekdays)
162.
163.               #M_np = np.array(M)
164.
165.           if WEEKENDS == 1:
166.               j=0
167.               for i in range (0,n_consumers):
168.                   j+=1
169.
170.                   start_1_weekends = int(i*data_interval_weekends)
171.                   end_1_weekends = int (j*data_interval_weekends)
172.
173.                   Weekends_use = clean_consumers_weekdays[start_1_weekends:en
    d_1_weekends,1].astype(float)
174.                   start_ID = int(i*data_interval)
175.                   end_ID = CONST_SAMPLE_SIZE+start_ID
```

```
                    consumer_ID = consumers[start_ID:end_ID,2]

                    M_Weekends = np.array([], dtype=float).reshape(0,CONST_SAMPLE_SIZE)

                    u = 0
                    for z in range (0,n_weekends):
                        u+=1
                        start_2 = int(z*CONST_SAMPLE_SIZE)
                        end_2 = int(u*CONST_SAMPLE_SIZE)
                        day_use_weekend = Weekends_use[start_2:end_2]

                        M_Weekends = np.r_[M_Weekends,[day_use_weekend]]

                    Average_weekends = np.average(M_Weekends,axis=0)

                    M_representative_Weekends = np.array([], dtype=float).reshape(CONST_SAMPLE_SIZE,0)
                    M_representative_Weekends = np.c_[M_representative_Weekends,dates,Average_weekends,consumer_ID]

                    M.append(M_representative_Weekends)

                #M_np = np.array(M)


            return M


        def weakly_profile(consumer, sample_size):

            sample_size = 7*sample_size

            AvConsumer_Wdays = []



        #Cálculo de los valores medios
            for h in range(0,sample_size): #h is the sameple size, 1h (24) or 15 (96) min
                suma = 0
                average = 0

                for l in range (0,int((len(consumer)/(sample_size)))): #it calculates the numbers of days of the input
                    suma = suma + consumer[(l*sample_size)+h][1]


                average = suma/int((len(consumer)/sample_size))
                AvConsumer_Wdays.insert(h,[consumer[h][0],average,consumer[h][2]])
                suma = 0
                average = 0



            return AvConsumer_Wdays



        def norma_profile(consumer, sample_size):

            consumer_norma = []
```

```
236.
237.           col = 1
238.           colList = [consumer[i][col] for i in range(len(consumer))]
239.           peak = max(colList)
240.
241.           for j in range (0,len(consumer)):
242.
243.               consumer_norma.insert(j,[consumer[j][0],consumer[j][1]/peak,con
    sumer[j][2]])
244.
245.           return consumer_norma
246.
247.
248.       #Markov Transition Matrix
249.       def MTM(consumer,sample_size, Q):
250.
251.           CONST_Q = Q #number of quantile bins
252.
253.           #Function to find the peak power of a TimeSerie
254.           a = 1
255.           colList = [consumer[i][a] for i in range(len(consumer)) ]
256.           q = max(colList)/CONST_Q #each quantile has the same size q
257.           quantile = [] #vector with the quantile of each point
258.           W = [] #Markov Transition Matrix
259.
260.
261.
262.           #Fill the vector quantile
263.           for i in range (0,len(consumer)):
264.               if consumer[i][1] == 1:
265.                   quantile.insert(i,CONST_Q-1)
266.               else:
267.                   for j in range (0,CONST_Q):
268.
269.                       if j*q <= consumer[i][1] < (j+1)*q:
270.
271.                           quantile.insert(i,j)
272.
273.           #Algorithm to fill W
274.           for row in range (0, CONST_Q):
275.
276.               fila = []
277.               for col in range (0, CONST_Q):
278.
279.                   suma = 0 #reiniciamos el contador de elementos por quantile

280.                   suma2 = 0
281.                   for p in range (0, len(quantile)-
    1): #the last data point does not jump anywhere, there is no next point (reaso
    n -1)
282.                       if quantile[p] == row:
283.                           suma = suma + 1
284.                           if row == col: #diagonal self-probability
285.                               if quantile[p] == quantile [p+1]:
286.                                   suma2 = suma2 + 1
287.                           else:
288.                               if quantile[p+1] == col:
289.                                   suma2 = suma2 + 1
290.                   if suma == 0:
291.                       fila.append(0)
292.                   else:
293.                       fila.append(suma2/suma)
294.               W.append(fila)
295.           return W
296.
297.       def MTF(consumer,sample_size, Q):
```

```
298.
299.           CONST_Q = Q #number of quantile bins
300.
301.           #Function to find the peak power of a TimeSerie
302.           a = 1
303.           colList = [consumer[i][a] for i in range(len(consumer)) ]
304.           q = max(colList)/CONST_Q #each quantile has the same size q
305.           quantile = [] #vector with the quantile of each point
306.           W = [] #Markov Transition Matrix
307.           M = [] #Markov Transition Field
308.
309.
310.           #Fill the vector quantile
311.           for i in range (0,len(consumer)):
312.               if consumer[i][1] == 1:
313.                   quantile.insert(i,CONST_Q-1)
314.               else:
315.                   for j in range (0,CONST_Q):
316.
317.                       if j*q <= consumer[i][1] < (j+1)*q:
318.
319.                           quantile.insert(i,j)
320.
321.           #Algorithm to fill W
322.           for row in range (0, CONST_Q):
323.
324.               fila = []
325.               for col in range (0, CONST_Q):
326.
327.                   suma = 0 #reiniciamos el contador de elementos por quantile

328.                   suma2 = 0
329.                   for p in range (0, len(quantile)-
    1): #the last data point does not jump anywhere, there is no next point (reaso
    n -1)
330.                       if quantile[p] == row:
331.                           suma = suma + 1
332.                           if row == col: #diagonal self-probability
333.                               if quantile[p] == quantile [p+1]:
334.                                   suma2 = suma2 + 1
335.                           else:
336.                               if quantile[p+1] == col:
337.                                   suma2 = suma2 + 1
338.                   if suma == 0:
339.                       fila.append(0)
340.                   else:
341.                       fila.append(suma2/suma)
342.               W.append(fila)
343.
344.           row = 0
345.           col = 0
346.       #Algorithm to fill M
347.           for row in range(0, sample_size):
348.               fila = []
349.               q1 = quantile[row]
350.               for col in range(0, sample_size):
351.                   q2 = quantile[col]
352.                   fila.append(W[q1][q2])
353.
354.               M.append(fila)
355.
356.
357.
358.           return M
359.
360.       def M_use(AV_consumer_Wdays_np_array, CONST_SAMPLE_SIZE):
```

```
361.
362.             Consumer_profiles = []
363.
364.             for i in range(len(AV_consumer_Wdays_np_array)):
365.                 fila = []
366.                 for j in range (CONST_SAMPLE_SIZE):
367.                     fila.append(float(AV_consumer_Wdays_np_array[i][j][1]))
368.                 Consumer_profiles.append(fila)
369.
370.             return Consumer_profiles
371.
372.
373.
374.         def plot(consumer, sample_size):
375.
376.             X = []
377.             Y = []
378.
379.             for n in range(0,sample_size):
380.
381.                 X.insert(n,consumer[n][0])
382.                 Y.insert(n,consumer[n][1])
383.
384.                 plt.plot(X,Y, marker = 'o' , linestyle='-', color='b',)
385.
386.             return
387.
388.         #Clustering functions
389.
390.         def MIA(labels, centroids, consumers_use, N_clusters, CONST_SAMPLE_SIZE
    ):
391.
392.             M_clusters = []
393.             for i in range(0,N_clusters):
394.                 cluster = []
395.                 fila = []
396.
397.                 for j in range (0, len(labels)):
398.                     if labels[j] == i:
399.                         for k in range(0,CONST_SAMPLE_SIZE):
400.                             fila.append(consumers_use[j][k])
401.                         cluster.append(fila)
402.                         fila = []
403.
404.
405.                 cluster_np = np.array(cluster)
406.                 M_clusters.append(cluster_np)
407.
408.
409.             #Distancia de cada profile con su centroide
410.             M_average_clusters = np.array([])
411.             MIA_per_cluster = np.array([])
412.             for i in range(0,N_clusters):
413.                 M_average_class = np.array([])
414.                 centroid_1D = centroids[i,:]
415.                 for j in range (0,len(M_clusters[i])):
416.                     perfil_1D = M_clusters[i][j,:]
417.                     dif = np.subtract(centroid_1D,perfil_1D)
418.                     dif_cuadrado = np.multiply(dif, dif)
419.                     average_perfil = np.average(dif_cuadrado, axis=0)
420.                     M_average_class = np.append(M_average_class,average_perfil)

421.                 Average_cluster = np.average(M_average_class, axis=0)
422.                 M_average_clusters = np.append(M_average_clusters, Average_clus
    ter)
423.
```

```
            MIA_per_cluster = np.sqrt(M_average_clusters)
            average_clusters = np.average(M_average_clusters)
            MIA = np.sqrt(average_clusters)
            return MIA, MIA_per_cluster

        def CDI(labels, centroids, consumers_use, N_clusters, CONST_SAMPLE_SIZE
    ):

            M_clusters = []
            for i in range(0,N_clusters):
                cluster = []
                fila = []

                for j in range (0, len(labels)):
                    if labels[j] == i:
                        for k in range(0,CONST_SAMPLE_SIZE):
                            fila.append(consumers_use[j][k])
                        cluster.append(fila)
                        fila = []


                cluster_np = np.array(cluster)
                M_clusters.append(cluster_np)

        #Distancia entre elementos de un mismo cluster/NUMERADOR

            M_average_class = np.array([])
            for i in range(0,N_clusters):
                M_average_element_class = np.array([])
                for l in range (0,len(M_clusters[i])):
                    perfil_modelo_1D = M_clusters[i][l,:]
                    M_average_element = np.array([])
                    for j in range (0,len(M_clusters[i])):

                        if l!=j:

                            perfil_1D = M_clusters[i][j,:]
                            dif = np.subtract(perfil_modelo_1D,perfil_1D)
                            dif_cuadrado = np.multiply(dif, dif)
                            average_element = np.average(dif_cuadrado, axis=0)

                            M_average_element = np.append(M_average_element,ave
    rage_element)

                    average_elemnt_class = np.average(M_average_element,axis=0)

                    M_average_element_class = np.append(M_average_element_class
    ,average_elemnt_class)
                average_class = np.average(M_average_element_class,axis=0)
                M_average_class = np.append(M_average_class,average_class)


            average_clusters = np.average(M_average_class, axis=0)
            numerador = np.sqrt(average_clusters)

        #Dispersion entre centroides

            M_centroid = np.array([])
            for i in range (0,N_clusters):
                centroid_modelo = centroids[i,:]
                M_centroid_centroid = np.array([])
                for j in range (0,N_clusters):
                    if i!=j:
                        centroid_1D = centroids[j,:]
                        dif = np.subtract(centroid_modelo,centroid_1D)
                        dif_cuadrado = np.multiply(dif, dif)
```

```
485.                         average_element = np.average(dif_cuadrado, axis=0)
486.                         M_centroid_centroid = np.append(M_centroid_centroid,ave
    rage_element)
487.                 average_centroid = np.average(M_centroid_centroid, axis=0)
488.                 M_centroid = np.append(M_centroid,average_centroid)
489.
490.             average_den = np.average(M_centroid, axis=0)
491.             denominador = np.sqrt(average_den)
492.
493.
494.             cdi = numerador/denominador
495.             return cdi
496.
497.
498.         def moving_average_centroids(centroids, n, N_clusters, CONST_SAMPLE_SIZ
    E) :
499.
500.
501.             m = n-1
502.             M_MA = np.array([],dtype=float).reshape(0,CONST_SAMPLE_SIZE)
503.             for i in range (0,N_clusters):
504.                 a = centroids[i,:]
505.                 #Para poder hacer la media móvil de las 00:00 necista las horas
     de 1,2 y 3 por eso se añaden
506.                 a = np.append(a,a[0:m])
507.                 ret = np.cumsum(a, dtype=float)
508.                 ret[n:] = ret[n:] - ret[:-n]
509.                 c = ret[n - 1:] / n
510.                 M_MA = np.r_[M_MA, [c]]
511.             return M_MA
512.
513.         def consecutive(data, stepsize=1):
514.             return np.split(data, np.where(np.diff(data) != stepsize)[0]+1)
515.
```