Master en Ingeniería de Telecomunicaciones

Trabajo de Fin de Master

Aplicación de técnicas de procesado de señal para la generación automática de características en Aprendizaje Automático

Autor
Álvaro Clemente Verdú

Supervisores
Pablo González Carrizo
Javier Matanza Domingo

Madrid
Abril 2020

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

APLICACIÓN DE TÉCNICAS DE PROCESADO DE SEÑAL PARA LA GENERACIÓN AUTOMÁTICA DE CARACTERÍSTICAS EN APRENDIZAJE AUTOMÁTICO

en la ETS de Ingeniería - ICAI de la Universidad Pontificia Comillas en el curso académico 2019/20 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.: Álvaro Clemente Verdú Fecha: 20/04/2020

Autorizada la entrega del proyecto

EL DIRECTOR DEL PROYECTO

Fdo.: Pablo González Carrizo Fecha: 20/04/2020

A mi tutor, Pablo González, por su inestimable guía y ayuda.
A BigML, y en particular a Francisco Martín, ejemplo e inspiración para cualquier ingeniero, por confiar en mi y darme esta oportunidad.
A mis amigos, y en especial a Pilar, por aguantarme y apoyarme en este largo proceso.
A mis padres y hermanos, sin vuestro apoyo y cariño hoy no tendría la oportunidad de escribir estas palabras.

APLICACIÓN DE TÉCNICAS DE PROCESADO DE SEÑAL PARA LA GENERACIÓN AUTOMÁTICA DE CARACTERÍSTICAS EN APRENDIZAJE AUTOMÁTICO

Autor: Clemente Verdú, Álvaro
Director: González Carrizo, Pablo
Entidad Colaboradora: BigML, Inc

RESUMEN DEL PROYECTO

Con el objetivo de hacer más accesible el *Machine Learning*, se ha creado una librería que automatiza la generación de características sobre señales aplicando técnicas de procesado digital de señales para hacer transformaciones y extraer características que sean analizables por estos algoritmos. Se ha implementado una prueba de concepto que ha dado resultados satisfactorios sobre un conjunto variados de datasets, y se ha utilizado con éxito en un caso de uso de mantenimiento predictivo en un proceso de fabricación de automóviles.

Palabras clave: *Machine Learning*, *Feature Engineering*, *característica*, *señal*

1. Introducción

El *Machine Learning* es la rama de la Inteligencia Artificial que más popularidad ha conseguido en los últimos años, gracias la unión de factores como avances en las tecnologías disponibles (tanto hardware como software) y en las investigaciones, junto a la creciente disponibilidad de una mayor cantidad de datos. *Machine Learning* se define como el estudio de algoritmos y técnicas que son capaces de mejorarse a sí mismos de forma automática, a partir de la experiencia, que se consigue con datos.

De esta forma, los datos son la clave de este proceso y se pueden encontrar en orígenes diferentes y en formatos que no siempre son compatibles con estos algoritmos que están diseñados para funcionar con valores numéricos. De esta forma la transformación de los datos se convierte en la fase del proceso en la que más tiempo se emplea [1], y es la que tiene una mayor influencia en el resultado final. Ya que estos algoritmos funcionan encontrando relaciones entre los datos, es muy importante que las características que componen estos datos reflejen estas relaciones de la forma más clara posible. Al proceso de aplicar conocimiento de dominio para generar características que expliciten estas relaciones se denomina *Feature Engineering* (o Ingeniería de Características).

Los datos pueden tener distintos tipos como numéricos, textuales, fechas, o en algunos casos de gran interés, señales. Las señales son listas de valores numéricos que tienen una relación secuencial, es decir, en las que el orden de los valores contiene información. Estos son datos que los algoritmos tradicionales no son capaces de procesar por si solos, y es necesario transformarlos. Sin embargo, hoy en día las técnicas de *Feature Engineering* para señales no están estandarizadas, y requieren conocimiento específico e investigación para encontrar las herramientas que permitan procesarlas.

2. Definición del proyecto

Con este proyecto se va a proponer un sistema que permita solucionar el problema de la falta de herramientas para el procesamiento automático de señales que reduzcan la barrera de entrada a proyectos de *Machine Learning* que requieran tratar con señales. Esto puede tener un gran impacto, ya que hay muchas aplicaciones de vital importancia que requieren del procesamiento de señales, como son aplicaciones médicas, prevención de desastres naturales, o aplicaciones industriales de control de calidad y prevención de riesgos.

De esta forma, se va a diseñar una librería que abstraiga las tareas de procesado digital de señales necesarias para hacer *Feature Engineering* con señales. Además, se va a utilizar esta librería como pieza clave en el proceso transformación de datos en un caso de uso industrial de detección y prevención de errores en una de las fábricas principales de una importante compañía internacional del sector automovilístico. Por último, se va a desarrollar un *benchmark* que permita hacer una comprobación empírica de la utilidad que tenga esta herramienta a la hora de modelar señales con distintos orígenes, y con distintas propiedades.

3. Descripción de la herramienta

Se quiere implementar una herramienta que genere unas características que sean útiles para un mayor rango de casos posibles, al mismo tiempo que se mantenga un nivel de usabilidad alto, de forma que se pueda reducir al máximo la barrera de entrada a esta rama del *Machine Learning*.

Para cumplir esta función, la librería sigue un proceso que se ha utilizado ya anteriormente con éxito en diversos casos de uso de procesamiento de señales. Este proceso, ilustrado en la Figura 1, se compone de 2 pasos principales:

1. Transformación de la señal: Se aplica una función matemática sobre la señal original para producir generalmente otra señal que exponga ciertas propiedades de esta de una forma que sea más fácil de extraer. Un ejemplo de estos algoritmos es la *Trasformada Rápida de Fourier (FFT)*.

2. Aplicar un algoritmo de extracción de características sobre las transformaciones, de forma que de estas nuevas señales que se han producido se puedan extraer unos pocos valores que representen propiedades concretas que en su conjunto mantengan la mayor cantidad posible de información sobre la señal original para facilitar su modelado. Un ejemplo de estos algoritmos es la detección de picos, que extraen máximos locales de una señal que cumplen ciertos requisitos.

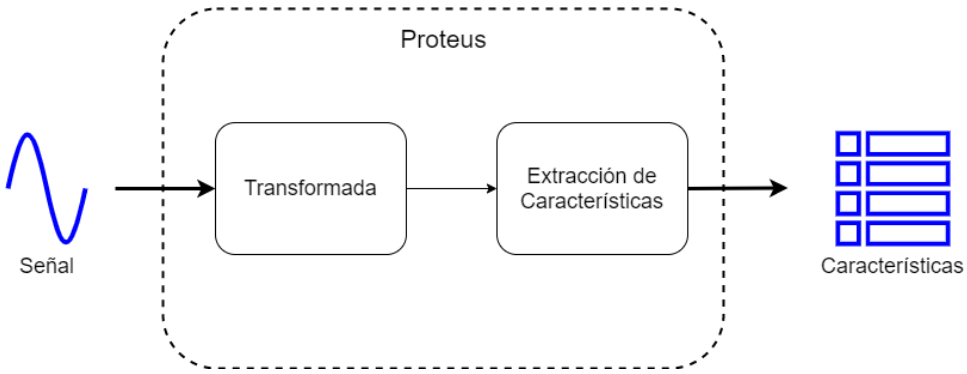

Figura 1- Arquitectura de la librería

La usabilidad de la librería se muestra también en la Figura 1, donde se observa que para funcionar lo único que se necesita es la señal de entrada, y con una configuración por defecto se producen características que son útiles para la mayoría de los casos. Además, se permite la configuración en caso de que un usuario experto quiera adaptar y optimizar el funcionamiento de la librería a su caso de uso particular. Para elegir el conjunto de transformaciones y características que den mejores resultados, se ha implementado un *benchmark* que permita obtener una valoración empírica de su utilidad. Finalmente, se han elegido las siguientes características:

- Estadísticos de la señal original
- Top K picos de la Transformada de Fourier (FFT)
- Top K picos de la Densidad Espectral de Potencia (PSD)
- Estadísticos de la Función de Autocorrelación (ACF)
- Top K picos de la Transformada de Ondícula Continua (CWT)
- Opcionalmente, una medida de la distancia a una señal de referencia

4. Resultados

En primer lugar, se ha implementado una primera versión de la librería para desarrollar una prueba de concepto de la solución desarrollada para el caso de uso. En esta solución, se utilizan las características generadas como entrada de un modelo de detección de anomalías para detectar posibles errores en un proceso de soldaduras, y generar una alerta para su revisión.

Estos modelos son no supervisados, es decir, no se tiene acceso a información sobre errores reales durante el entrenamiento. Se ha generado un modelo por cada una de las máquinas soldadoras, y se han evaluado con unas pocas muestras de soldaduras que se sabe que resultaron en un error. Por la naturaleza del problema, se busca un modelo que sea capaz de detectar todos los errores, con el menor número de falsos positivos posibles. Esto se mide con recall y precision, respectivamente. Para 3 entidades diferentes, los resultados obtenidos se muestran en la Tabla 1. Analizando los resultados se puede que los modelos son capaces de detectar errores, aunque en algunos casos con una tasa alta de falsas alarmas. Aun así, esto valida la hipótesis de que estos modelos pueden utilizarse con este fin, y se considera un éxito.

Entidad	*Recall (%)*	*Precision (%)*	*Errores*
Entidad A	5.82	100	6
Entidad B	100	100	1
Entidad C	0	0	3

Tabla 1 - Resumen de la evaluación del caso de uso

Este resultado sirve también para validar la utilidad de la librería en un caso de uso real. Al analizar los modelos generados se ha observado que las características generadas por la librería eran las más importantes para detectar los errores. Sin embargo, se ha visto que había redundancia y margen de mejora.

Para solucionarlo, se han realizado distintos experimentos sobre el *benchmark* diseñado con el fin de elegir la combinación de características más útil. El resumen de los resultados se muestra en la Figura 2:

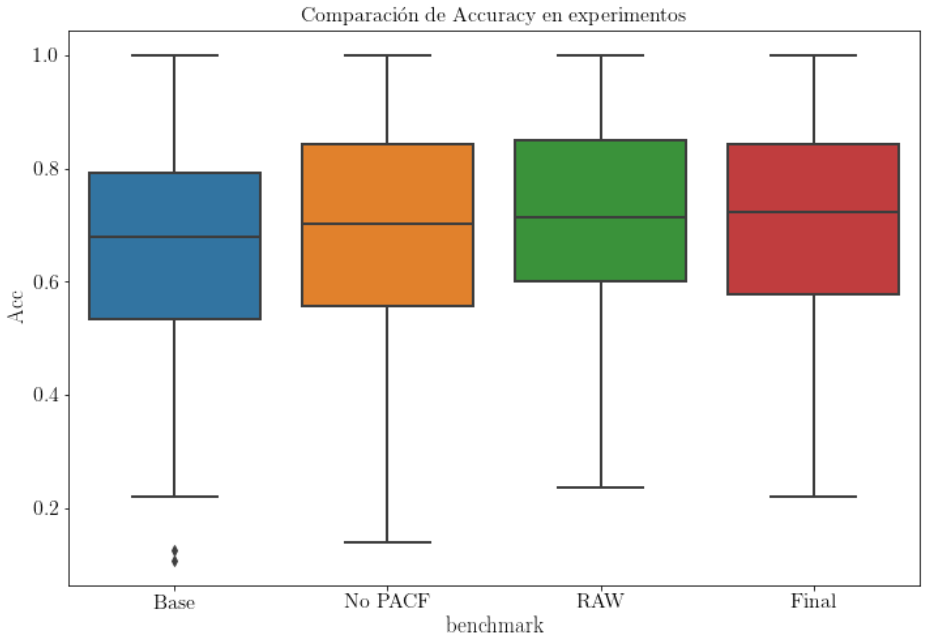

Figura 2- Comparación de accuracy en experimentos realizados sobre 116 conjuntos de datos etiquetados

Con los resultados de esta evaluación, se han elegido eliminado algunas características sobre el conjunto original para el caso de uso (como la PACF), y se han añadido nuevas características (como las relacionadas con la señal original), resultando en el conjunto de características que se ha descrito en el capítulo 3, con el que se ha conseguido un *accuracy* medio *0.704*, al nivel de otras soluciones similares.

5. Conclusiones

En *Machine Learning*, los modelos tradicionales siguen siendo las opciones más utilizadas con diferencia en la industria, gracias al equilibrio que ofrecen entre

simplicidad, interpretabilidad junto con buenos resultados, y esto quiere decir que hacen falta herramientas para extender y facilitar su uso, especialmente para casos como son el tratamiento de señales que antes se evitaban. Este tipo de herramientas tienen el potencial de generar un gran impacto, acercando estas técnicas a un público más general.

A la hora de implementar esta solución, se ha descubierto que la simplicidad es difícil, y abstraer tareas complejas como son el tratamiento de señales, exige un trabajo importante de diseño. Un ejemplo de esto es la elección de las transformaciones, que son operaciones complejas que pueden generar valores difíciles de interpretar, y esto limita la usabilidad de la librería. De esta forma ha sido necesario un esfuerzo adicional para mantener un equilibrio entre utilidad y usabilidad.

En cuanto a los trabajos futuros, aunque se ha validado la prueba de concepto, la librería sigue teniendo margen de mejora, pudiendo estudiar nuevas características que puedan mejorar su utilidad en campos como el procesamiento de audio, donde esta versión no obtiene los resultados deseados. Se debe realizar un estudio más profundo de sus fortalezas y puntos débiles, y probar en otros casos reales variados. Además, hay margen de mejora desde un punto de vista de la eficiencia de la implementación, lo cual beneficia a casos de uso de tiempo real.

Por otro lado, se propone desarrollar un *benchmark* más completo, con una metodología más robusta como *train, test y validation split*, que permite realizar una estimación de la capacidad de generalización con uno subconjunto de los datos que no se han utilizado en el entrenamiento o diseño (*validation*), o también el uso de *Cross Validation*, técnicas que reducen el *overfitting* que se puede hacer a los conjuntos de datos concretos del *benchmark*.

6. Referencias

[1] State of Data Science and Machine Learning 2019. https://www.kaggle.com/kaggle-survey-2019.

[2] Hoang Ahn Dau y col. “The UCR time series archive”. En *IEEE/CAA Journal of Automatica Sinica 6.6 (2019)*

USING DIGITAL SIGNAL PROCESSING TECHNIQUES FOR AUTOMATIC FEATURE GENERATION IN MACHINE LEARNING

Author: Clemente Verdú, Álvaro
Director: González Carrizo, Pablo
Collaborating Entity: BigML, Inc

ABSTRACT

With the goal of making *Machine Learning* more accessible, we build a library that automates feature generation from signals by using digital signal processing techniques for applying transformations and extracting features from them. We implemented a proof of concept which yielded satisfactory results in a general benchmark developed for this purpose. This library was also used successfully in a real-world use case of predictive maintenance in a car manufacturing process.

Keywords: *Machine Learning, Feature Engineering, feature, signal*

1. Introduction

Machine Learning is one of the branches of *Artificial Intelligence* which has gained more popularity in the recent years, due to the convergence of several factors such as technology advances (both in hardware and software) and advances in research, as well as the increase in availability of large datasets. *Machine Learning* is defined as the study of algorithms and techniques that can improve themselves automatically from experience, achieved through data.

Therefore, data is key in this process, and it can be found in different locations and in different formats, which are not always compatible with these algorithms, designed to work with numerical data. For that reason, data transformation is a key step in the *Machine Learning* process, and it is the one in which more time is spent [1], and the one which has the biggest influence in the performance of the model. Since these algorithms work by learning patterns in the data, it is very important that the features contained in this data reflect these patterns as explicitly as possible. The process of using domain specific knowledge to generate features that express these relations and patterns is called *Feature Engineering*.

Features can have different types, such as numeric, text, dates, or in some interesting cases, signals. A signal is a list of numerical values which have a sequential relation, that is, the order of the values conveys information as well. This type of data as it is cannot be processed by traditional algorithms, and must be transformed into numerical features. However, *Feature Engineering* techniques for signals are not well standardized and require specific knowledge and research to find the right tool to perform these tasks.

2. Project Definition

In this project a tool will be built to solve the problem of lack of standard and simple tools to perform automatic feature generation from signals, lowering the barrier of entry of *Machine Learning* projects that require dealing with this type of data. This

can have a big impact, since there are many critical use cases that can benefit from processing signals, such in medical applications, natural disaster prevention or industrial applications of quality control and risk prevention.

For this purpose, we will build a library that abstracts all digital signal processing tasks required to perform *Feature Engineering* on signals. In addition, this library will be used as a key piece in the data transformation pipeline of a real industrial use case of predictive maintenance, detecting and preventing errors in a car manufacturing process. Lastly, a benchmark will be developed to get an empirical estimate of the general performance of this library in a variety of datasets with different kinds of signals which have different properties.

3. Tool Description

The goal is to build a tool which generates features that are useful for as many different use cases as possible, at the same time maintaining a high level of usability, so that we can keep the barrier of entry low.

To solve this, the library will perform a two-step process which has been used successfully for this kind of task before. These two steps, illustrated in Figure 1 are the following:

1. Signal Transformation: A mathematical function is applied to the original signal, producing a new signal which exposes some properties of the signal in a way that is easier to extract. An instance of one of these algorithms is the *Fast Fourier Transform (FFT)*.

2. Apply a feature extraction algorithm to the result of the previous transformation, so that these new signals produced can be reduced to a small set of significant values which can be used as features for modeling. For instance, *Peak Detection* algorithm can be used to extract local maxima with some specific requirements.

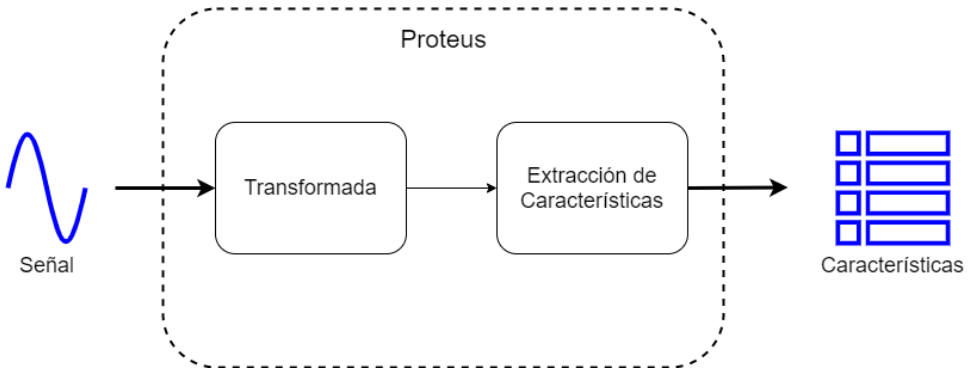

Figure 1- Architecture

In the Figure 1 the usability of the library can be observed as well, since in order to work it only needs the input signal, using the default configuration, which generates features useful for the average use case. In addition, full configuration is supported, so that an expert user can tune and optimize the behavior of the library to his specific use case. To pick the most relevant set of features, a benchmark has been implemented, to provide an empirical evaluation of the average performance of the

library across a variety of use cases. Finally, the set of features generated by the library can be summarized in:

- Descriptive statistical parameters of the original signal
- Top K peaks from the Fast Fourier Transform (FFT)
- Top K peaks from the Power Spectral Density (PSD)
- Descriptive statistical parameters of the Autocorrelation Function (ACF)
- Top K peaks from the Continuous Wavelet Transform (CWT)
- Optionally, a measure of the distance to a reference signal

4. Results

First, an early version of the library was built to develop a proof of concept solution for the use case. In this solution, the features are used as input to an *Anomaly Detector* model whose mission is to detect potential errors during a welding process and generate an alert.

These are unsupervised models, which means that they do not have access to the real label of each of the samples during training. A model has been generated per welding gun, and each has been evaluated using a small test set comprised of real production errors. The goal is to create a model that detects all the errors, while maintaining a low false positive rate. This is measured with precision and recall evaluation parameters, respectively. For 3 different entities, the results achieved are shown in Table 1. Looking at the results, the models can detect most of the errors, although some of them have high false positive rates. Even so, these results are considered a success, and they validate the hypothesis that these types of models can be used to detect these errors.

Entity	*Recall (%)*	*Precision (%)*	*Errors*
Entity A	5.82	100	6
Entity B	100	100	1
Entity C	0	0	3

Table 1- Evaluation Summary

These results also validate the usefulness of the features generated by the library in a real use case. After analysis, the features generated by the library appear to be the most important features for detecting these errors. However, there was some redundancy and margin for improvement.

To make these improvements, several experiments were run using the developed benchmark. The summary of the results is shown in Figure 2.

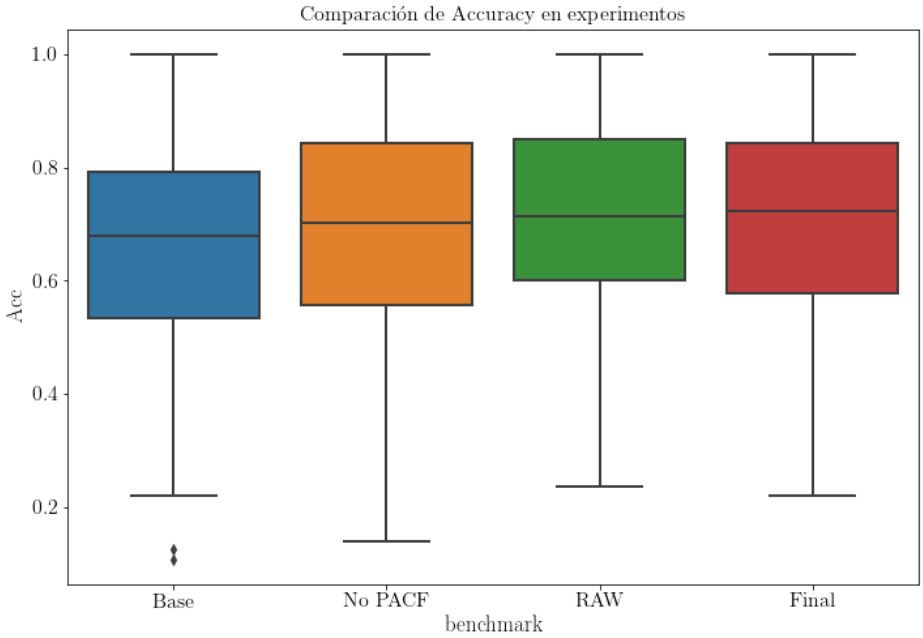

Figure 2- Accuracy comparison across experiments

Using these results some features were removed (such as PACF) and some others were added (for instance, original signal statistics), resulting in the feature set described on chapter 3. With this set of features, the library achieved a *0.704* mean accuracy across all the datasets, which is on par with other similar solutions.

5. Conclusions

Traditional *Machine Learning* models are still the most popular option across industries, due to the balance between simplicity, interpretability and good performance, and that means that there is a need for tools that extend the use of these models, especially on cases where traditionally they have been avoided, such as with signals. These kinds of tools have the potential to have a big impact on society, making these techniques more approachable for the general user.

After implementing this solution, the main takeaway is that *simplicity is hard*, and abstracting complex tasks such as signal processing in a way that makes them both easy and powerful requires hard design work. An example of this is the decision of which transformations to include, since these are inherently complex algorithms that can generate features that are hard to interpret, which would hinder the usability. In this manner, this design was a constant tradeoff between utility and usability.

As of future developments, even though the proof of concept was validated, more work is required to improve the library, researching new transformations and techniques to generate the best features, especially in use cases where the current feature set does not perform as wished, such as audio processing. A deeper study should be conducted to get a better sense of the strengths and weaknesses of the library, and more extensive testing on other real-world use cases. Also, from an efficiency perspective, optimizations are very useful, especially for real time use cases.

Lastly, a new benchmarking process should be developed, one that used more robust methodologies such as a *3-way split (train, test, validation)* which makes an estimation of the generalization capabilities of the process by evaluating it with a subset of the data that it was never seen during training of design, as well as *Cross Validation*, which are techniques that lower the chance of *overfitting* on the specific datasets and splits in the benchmark.

6. References

[1] State of Data Science and Machine Learning 2019. https://www.kaggle.com/kaggle-survey-2019.

[2] Hoang Ahn Dau et al. "The UCR time series archive". In *IEEE/CAA Journal of Automatica Sinica 6.6 (2019)*

Índice general

Índice de figuras

Índice de tablas

Capítulo 1

Introducción

La *Inteligencia Artificial*, o *IA*, es una rama de la ciencia que se encuentra en la intersección de muchas otras ramas como son la informática, matemáticas, biología y filosofía, entre otras. La *IA* busca realizar modelos computacionales del comportamiento humano. Este es un campo de investigación muy activo y apasionante, que ha capturado el interés de las mejores mentes de las últimas décadas y ha inspirado innumerables historias.

Una de las ramas de la *IA* que más popularidad, y más avances ha encontrado en los últimos años, es el *Machine Learning*, o *Aprendizaje Automático*. El *Machine Learning* es el estudio de algoritmos y técnicas que son capaces de mejorarse a sí mismos a partir de la experiencia. Los modelos de *Machine Learning* son funciones matemáticas que son capaces de adaptar su funcionamiento a raíz de un conjunto de datos que se les provee como "entrenamiento". De esta forma los modelos construyen una experiencia, que les permite optimizar un objetivo concreto. A partir de estos datos, los algoritmos "aprenden", memorizando de distintas formas patrones y propiedades estadísticas en los datos de entrenamiento. Estas relaciones se utilizan después para realizar predicciones sobre nuevas muestras.

En la Figura 1.1 se puede ver una descripción de la relación entre el *Machine Learning* y la *IA*, incluyendo técnicas modernas más complejas como *Deep Learning*, que es una rama del *Machine Learning* que engloba algoritmos basados en combinaciones complejas de unidades de procesamiento formando cascadas de capas, como son las redes neuronales.

Este tipo de aprendizaje ha demostrado ser muy útil desde un punto de vista práctico, y se utiliza con éxito hoy en día en numerosas aplicaciones, desde el filtrado de correo no deseado hasta detección y prevención de enfermedades, pasando por reconocimiento de voz o conducción autónoma.

El *Machine Learning* se encuentra en su momento más popular desde que comenzó a desarrollarse en los años 50. Esto se debe a una mezcla de avances en hardware, en software, en la investigación, que permiten también la generación y

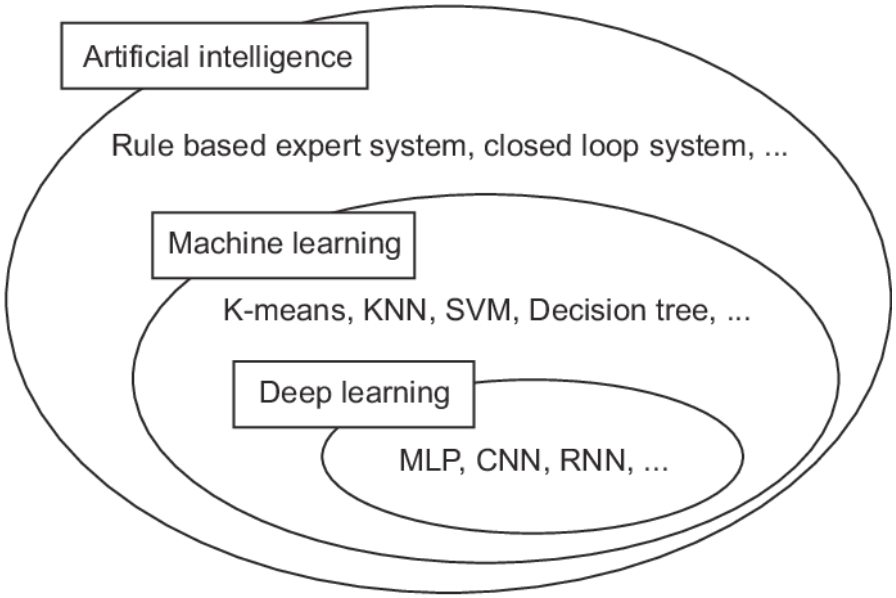

Figura 1.1: Partes de la Inteligencia Artificial [1]

almacenamiento de mayores cantidades de datos, y en el comienzo de la adopción de estas técnicas por parte de la industria. Empresas como Google, Amazon, Microsoft o Netflix han demostrado que utilizar las técnicas de *Machine Learning* en su negocio supone una gran ventaja competitiva, y que el que no implemente el siguiente nivel de automatización que estas técnicas permiten, corren el riesgo de perder sus ventajas con respecto a sus competidores. Es por esto que hay un apogeo en la cantidad de proyectos relacionados con el *Machine Learning* desde todas las industrias, y una carrera por encontrar el caso de uso que suponga la mayor ventaja competitiva.

La clave en todo proyecto de *Aprendizaje Automático* son los datos que se utilizan para entrenar los algoritmos. Un modelo solo va a ser tan bueno como los datos con los que trabaja. Y estos datos pueden estar separados en múltiples lugares y en diferentes formatos, y por eso es necesario pasar por una fase de transformación que prepare los datos en una estructura que permita un modelado lo más sencillo posible.

Como ya se ha comentado, el funcionamiento de los algoritmos de *Machine Learning* se basa en encontrar patrones en los datos que se les provee a la entrada, buscando relaciones matemáticas y analizando las propiedades estadísticas de los datos. De esta forma, los resultados que se obtienen son muy sensibles a la estructura y las propiedades de los datos que se tienen.

Es por esto, que en todo proyecto de *Machine Learning*, una de las tareas más importantes, y en las que más tiempo se emplea es la fase de *Feature Engineering*.

Feature Engineering consiste utilizar el conocimiento de dominio para modificar la estructura de los datos, y crear nuevas características con el objetivo de mejorar los resultados obtenidos en la fase de modelado. Dichas características buscan exponer de forma explícita propiedades y relaciones que de otra forma el algoritmo tendría que descubrir de forma automática, cosa hace que el modelo y su entrenamiento sean más complicados, o incluso puede ser imposible. Este proceso es el más determinante en el éxito de un proyecto de *Aprendizaje Automático*, y es donde más esfuerzo y tiempo se debe dedicar.

Existen técnicas estándar diferentes, en función del tipo de datos que se esté procesando. Uno de los tipos de datos más importantes en muchos problemas son las señales. Las señales son sucesiones de valores, en una o más dimensiones, ya sea a lo largo del tiempo (un voltaje, una corriente) o en el espacio (una imagen). Hay muchos tipos de problemas y aplicaciones donde las señales son una característica clave:

- Médicos: electrocardiogramas, encefalogramas
- Desastres Naturales: sismogramas
- Industriales: corrientes, tensiones provenientes de sensores en procesos industriales
- Financieros: precios de activos

En todos estos casos, las señales son características clave que tienen una gran cantidad de información codificada en ellas, o en algunos, son la única fuente de información disponible. Transformarlas es necesario, ya que los modelos tradicionales no pueden procesar secuencias de valores, por lo que es necesario extraer las propiedades de forma manual. Procesar y extraer información de ese tipo de señales es un proceso complicado que requiere de conocimientos específicos de comunicaciones y procesado de señales que no siempre son accesibles para científico de datos medio.

Este es el contexto en el que surge este proyecto. En él, se van a plantear tanto la motivación como el proceso de diseño, implementación y prueba de una herramienta que haga las tareas de *Feature Engineering* sobre señales accesibles a cualquier persona, sin necesidad de tener conocimientos específicos sobre técnicas de tratamiento de señales.

Adicionalmente, se va a comentar su papel como pieza clave en un caso de uso real de la industria. Esta herramienta va a ser el componente principal del proceso de *Feature Engineering* del *pipeline* de *Extracción, Transformación y Carga*, o *ETL*, de un proyecto de detección de errores para mantenimiento predictivo en un proceso industrial de fabricación de automóviles.

Este es un proyecto que se ha hecho en colaboración con BigML, que es una empresa que ha desarrollado la primera plataforma de *MLaaS*, y cuyo objetivo es hacer el *Machine Learning* accesible para todo el mundo. Así, además de ofrecer una herramienta web desde la que se puede realizar modelado de *Machine Learning*, realizan cursos de formación, eventos y proyectos en colaboración con distintas empresas para promover la adopción de estas técnicas. Este proyecto surge para apoyar esta misión.

Capítulo 2

Descripción de las tecnologías

En este capítulo se van a describir las principales tecnologías utilizadas. Para ello se van a mencionar las distintas decisiones que es necesario tomar a la hora de elegir las tecnologías que se van a utilizar para el proyecto. Los objetivos, requisitos y decisiones de diseño del proyecto se explicarán más en detalle en el Capítulo 4. La elección de las tecnologías es una de las decisiones más importantes que se van a tomar a la hora de desarrollar una herramienta, ya que son estas las que van a formar el marco sobre el que se tomen el resto de decisiones en el diseño y la implementación.

Para ello es necesario hacer un estudio de las ventajas e inconvenientes de cada una de las posibles elecciones, y tomar decisiones teniendo en cuenta el equilibrio entre rapidez de desarrollo, facilidad de modificación y extensión en el futuro, y la usabilidad de la herramienta. También es necesario tener en cuenta las limitaciones que se tienen en el entorno en el que se va a desplegar el caso de uso principal, que van marcadas por el cliente.

El proyecto se centra en desarrollar una herramienta general para extracción de características o *Feature Engineering* en señales para aprendizaje automático. Con este proyecto se busca ofrecer una solución útil, fácil de usar y eficiente (los objetivos están detallados en la Sección 4.2). También se busca que el proceso de desarrollo sea rápido y ágil. Estos objetivos están en mente a la hora de elegir las tecnologías.

Con esto en mente, las elecciones principales que hay que tomar en cuanto a las tecnologías a utilizar para la herramienta de *Feature Engineering*:

- Lenguaje de Programación
- Herramientas numéricas y científicas
- Herramientas de *Aprendizaje Automático*, en particular, Plataformas *MLaaS*

Aunque la herramienta de *Aprendizaje Automático* no debería influenciar el desarrollo de una solución de *Feature Engineering*, se va a tener en cuenta ya que se consideran un entorno en el que esta herramienta se puede integrar para maximizar su impacto. En particular, uno de los objetivos tras la consecución de este proyecto será su integración dentro de BigML, una de estas plataformas.

2.1. Lenguaje de programación

La elección del lenguaje de programación en el que implementar la solución es una de las más importantes, ya que influye en el resto de tecnologías que se van a utilizar. Existen decenas de opciones diferentes a la hora de elegir un lenguaje. A la hora de decidir en un lenguaje hay que plantearse:

- Capacidad de procesado numérico
- Herramientas disponibles, especialmente científicas
- Ecosistema, disponibilidad de recursos de ayuda
- Popularidad
- Agilidad en el desarrollo

En cuanto a la capacidad de procesado numérico, hoy en día la gran mayoría de los lenguajes populares ofrecen soluciones para la implementación de programas basados en el procesado numérico y científico. Aun así, hay algunos lenguajes que están mejor equipados para este cálculo que otros. Por ejemplo, en lenguajes como *Javascript* es necesario usar librerías y extensiones del lenguaje para hacer cálculos precisos, ya que por defecto todos sus tipos numéricos usan el IEEE 754 estándar de coma flotante [2].

Este estándar permite representar un enorme rango de valores en solo 64 bits. Sin embargo, el limitar su tamaño hace que la precisión sea finita, en particular para valores cercanos a 0 y valores muy grandes.

Por ejemplo:

```
> 0.1 * 3
> 0.30000000000000004
```

Snippet 2.1: Ejemplo de imprecisión en Javascript

De esta forma, entre los candidatos principales para este proyecto se encuentran:

- Python
- R
- Matlab
- Clojure
- Java
- C
- C++

De esta lista cabe destacar *Clojure*, que es un lenguaje menos popular que los demás. Es un lenguaje principalmente funcional, de la familia de lenguajes basados en *LISP*, que ofrece estructuras de datos inmutables, y otras primitivas que lo hacen un buen candidato para código concurrente. Clojure produce código ejecutable en la JVM, y permite una integración casi perfecta con cualquier librería escrita en *Java*, lo que hace que a pesar de ser el lenguaje menos popular de la lista, le da acceso a todo el ecosistema *Java*, que está ampliamente probado. Además, este es el lenguaje en el que está implementado el *backend* de BigML, por lo que la elección es atractiva desde el punto de vista de una futura integración con el resto de la plataforma.

Aunque a priori esta podría ser una buena elección, el implementar una librería de esta complejidad al mismo tiempo que se aprende a diseñar programas con un paradigma tan diferente como el que promueve *Clojure* hacen que esta no se haya considerado como la mejor decisión, ya que para este proyecto se require una iteración rápida. Sin embargo, se considera como una muy buena opción a futuro, observando como evoluciona el ecosistema de herramientas científicas en este lenguaje.

Por un razonamiento similar, se han eliminado *Java*, *C* y *C++*. Todas estas opciones son lenguajes populares, maduros con unos ecosistemas establecidos. Sin embargo su naturaleza estática, junto con sus implementaciones los hace extremadamente eficientes en la ejecución de código, los hace demasiado lentos a la hora de desarrollar. Estas opciones se considerarán si el rendimiento de la librería no es el esperado con lenguajes dinámicos.

Por su parte, Matlab se ha descartado porque, a pesar de ser ciertamente popular en entornos científicos y académicos, el hecho de no ser de código abierto, y su alto coste de licencia limita tanto el acceso para desarrollar como la cantidad de futuros usuarios de la herramienta.

De esta forma, la elección se ha reducido a las dos opciones más populares en el espacio de ciencia de datos (*Data Science*), *Machine Learning* y *Feature Engineering* que son *R* y *Python*. Llegados a este punto, cualquiera de las dos opciones

es válida, ya que ambos son lenguajes dinámicos, con un amplio ecosistema de librerías y de usuarios con recursos de ayuda, que los hace ideales para tareas de procesamiento de datos como esta.

El lenguaje que se ha elegido es *Python*. De las dos opciones, es el más popular, y se ha convertido en el estándar *de facto* en esta industria. Y no es por casualidad. Aunque las dos opciones son similares, *Python* tiene varias ventajas sobre *R* que hacen que en la mayor parte de los casos sea la mejor decisión.

La principal discrepancia es que, al contrario que *R*, que se diseñó para tratar datos, *Python* es un lenguaje de propósito general. Esto se traduce en que de forma nativa, las características del lenguaje faciliten tareas muy diversas, como acceder al sistema de archivos, abrir conexiones a servidores externos o incluso implementar sistemas enteros. Esto se puede hacer en *R* también, pero requiere de mucho más trabajo, y la expresividad que ofrece a la hora de procesar datos no se ve reflejada en otras tareas.

Por esta misma razón, *Python* tiene una base de usuarios mucho mayor que *R*, y esto naturalmente se traduce en una mayor cantidad de recursos y librerías de calidad disponibles. Además, permite implementar todas las partes del proyecto en el mismo lenguaje, lo que facilita mucho la integración, la reutilización de código entre las distintas partes, y reduce los cambios de contexto que deben hacer los desarrolladores.

2.2. Herramientas numéricas y científicas

Una vez elegido el lenguaje, es necesario elegir las herramientas que se van a utilizar para realizar el procesamiento de los datos, y en particular las transformaciones sobre las señales.

Desde un primer momento, se va a intentar aprovechar implementaciones ya existentes de algoritmos conocidos para procesamiento de los datos y señales. De esta forma, se aprovechan librerías *Open Source*, de código abierto, que ya han sido probadas y que tienen implementaciones eficientes.

En *Python* existe un enorme ecosistema de librerías para procesamiento numérico y científico. En este ecosistema cabe destacar numpy, pandas y Scikit-Learn.

Numpy [3] es la librería por excelencia para el procesamiento numérico en *Python*. Se basa en el procesamiento extremadamente eficiente de vectores numéricos. Para ello ofrece un contenedor de datos, el *ndarray* que representa un vector n-dimensinal que puede contener diferentes tipos de datos, y una serie de primitivas que operan sobre estos datos de forma eficiente. Para conseguir la velocidad de ejecución, las primitivas están implementadas en *C*, gracias a una interfaz nativa de *Python* con *C*, y operaciones enteras sobre un *ndarray* son ejecutadas directamente por código *C*. Sobre esta librería y sus primitivas se construyen el resto de

librerías que se va a mencionar, y también este proyecto.

Pandas [4] es una librería que extiende *numpy* ofreciendo dos estructuras de datos pensadas para el tratamiento de datos en formato de tabla: *Series* y *DataFrame*. Esta librería introduce un *Domain Specific Language* o *DSL* para el tratamiento de datos el columnas y tablas, inspiradas en el *Dataframe* de *R*.

Scikit-Learn [5] es una familia de librerías que extienden las dos anteriores añadiendo un estructuras de datos, interfaces y operaciones centradas en el *Machine Learning*. Definen una interfaz estándar que debe implementar cualquier *modelo*, o sistema que coge como entrada un *ndarray*, *Series* o *DataFrame* y lo transforma de alguna manera. También incluyen implementaciones de una gran variedad de modelos, transformaciones estándar, evaluaciones y formas de combinarlas. Esta es una de las librerías más completas para *Aprendizaje Automático*, y el hecho de basarse en una interfaz tan simple, junto con la naturaleza dinámica de *Python* hace que extenderla sea muy sencillo.

En la misma familia se encuentra Scikit-Image [6], que ofrece numerosas funciones centradas en el procesado de imágenes. Todas estas librerías forman parte de una gran familia de librerías para el procesamiento numérico y científico en *Python*, SciPy [7].

En esta familia de herramientas se encuentra matplotlib [8], que ofrece herramientas para producir gráficos. Esta herramienta permite un control granular sobre la apariencia de los gráficos, lo que la hace muy potente, aunque en ocasiones en las que se busca sencillez puede ser resultar tosca. Esta librería ofrece una interfaz que está fuertemente inspirada en la que ofrece *Matlab*, lo que permite que la gente que tiene experiencia en ese entorno pueda cambiar a *Python* y la pueda aplicar directamente.

En la Figura 2.1 se muestra un ejemplo de una figura generada con *numpy* y *matplotlib*. El código que genera el gráfico es el siguiente [9]:

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

# Data for plotting
t = np.arange(0.0, 2.0, 0.01)
s = 1 + np.sin(2 * np.pi * t)

fig, ax = plt.subplots()
ax.plot(t, s)

ax.set(xlabel='tiempo (s)', ylabel='voltaje (mV)',
       title='Grafico Simple')
```

```
ax.grid()

fig.savefig("test.png")
plt.show()
```

Snippet 2.2: Código de muestra de gráfico con matplotlib y numpy

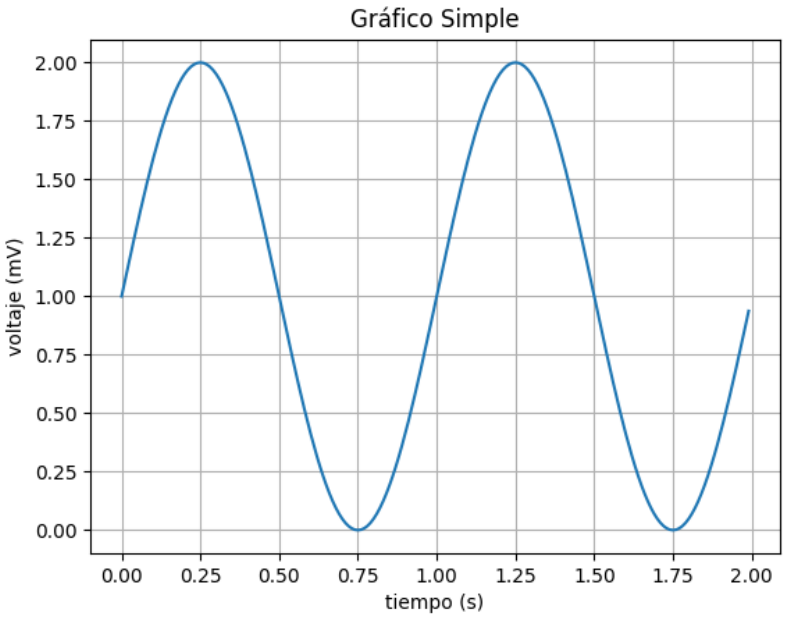

Figura 2.1: Ejemplo de gráfico con *numpy* y *matplotlib*

Algunas de las librerías principales que se han utilizado para implementar este proyecto son las siguientes:

- numpy: Para cálculos numéricos en general, y como estructura de datos base
- statsmodels: Contiene implementaciones para el cálculo de estadísticos
- scipy: Contiene implementaciones científicas en general, como para calcular la entropía
- PyWavelets: Incluye implementaciones de funciones relacionadas con *Wavelet Transform*
- dtw-python [10]: Es una implementación del algoritmo de *Dynamic Time Warping*

Para utilidades y visualizaciones se han utilizado también:

- matplotlib: Para crear gráficos de visualización de transformaciones y características en general, como Figura 5.6
- scaleogram: Para crear visualizaciones sobre la *Wavelet Transform*

2.3. Plataformas MLaaS y BigML

Este proyecto se ha desarrollado con el objetivo de poder ser integrado en el futuro dentro de una plataforma *MLaaS*.

Las plataformas *MLaaS*, o *Machine Learning as a Service* son herramientas online que agrupan una serie de servicios relacionados con los *workflows* típicos de *Machine Learning*, haciendo de estas tareas accesibles a un público más amplio. Estos servicios abstraen la gestión toda la complejidad relacionada con el almacenamiento de datos, implementación de modelos, computación distribuida, evaluación, predicción, y otras tareas más específicas. De esta forma el usuario puede concentrarse en las tareas de un proyecto de *Machine Learning* que son específicas de su caso de uso e implementar su lógica de negocio, confiando el resto de responsabilidades en las plataformas.

Entre los servicios que se ofrecen en este tipo de plataformas, se incluyen tareas típicas de *Feature Engineering*, que permiten realizar transformaciones sobre los datos dentro de los límites de la abstracción que ofrece la plataforma. Integrar la librería de *Feature Engineering* de señales en una de estas plataformas de *Aprendizaje Automático* puede amplificar enormemente el impacto que esta tiene, ya que le da acceso a un público que de otra forma no la descubriría.

En este espacio las soluciones más populares son *AWS Machine Learning* de Amazon, *Google AI Platform* de Google, *Azure Machine Learning* de Microsoft, que son ejemplos de plataformas que están integradas dentro de una amplia oferta de servicios de computación en la nube, lo que les da la ventaja de integrarse de forma nativa con otras herramientas como puede ser el popular servicio de almacenamiento *S3* en Amazon.

Sin embargo, también existen otras soluciones independientes, que están más focalizadas en las tareas específicas de *Machine Learning*, entre las que destaca BigML [11]. En este proyecto se va a analizar más en detalle esta última.

BigML

BigML fue la primera plataforma que ofreció servicios de *Machine Learning* en la nube. Fundada en 2011, BigML tiene el objetivo de hacer el *Machine Learning*

accesible a todo el mundo, y esta misión se alinea perfectamente con los objetivos que se van a definir para esta solución. De esta forma, BigML va a ser un colaborador principal en el desarrollo de este proyecto.

Para hacer el *Machine Learning* accesible, la plataforma ofrece métodos para abstraer las distintas tareas típicas de un flujo de *Machine Learning*, y operar sobre ellas a través de un concepto que se denomina *recurso*. En BigML, todo es un recurso, y se ofrece una interfaz para crearlos, modificarlos y combinarlos. Aquí hay una lista de algunos de los recursos principales disponibles en la plataforma:

- Source: Representa una fuente de datos, con sus columnas y tipos. Las típicas son *CSV*, *JSON*, *inline* o directamente desde bases de datos como *Postgres* o *Elasticsearch*
- Dataset: Es una fuente de datos ya procesada en formato interno de BigML, lista para modelar.
- Model, Ensemble, DeepNet, etc.: Representan implementaciones concretas de modelos de *Machine Learning*, listos para entrenar, evaluar y generar predicciones.
- Evaluation: Recurso que encapsula un conjunto de métricas sobre el comportamiento de un modelo al pasarle un *Dataset*
- Script: Representa un flujo nuevo de datos, una combinación arbitrariamente compleja de recursos
- Execution: Representa el resultado de ejecutar un Script

Una de las claves que permiten que la plataforma sea accesible es permitir la automatización e integración con sistemas externos, y para ello, la solución más aceptada hoy en día es el exponer las funcionalidades de la plataforma mediante una *API*.

Una *API*, o *Application Programming Interface* es una interfaz que expone funcionalidades de un sistema hacia el exterior, y define como otros componentes externos pueden interactuar con el sistema. Con esto, se define un estándar que permite que sistemas externos accedan de forma controlada a funcionalidades ofrecidas por el sistema. Existen muchas formas de definir una *API*, que pueden ir desde implementaciones completamente personalizadas, hasta el uso de estándares predefinidos. Las más populares, especialmente entre servicios web, son las *API REST*, que es un estándar que se basa en los métodos típicos de *HTTP*: *GET*, *POST*, *PUT*, *UPDATE* y *DELETE* que se utilizan para dar significado a las distintas operaciones que se pueden realizar sobre un sistema.

BigML sigue una filosofía de desarrollo *API first*. Esto quiere decir que todas las acciones disponibles en la plataforma tienen una interfaz accesible mediante una petición *HTTP* a una *API REST*, intercambiando datos en formato *JSON*. De hecho, a pesar de que BigML tiene una interfaz web que hace posible crear flujos complejos mediante un solo click, esto no es más que una capa gráfica por encima de llamadas a la misma *API* que se ofrece públicamente a sistemas externos. Esto permite la integración con cualquier sistema que sea capaz de crear una petición *HTTP*, y esto la hace accesible hoy en día desde cualquier tecnología moderna.

En BigML, todos los elementos de un flujo de *Machine Learning* son recursos, incluido el propio flujo y su ejecución. Además, los recursos son inmutables y se pueden inspeccionar de forma sencilla, y exportar. Esto hace que la auditoría de la aplicación sea extremadamente sencilla y automática. Todos los recursos guardan una referencia al recurso que lo ha creado, de forma que siguiendo la cadena se pueda seguir la historia de cada uno de los pasos que se ha seguido para llegar al resultado, repetirlo si es necesario o generar un informe, que la inmutabilidad asegura que represente el estado en el que se encontraba cada recurso en cada paso del proceso.

Aunque una *API REST* es universal, en muchas ocasiones no es lo más cómodo de tratar. En casos de uso interactivo, la mejor opción es el uso de la interfaz web, que ofrece herramientas para la exploración de datos (ver Figura 2.2) y creación e interpretación de modelos (ver Figura 2.3), y creación workflows complejos con pocos clicks.

También, a la hora de automatizar las tareas e integrarlas con otros sistemas, interactuar directamente con la *API* no es la mejor experiencia de usuario. Para solucionar este problema, existen librerías en la mayoría de los lenguajes más populares que abstraen las llamadas *HTTP* en construcciones idiomáticas. De esta forma, se pueden crear flujos enteros en el lenguaje de programación que se elija con pocas líneas de código. Por ejemplo, en *Python*:

```
from bigml.api import BigML

# El usuario y la api key se extraen automaticamente
# del entorno
api = BigML()
source = api.create_source('data.csv')
api.ok(source) # La llamada es asincrona, se espera

dataset = api.create_dataset(source)
api.ok(dataset)

model = api.create_model(dataset)
```

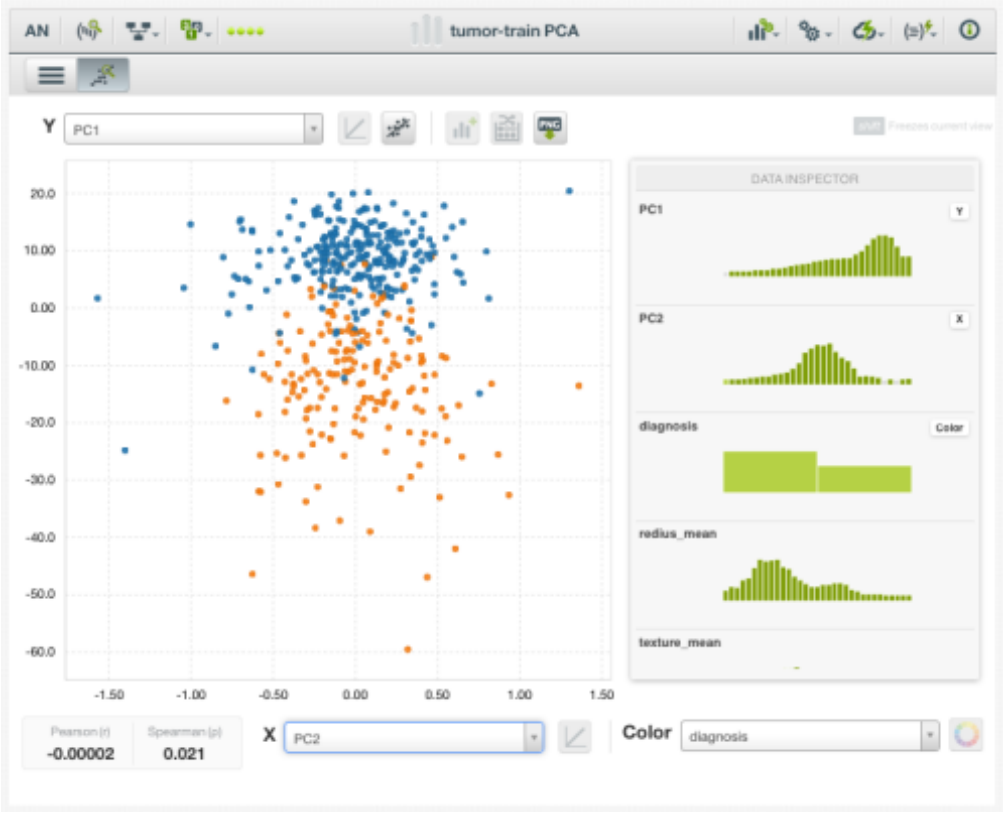

Figura 2.2: Ejemplo de visualización de datos en BigML [12]

```
api.ok(model)

evaluation = api.create_evaluation(model, dataset)
```

Snippet 2.3: Ejemplo de BigML con los bindings en *Python*

En este ejemplo se han cargado unos datos que se encuentran en formato *CSV*, se ha creado un dataset, un modelo y se ha evaluado. Con esa filosofía se pueden hacer flujos arbitrariamente complejos. Como todas las operaciones son asíncronas, es necesario esperar a que su creación termine. Para ello, en los bindings se dispone de la función "ok", que se encarga de hacer *polling* a la plataforma, consultando el estado del recurso hasta que este se encuentre en un estado que indica el fin del proceso, ya sea éxito o error.

A pesar de que con estas herramientas cualquier desarrollador puede trabajar, hay cierta redundancia en el código y tiene el problema de que hay que gestionar el flujo de forma manual. Se puede ver que la mayoría de las operaciones son

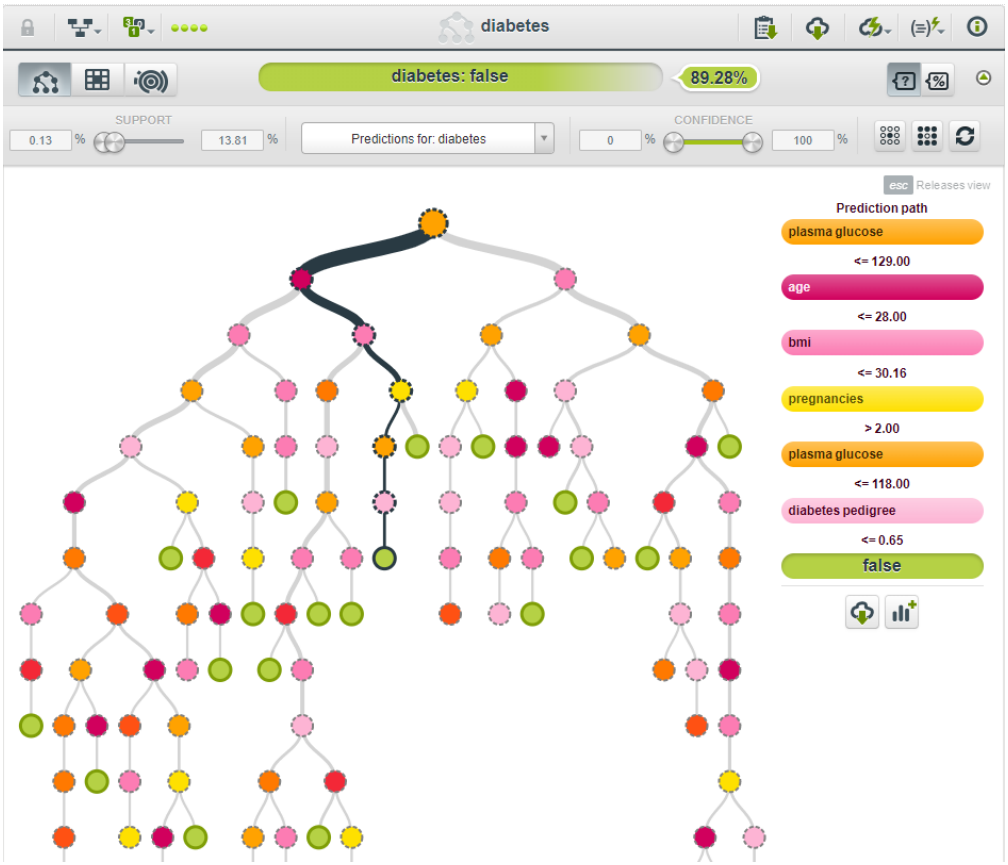

Figura 2.3: Ejemplo de modelo de predicción de diabetes entrenado en BigML

pesadas y pueden llevar tiempo (por ejemplo, subir un dataset de varios GB, entrenar un modelo, etc.), por lo que en un código de cliente es necesario esperar a estas acciones. Además, el flujo de los datos es manual y explícito. Y eso es particularmente evidente cuando se desea introducir paralelismo, ya que queda a discreción del desarrollador la gestión de las tareas, de la sincronización y de los errores.

Para solucionar esto, en BigML se introdujo el concepto de *Script*, que es un programa escrito en *WhizzML*, un *DSL* creado explícitamente para este tipo de tareas, y que ofrece los recursos de BigML como primitivas y las acciones sobre ellas como recursos. De esta forma el mismo flujo se puede expresar de forma concisa, y sobre todo, es un recurso más que es repetible, auditable y exportable, como el resto de recursos en BigML. Por ejemplo, el mismo flujo se puede expresar en *WhizzML* de la siguiente forma:

```
(define dataset
    (create-dataset source))
```

```
(define model
    (create-model dataset))
(define evaluation
    (create-evaluation evaluation))
```

Snippet 2.4: Ejemplo de BigML con *WhizzML*

De la misma forma que con los *bindings*, en *WhizzML* se pueden representar flujos tan complejos como se necesite, con la ventaja de que la propia plataforma se encarga de gestionar paralelismo, sincronización y errores. De esta forma, a lo largo del proyecto los flujos de *Machine Learning* se expresarán en forma de scripts de *WhizzML*.

Una parte muy importante de los flujos de *Machine Learning* es la preparación de los datos e *Ingeniería de Características*. BigML ofrece diferentes primitivas para realizar la mayoría de las transformaciones, ya sea mediante operaciones explicitas en la API, como sería el filtrado de datos, o mediante código *SQL*. Para esta opción, se ofrece también un pequeño *DSL*, llamado *flatline* que permite expresar de forma sencilla transformaciones sobre datasets de BigML.

Además, la plataforma ofrece transformaciones automáticas por defecto, que se aplican cuando se detectan tipos de datos específicos que requieren transformaciones. Por ejemplo, cuando se detecta una fecha, se extrae automáticamente características como año, mes, día, etc., o en el caso de texto, se aplican técnicas de *tokenización*, *lematización*, etc.

Por último, cabe destacar la posibilidad que ofrece BigML de despliegues privados. Una de las razones más comunes para descartar el uso de una plataforma como BigML es la confidencialidad de los datos. Es común que en compañías se traten datos que especialmente sensibles, como datos estratégicos de procesos internos, o datos personales de clientes y empleados. Este tipo de datos exigen tener especial cuidado y control sobre su uso, y dependiendo de la localización, limitaciones legales sobre su almacenamiento y movimiento. Este tipo de limitaciones hace que usar plataformas *Machine Learning* externas a la compañía en cuestión sea muy complejo, o imposible. Esta limitación afecta también al uso de la herramienta principal de *BigML*, *BigML.com*, ya que es un servicio que se da construido sobre la infraestructura de un proveedor *cloud*.

Sin embargo, existe una solución intermedia, mediante la cual se ofrece la solución completa de *BigML* empaquetada para su instalación en un equipo controlado por un cliente, una solución *on premise*. Esta solución no obstante no ofrece algunas de las ventajas de un servicio en la nube, como es la escalabilidad automática, ya que las maquinas físicas habrá que aprovisionarlas de forma manual, pero a cambio nos otorga las ventajas de la plataforma original, con las futuras actualizaciones, mantenimiento y una capa de personalización.

Capítulo 3

Estado de la cuestión

En este capítulo se va a exponer el estado del *Machine Learning* y su adopción en la industria, además de las herramientas, en particular de las relacionadas con *Feature Engineering*. En primer lugar se va a exponer el contexto en el que se encuentra el *Machine Learning* en cuanto a popularidad y adopción, el estado en la industria. Después, se va a poner en contexto la importancia de la *Feature Engineering* en el proceso de *Machine Learning*. Por último, se va a detallar la importancia de las señales como dato, y algunas de las técnicas que existen, y por qué es necesaria una nueva solución.

3.1. Machine Learning y su adopción en la industria

El *Machine Learning* lleva tiempo siendo utilizado como término de moda en la industria desde finales de los años 90, aunque en entornos académicos es un campo que se lleva explorando desde la primera mitad del siglo XX.

La primera mención a las redes neuronales data de 1943, cuando el neurólogo Warren McCulloch y el matemático Walter Pitts presentaron un artículo [13] detallando cómo funcionan las neuronas y un modelo implementado en un circuito eléctrico. A partir de ahí, el campo avanzó rápidamente durante los años 50, 60 y 70, aunque el avance se ralentizó durante los 80 y 90. Aun así, en esta época se hicieron avances que en futuro serían clave para la explosión del *Machine Learning*, como la popularización del uso del *Back Propagation* en las redes neuronales [14]. Sin embargo, el campo empezó a atraer la atención el público general cuando en 1997, el sistema *Deep Blue* de *IBM* venció a Gary Kasparov, campeón del mundo de Ajedrez. Desde este momento, el mundo entero ha estado expectante de los avances en el campo.

En la industria, las compañías tecnológicas más importantes han demostrado

que el uso generalizado del *Machine Learning* a través de todas las ramas de la empresa ofrece una gran ventaja competitiva A día 31 de diciembre de 2019, 5 compañías en las que el uso generalizado de técnicas de analítica avanzada y *Machine Learning* monopolizaban la lista de las 5 compañías con mayor capitalización bursátil (ver Tabla 3.1).

Símbolo	Compañía	Capitalización (M $)
AAPL	Apple Inc.	1,305,000
MSFT	Microsoft	1,203,000
GOOGL	Alphabet Inc.	922,130
AMZN	Amazon.com	916,150
FB	Facebook, Inc.	585,320

Tabla 3.1: Lista de compañías por capitalización bursátil

Según *Gartner* [15], de 2018 a 2019 el número de empresas que tenían algún sistema de "AI" implementado creció del 4 % al 14 %. No obstante, la adopción es lenta, y hoy por hoy la prioridad es utilizar estas herramientas y las técnicas de analítica avanzada como ayuda para extraer conocimiento sobre el negocio y los procesos, y asistir a analistas y directivos a la hora de tomar decisiones. Este fenómeno se puede observar en la Figura 3.1, donde se ve que entre las principales prioridades están los sistemas de *reporting*, *dashboard* o *advanced visualization*.

Por lo tanto, a la hora de crear modelos y aplicaciones predictivas, es importante tener un foco en buscar que los modelos sean interpretables, de forma que se puedan usar para extraer conocimiento de ellos.

Hoy en día la tecnología de vanguardia y la investigación se encuentra en el *Deep Learning*. En este campo se están consiguiendo avances que eran imposibles de prever hace 20 años. Redes convolucionales consiguen mejores resultados que humanos en algunos *benchmark* de clasificación de objetos [17]. Redes masivas basadas en *Transformers* con billones de parámetros como es *GPT-2* [18] son capaces de generar textos que en ocasiones son capaz de engañar a humanos.

Sin embargo, estos modelos y estas arquitecturas son extremadamente complejas, imposibles a día de hoy de interpretar. Además, para funcionar necesitan de cantidades ingentes de datos, que tienen que estar etiquetados, y junto con la complejidad hacen que los entrenamientos sean lentos y caros, al igual que la ejecución del modelo. Es por esto que, hoy por hoy, no se ven modelos de estas características funcionando en tareas fuera del ambiente académico, y los métodos tradicionales de *Machine Learning* son más populares que nunca.

Kaggle, el popular portal de competiciones y contenido educativo relacionado con *Data Science* y *Machine Learning*, realiza anualmente una encuesta entre sus más de 1 millón de usuarios, y en la más reciente (2019), los modelos más populares

Figura 3.1: Importancia de las tecnologías relacionadas con Business Intelligence [16]

eran *Regresión Lineal o Logística* y *Árboles de Decisión y Ensembles* (Figura 3.2) [19] .

Esto demuestra que a día de hoy se siguen utilizando algoritmos que se inventaron décadas atrás, y las innovaciones y mejoras hay que buscarlas en otras partes del proceso de *Machine Learning*.

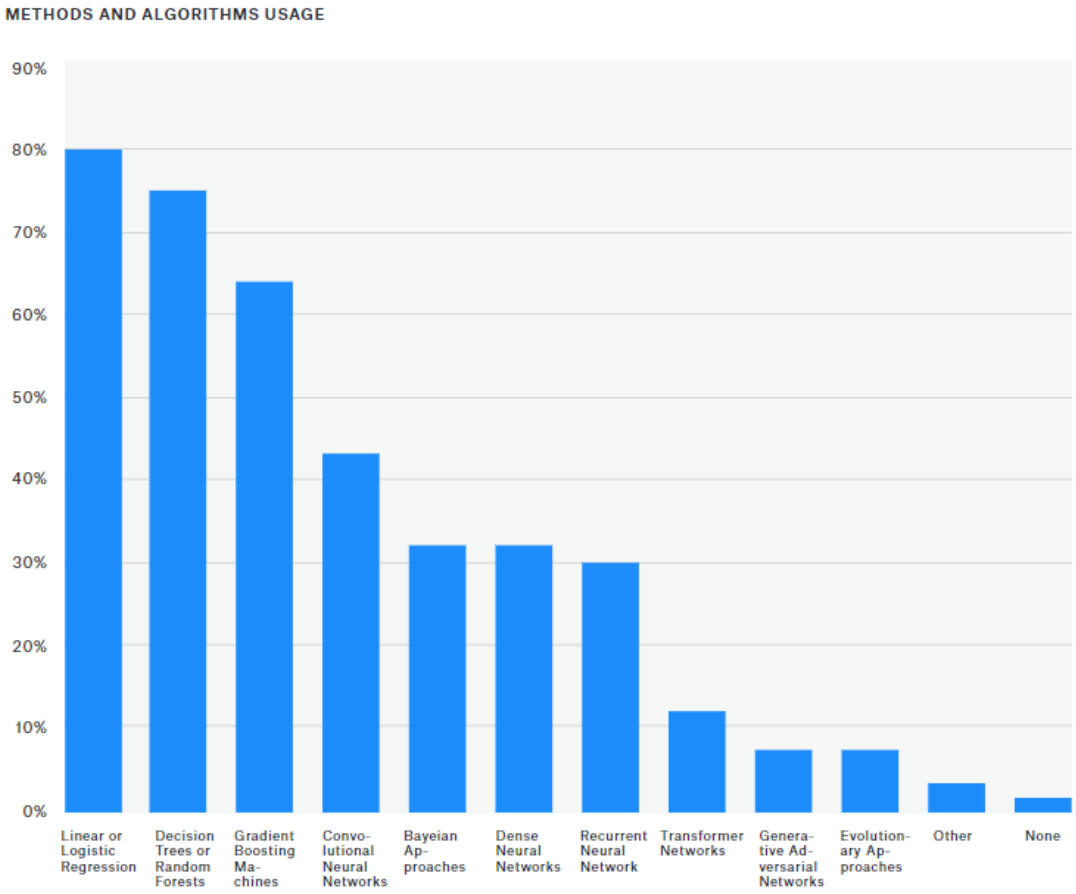

Figura 3.2: Algoritmos más utilizados por los usuarios de Kaggle

3.2. Feature Engineering

Cualquier proyecto de *Feature Engineering* tiene asociadas ciertas tareas que son comunes siempre. El pilar de cualquier proyecto son los datos. Es imposible realizar un modelo que funcione correctamente con unos datos erróneos, mientras que es posible realizar un modelo erróneo con unos datos correctos. De esta forma, sólo con tener unos datos correctos, no es suficiente para conseguir buen modelo, y es necesario comprenderlos y transformarlos para extraer la mayor cantidad de información posible.

Típicamente, el grueso del trabajo y de la complejidad en un proyecto de *Machine Learning* se encuentra en la preparación de los datos, en las tareas de *Extracción*, *Transformación* y *Carga* de datos, o *ETL*. Es poco común que los datos originales se encuentren en un formato óptimo para el modelado. Este fenómeno

se puede observar en la Figura 3.3, en la que se ve que las tareas que más atención requieren de los científicos de datos son las relacionadas con comprender y adaptar los datos.

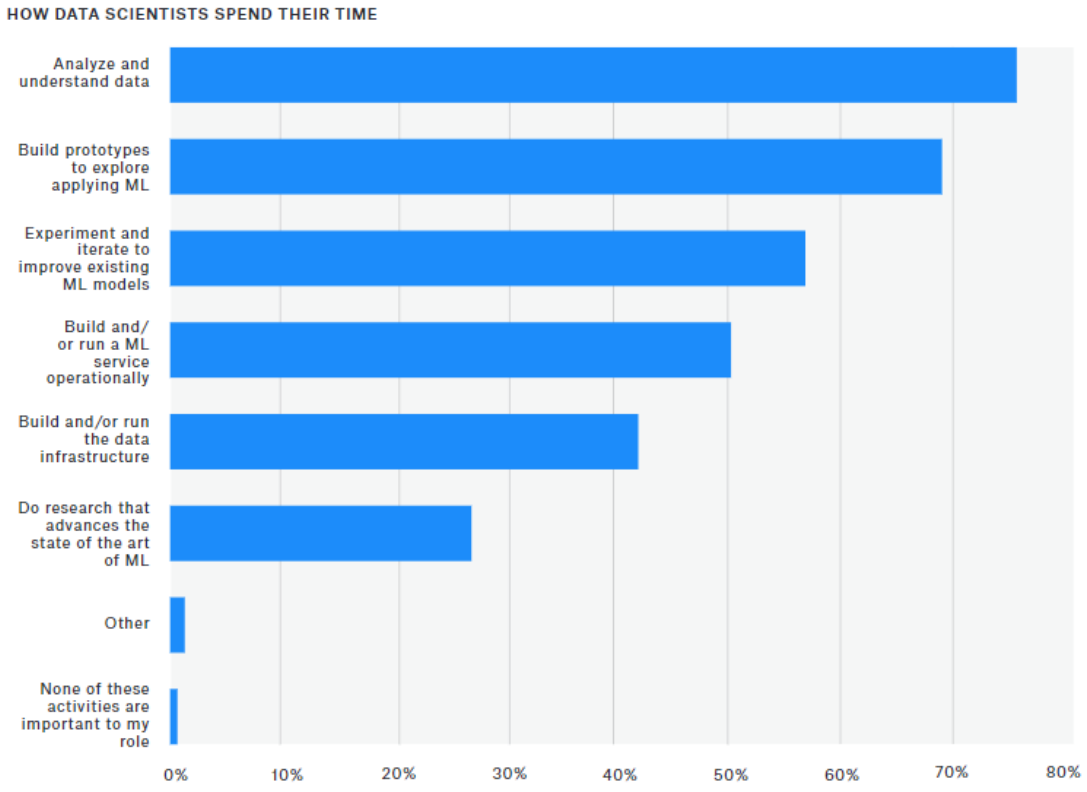

Figura 3.3: Tareas más importantes para los científicos de datos

Los modelos tradicionales de *Machine Learning* funcionan extrayendo patrones de los datos, analizando las relaciones lineales y/o no lineales que existen entre las variables, y haciendo asunciones sobre las distribuciones estadísticas de los mismos. Un ejemplo muy claro es la *Regresión Lineal*, que busca relaciones lineales entre el objetivo y las variables de entrenamiento. Y estas relaciones no siempre son evidentes en los datos originales, y las asunciones no siempre se cumplen.

Otras veces, es necesario transformar los datos ya que se encuentran en formatos que simplemente no son compatibles con los algoritmos. Por ejemplo, algoritmos como la regresión lineal sólo son capaces de procesar datos numéricos, por lo que datos en forma de texto requieren de un paso previo de pre-procesado.

Es por esto que es imprescindible para obtener el mejor resultado posible estudiar y transformar los datos de forma las condiciones sean las óptimas para el modelado en *Machine Learning*. Y este proceso esencial es el que denominamos

Ingeniería de Características o *Feature Engineering*.

Feature Engineering consiste en aplicar el conocimiento específico del dominio de un experto para exponer de forma explícita relaciones en los datos para su modelado. Debido a la naturaleza del problema, el *Feature Engineering* se realiza aplicando métodos y transformaciones que son dependientes del dominio en el que se está trabajando, y el tipo de datos que se dispone.

3.3. Señales

Para el modelado comúnmente se utilizan características numéricas, categorías o texto. Sin embargo, en muchos casos de uso, hay otros tipos de datos que pueden contener información valiosa sobre el dominio que se quiere modelar. En algunos casos, estos datos son la única fuente de información. Uno de estos casos, son las señales.

Según la *ITU* (*International Telecommunication Union*), una señal es un fenómeno físico en el cual pueden variar una o más características para representar información [20]. Las señales se representan como una sucesión de valores o muestras, en una o más dimensiones, que tienen una relación secuencial, es decir, hay información codificada en el orden en el que se encuentran las muestras. La relación más típica suele ser una sucesión de valores en el tiempo, como un voltaje o una corriente, aunque también puede ser una relación espacial, como una imagen. Además, este concepto se puede extender a fenómenos que no son fundamentalmente físicos, como la evolución del precio de un activo.

Las aplicaciones más tradicionales de las señales se han dado en entornos de comunicaciones, donde se usan ondas electromagnéticas (variaciones en el campo eléctrico que se propagan por el espacio) para codificar y transmitir información. Sin embargo, las técnicas de procesado de señales se pueden aplicar con otros fines, como por ejemplo analizar y modelas otros fenómenos que se presentan en esta forma, y esto es particularmente interesante para casos de uso de *Machine Learning*.

Y es que, en el *Machine Learning* se encuentran numerosos casos de uso donde las señales son un tipo de datos que encapsula información que es clave a la hora de modelar y hacer predicciones. Algunos ejemplos de estas señales son la evolución del precio de un activo en aplicaciones financieras, en aplicaciones médicas con electrocardiogramas o encefalogramas, desastres naturales con sismogramas. En la Figura 3.4 se muestra un ejemplo de la señal de un electrocardiograma, que representa la actividad eléctrica (voltaje) en el corazón.

Sin embargo, los modelos tradicionales de *Machine Learning* no son capaces de procesar listas numéricas como dato, y no mantienen la información debida a la naturaleza secuencial de los datos, donde el orden en el que aparecen tam-

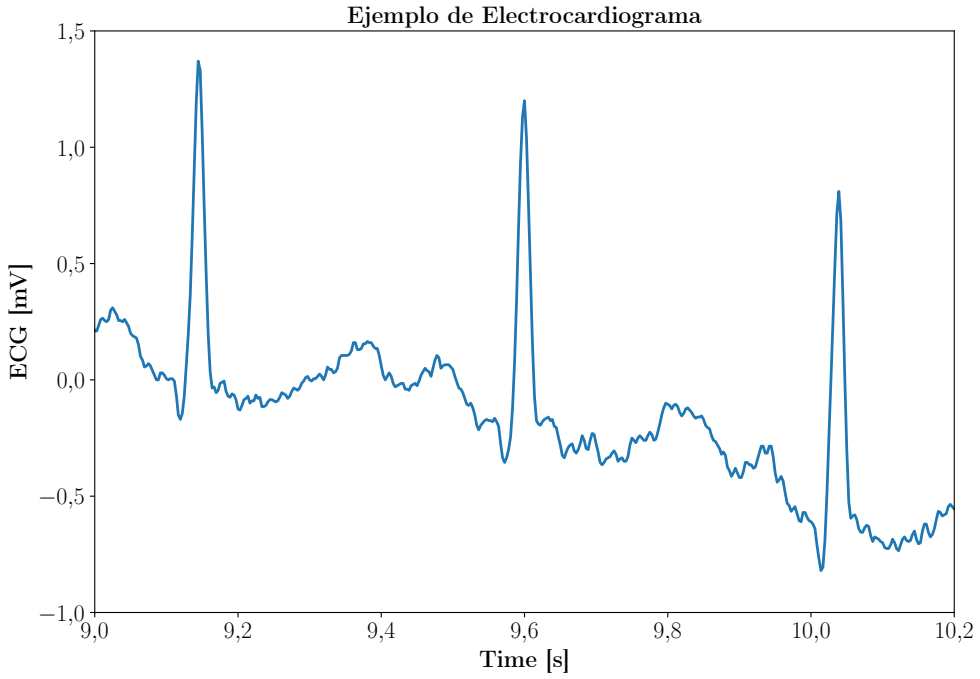

Figura 3.4: Ejemplo de Electrocardiograma

bién aporta información. Por lo tanto, es necesario transformarlos a una serie de características numéricas, intentando mantener la mayor cantidad de información posible.

Esto quiere decir que la fase de *Feature Engineering* es una pieza clave en cualquier proyecto de *Machine Learning* que quiera trabajar con este tipo de datos, ya que no se van a poder aprovechar los datos originales y todas las características de las que se dispongan van a tener que ser generadas o extraídas a base de transformaciones sobre la señal original.

El *Feature Engineering* aplicado a señales no está estandarizado a lo largo de la industria y academia. Este es un campo de investigación abierto, y existen técnicas diferentes que están desarrolladas y especializadas en una aplicación o problema muy concreto.

Las técnicas de *Feature Engineering* sobre señales se centran en hacer transformaciones sobre la señal original, generalmente produciendo nuevas señales que comprimen la información en un número reducido de valores, y utilizar técnicas que permiten localizar estos valores y utilizarlos como características para el modelado. De esta forma se consiguen características numéricas que describen el comportamiento general de la señal, de formas que para modelos tradicionales son más difíciles de descubrir.

Un ejemplo de estas técnicas es el uso de la *Transformada de Fourier*, o la *Densidad Espectral de Potencia* y la extracción de picos sobre estas, para extraer valores que describen una señal y permiten comparar señales, y así implementar algoritmos de clustering o clasificadores.

En la Figura 3.5 se puede ver un ejemplo de una de estas transformadas, la *Short Time Fast Fourier Transform*, aplicada sobre una señal de audio. Como se puede ver, el resultado de esta transformación es otra señal, en este caso una representación 2D de la magnitud de las componentes espectrales de una señal en diferentes ventanas de tiempo. Sobre esta nueva señal generada se aplicarían técnicas como la extracción de picos para conseguir características numéricas que poder utilizar en un modelo.

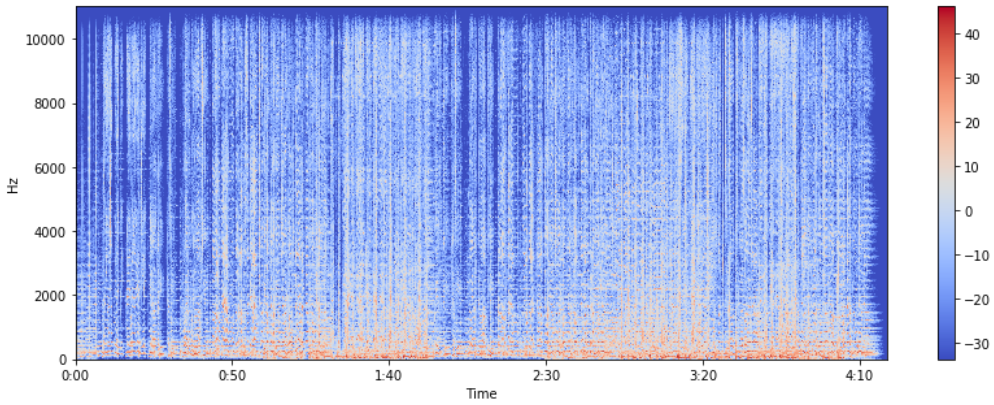

Figura 3.5: Ejemplo de una *Short Time Fast Fourier Transform* [21]

Por ejemplo, en aplicaciones biomédicas, para clasificación de electrocardiogramas y detección de enfermedades coronarias crónicas, las características basadas en la *FFT* (*Transformada de Fourier Rápida*) de la señal han sido efectivas [22]. En el campo del reconocimiento de voz, una práctica estándar es calcular los *MFCC* (*Mel Frequency Cepstral Coefficients*) [23], que se basan en dar una representación de la señal basada en la percepción humana. Basada en la *FFT*, la técnica es similar a la anterior mostrada, pero incluyendo unos filtros que adaptan la señal a la escala de Mel, que se adapta mejor a las no linealidades de la escucha humana.

Para la detección y prevención de terremotos, se ha utilizado con éxito otras transformadas como la *Transformada de Wavelets*, tanto en su versión discreta como filtro de ruido, como en su versión continua, a la que se aplican técnicas de extracción de características como la detección de picos, con resultados satisfactorios [24].

Existen decenas de transformaciones y técnicas diferentes, populares en distintos campos del análisis de señales. Sin embargo, no existen herramientas o librerías

estandarizadas que aglutinen estas técnicas en un único lugar dando accesibilidad a un campo ya complicado de por sí.

Matlab ofrece "toolboxes" (equivalente a una librería en otros entornos), como la *Signal Processing Toolbox* (ver Figura 3.6), o el *Audio Toolbox*, que contienen eficientes implementaciones de estas transformadas, aunque se encuentran detrás de una barrera de pago que las hacen de difícil acceso.

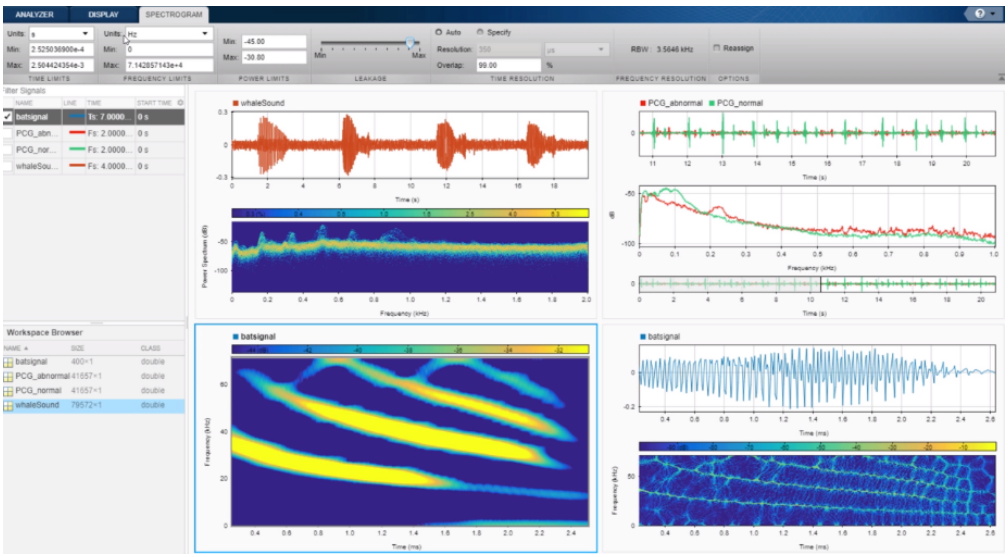

Figura 3.6: Signal Processing Toolbox de Matlab [25]

En esta línea, para el lenguaje *Python*, el más popular entre los científicos de datos, existen algunas librerías que cumplen esta función, pero, al igual que resto de soluciones, están diseñadas con aplicaciones específicas en mente.

Por un lado, la más popular es la librería *tsfresh* [26], centrada en el análisis de series temporales, que agrupa algunas de las técnicas mencionadas, así como otros análisis estadísticos más tradicionales, como media, mediana, cuartiles, entropía, energía, etc. Esta librería es capaz de generar más de 200 características diferentes para una misma serie temporal, y tiene buena integración con *Scikit Learn*, el estándar de *Machine Learning* en *Python*.

Por otro lado, se encuentra *LibROSA* [27], que está especializada en análisis de audio y música, incluyendo las técnicas como *MFCC*, análisis de espectrogramas, y otras como tempogramas, centroides de Tonnetz, y más técnicas avanzadas para el análisis musical.

Aunque son soluciones completas y que han demostrado ser de utilidad, este tipo de librerías tienen varios problemas que las hace no deseables como herramientas generales para procesar cualquier tipo de señal.

En primer lugar, están pensadas con aplicaciones concretas en mente. Por un

lado, esto les permite obtener mejores resultados, ya que pueden hacer asunciones sobre los datos que permiten incluir transformaciones más específicas, pero por otro lado esto hace que no generalicen bien a otros tipos de problemas.

En segundo lugar, son complejas de utilizar y de interpretar. Al ser transformaciones muy específicas, requieren de conocimientos muy concretos de técnicas de procesamiento digital de señales, y en el caso de LibROSA, de procesamiento con fines musicales para poder sacar partido a la librería.

Por último, debido a la naturaleza de muchas de algunas de estas transformaciones, el tiempo que requieren para procesar una señal puede ser elevado, que para casos de uso donde el tiempo o la capacidad de cómputo es una limitación que las haga impracticables. Un ejemplo de este tipo de casos son soluciones IoT de mantenimiento predictivo o detección de errores, donde se están generando grandes cantidades de datos y es necesario procesarlos en casi tiempo real, y no son aceptables grandes retrasos en el tratamiento de los datos.

No existe una solución estándar que permita procesar señales, sin importar el tipo de señales que sean ni el caso de uso, y que al mismo tiempo ofrezca una interfaz sencilla que se acerque a un usuario no experto en este campo.

Capítulo 4

Descripción de la solución

En este capítulo se va a detallar una descripción del modelo. En primer lugar se va a exponer la motivación por la que se ha decidido invertir tiempo en este proyecto. Después se va a comentar la metodología utilizada, basada en metodologías ágiles con trabajo remoto, y el uso de *benchmarking* y datos para tomar algunas de las decisiones a lo largo del desarrollo.

4.1. Motivación

A pesar de existir una gran cantidad de información, con artículos, blog posts y librerías, incluir señales como parte de un modelo de *Aprendizaje Automático* sigue siendo una tarea ardua que exige un trabajo de investigación importante, que se puede hacer inmanejable si no se disponen de conocimientos previos de procesamiento de señales.

BigML es una empresa cuya misión es hacer el *Machine Learning* accesible a todo el mundo (ver Subsubsección 2.3). Siguiendo esta visión, su plataforma online es de acceso gratuito, es muy fácil de utilizar y ofrecen una gran cantidad de cursos, artículos y videos introductorios. En la plataforma se busca que un flujo completo de *Machine Learning* se pueda hacer con apenas unos clicks, aunque para usuarios avanzados siempre está disponible la completa configuración del proceso. Para poder ofrecer esto, es necesario automatizar muchas partes del proceso, y ofrecer unos valores por defecto que funcionen bien en la mayoría de los casos.

Entre las secciones automatizadas, se incluye la transformación de datos, y algunas tareas básicas de *Feature Engineering*. En función de los tipos de datos descubiertos, se realizan transformaciones y se extraen características relevantes de forma automática. Por ejemplo, si se detecta una fecha, se extrae el año, mes, día, hora, día de la semana y más y se añaden como características a los datos originales (ver Figura 4.1).

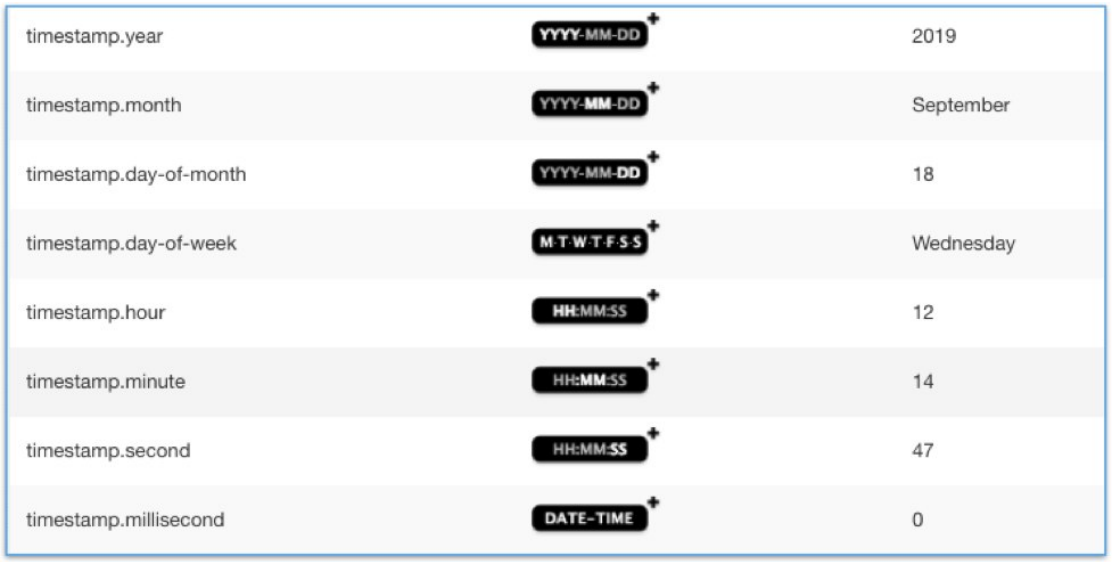

Figura 4.1: Ejemplo de *Feature Engineering* automático sobre fechas en BigML

Lo mismo se hace con campos textuales, donde se detecta el lenguaje, se aplican técnicas de *tokenización*, eliminar *stop words*, *stemming*, *n-gramas*, y más técnicas estándar de procesamiento textual, todo sin necesidad de interacción por parte del usuario. Todas estas técnicas permiten por un lado ofrecer unos valores por defecto razonables, al mismo tiempo que exponen configuración para que usuarios avanzados puedan afinar el comportamiento de las transformaciones a su caso específico.

Siguiendo esta filosofía, se quiere desarrollar una solución de *Feature Engineering* para señales que ofrezca esta combinación de facilidad de uso, sin perder la opción de configuración si se desea. Esta herramienta permitirá que cualquiera, sin necesidad de tener conocimientos específicos sobre señales o los pasos necesarios para su preparación para el modelado.

De esta forma se consigue que el *Machine Learning* sea accesible a un público más amplio. Con esto se consigue que se exploren nuevos problemas, o problemas conocidos con nuevas visiones traídas por personas con perfiles para los que antes el *Feature Engineering* tenía barreras demasiado grandes.

4.2. Objetivos

Como se ha comentado en la sección anterior, el objetivo es hacer una herramienta que abstraiga al máximo nivel posible las tareas de *Ingeniería de Características* sobre señales. En particular, se busca facilitar el modelado sobre eventos

que pueden contener datos tradicionales, que se enriquecen con características de señales asociadas a dichos eventos. Para ello, la solución considerada es una librería, que es la opción que ofrece más flexibilidad a la hora de integrar esta herramienta en soluciones más completas, como puede ser una solución específica, como es el caso de uso explicado en este proyecto, o soluciones más generales que aumenten el alcance de la herramienta, como la plataforma de BigML.

Con este proyecto se busca implementar una prueba de concepto de esta librería, demostrando su utilidad en una variedad de aplicaciones diferentes, probándolo con una serie de datasets con señales de orígenes variados, que estén bien estudiados.

Al mismo tiempo, esta librería se va a integrar en el proceso de *ETL* de una prueba de concepto de un proyecto real de mantenimiento predictivo, procesando señales provenientes de soldaduras en un proceso de fabricación de automóviles.

Con el fin de guiar el diseño e implementación del proyecto, se han definido una serie de objetivos que buscan dar solución a las necesidades que se han identificado, y reflejan la filosofía final del proyecto y de BigML, que es hacer el *Machine Learning* accesible a todo el mundo. A continuación se detallan cuáles son estos objetivos, ordenados por importancia. Este orden es importante a la hora de tomar decisiones que conlleven priorizar entre uno y otro objetivo.

- Utilidad: la librería debe devolver unas características que sean útiles, que aporten valor y que justifiquen el uso de esta solución sobre otras
- Usabilidad: la librería tiene que ser fácilmente utilizable, sin requerir conocimientos previos sobre el dominio o procesamiento de señales
- Eficiencia: las transformaciones y técnicas de extracción que se utilicen no deben ser demasiado pesadas desde un punto de vista computacional, de forma que se consigan resultados en poco tiempo y se pueda desplegar en todo tipo de dispositivos
- Interpretabilidad: la librería debe priorizar transformaciones que permitan en análisis e interpretación de los resultados
- Permitir configuración: la librería debe exponer al usuario avanzado parámetros que configuración que le permita optimizar las transformaciones para adaptarlas al caso de uso concreto
- Extensibilidad: la librería debe estar implementada de forma que sea fácilmente extensible, permitiendo introducir innovaciones y mejoras rápidamente

- Rápidez de desarrollo: el desarrollo debe ser ágil, para poder obtener una prueba de concepto lo antes posible, detectando problemas en el diseño y posibles mejoras, y ofreciendo utilidad en el caso de uso

Estos objetivos guiarán las decisiones tomadas a lo largo del desarrollo del proyecto, como las tecnologías, arquitectura del sistema o las transformaciones elegidas.

Por su parte, el caso de uso que se va a implementar tiene como objetivo principal la detección rápida de errores en un proceso de soldadura, para permitir así solucionar el problema en la misma estación de soldadura, donde es barato y rápido de solucionar. De esta forma, se van a analizar los valores de corriente, tensión y posición de pistolas soldadoras sobre brazos robóticos a lo largo de una soldadura completa.

En caso de que este proyecto funcione, se demostraría que estas técnicas son útiles para este tipo de procesos, y se puede extender el alcance de este proyecto con modelos de mantenimiento predictivo, mediante predicción de fallos en las pistolas de soldaduras.

4.3. Metodología

Como ya se ha comentado anteriormente (ver Sección 2.3), este proyecto se realiza en colaboración con BigML. BigML es una empresa que cuya organización interna se centra en aprovechar las ventajas del trabajo en remoto. De esta forma, el trabajo asociado a este proyecto se va a realizar de forma remota. Esto va a dar forma al resto de la metodología.

El organizar el trabajo de forma remota tiene numerosas ventajas. Algunos ejemplos son:

- Se tiene acceso al talento de cualquier parte del mundo
- La naturaleza asíncrona del trabajo remoto permite a los trabajadores trabajar en las condiciones que les resultan más cómodas, permitiéndoles más productivos
- Se reducen en número de interrupciones innecesarias
- Más flexibilidad a la hora de organizar reuniones, ya que no hay limitaciones geográficas
- Puede haber gente disponible 24 horas al día

Sin embargo, esta organización presenta algunos retos:

- La comunicación es más complicada, y esto puede presentar problemas a la hora de organizarse
- Los equipos muy distribuidos, encontrándose en husos horarios muy diferentes, pueden tener problemas para encontrar momentos para coincidir.
- Exigen disciplina en los trabajadores para mantener la concentración y productividad, ya que las distracciones pueden ser mayores y hay menos presión
- Su éxito requiere del uso de herramientas
- Hay algunos trabajos que por su naturaleza es imposible realizarlos de forma remota, como los que conllevan contacto directo con el cliente

El éxito de una organización basada en el trabajo remoto requiere del uso de herramientas. Un proyecto de *Machine Learning* es un candidato idóneo para este tipo de organización, ya que las tareas se son fácilmente divisibles, los empleados naturalmente dominan la tecnología, y la colaboración remota es un problema que está bien resuelta con herramientas como *Git* y *Github*.

Desde un punto de vista de organización y comunicación, herramientas como *Slack* permite que los empleados puedan comunicarse de forma fluida. *Google Meet* o *GotoMeeting* permiten mantener reuniones virtuales, y junto con herramientas como *Google Docs* y *Google Sheets*, que permiten compartir archivos, gestionar documentación y editarla de forma colaborativa, se facilita enormemente la coordinación y administración. Es importante coordinar con los posibles clientes para que las dos partes se puedan adaptar y ponerse de acuerdo en la mejor forma de organización para un proyecto.

Además del trabajo remoto, otro de los requisitos de este proyecto es ser capaces de producir un producto utilizable lo antes posible, para poder realizar evaluaciones y mejoras en función de los resultados obtenidos, y poder avanzar en paralelo con el caso de uso. Es por esto que se ha elegido seguir una organización basada en las metodologías ágiles, como *Scrum* o *Extreme Programming*.

Se va a realizar una planificación basada en *sprints* de entre 1 y 3 semanas de duración, dependiendo de las tareas, con reuniones semanales, tanto internas como con el cliente. La función de estas reuniones semanales es triple:

- Hacer un seguimiento del proyecto, que permite dar una estimación real del avance del proyecto y gestionar expectativas, y planificar los siguientes pasos
- Conseguir feedback sobre el producto, que permite corregir desviaciones en los requisitos y las funcionalidades implementadas lo antes posible para reducir al máximo posible el tiempo perdido debido a malentendidos o errores en la comunicación

- Involucrar a los *stakeholders* en el desarrollo del proyecto, asignando tareas que ellos pueden hacer para sentir que forman una parte activa del desarrollo del producto, lo que aumentará su apreciación por el mismo

El desarrollo de la librería sigue la misma organización, ya que su avance está estrechamente relacionado con el avance del caso de uso. Además, aprovechando el hecho de que se tiene acceso a una gran cantidad de datos de un caso de uso real, se van a emplear estos datos para hacer evaluaciones y pruebas sobre las diferentes técnicas en cuanto a procesamiento de señales. De esta forma, se asegura que el conjunto de transformaciones elegidas para la prueba de concepto tiene aplicabilidad.

Las tareas principales se corresponden con los objetivos mencionados en el apartado anterior, y se van a realizar en el mismo orden en el que se han presentado. Sin embargo, el proceso no es lineal, y nuevos requisitos en el caso de uso, así como los resultados de las evaluaciones en el caso y en los benchmark harán que algunas de las características elegidas en la primera fase tengan que ser revaluadas, y esto inducirá cambios en el producto final.

Las decisiones tomadas durante el transcurso del proyecto, sobre la elección de las transformaciones, técnicas de extracción e implementaciones concretas, se van a basar datos tomados como resultado de hacer pruebas y experimentos. A la hora de tomar todas las decisiones, se van a tener en cuenta los beneficios que ofrecen, junto con los costes que conllevan, y tener en cuenta el equilibrio entre sus aportaciones a los distintos objetivos, ponderados en función de la importancia que se le da a cada uno.

Para automatizar este proceso, y como técnica de evaluación final, se va a desarrollar un benchmark estandarizado que tome medidas sobre el comportamiento de la librería y permita comparar distintas opciones y configuraciones, para a así poder elegir la que de forma empírica ofrezca mejores resultados. Para ello, se van a utilizar un repositorio de datasets (ver Subsección 5.3.4 para más información) que contiene 128 conjuntos de datos con señales como entradas, con una división ya hecha entre train y test. El proceso se aplica en el benchmark a cada dataset del repositorio es el siguiente:

1. Utilizar la librería para transformar los datos de entrenamiento y de test
2. Entrenar un modelo de *ensemble* (*Random Forest*) con el conjunto de entrenamiento
3. Evaluar el modelo con el conjunto de test
4. Extraer métricas sobre la evaluación

5. Extraer métricas sobre la importancia de los campos relacionados con cada transformación

Con los resultados de cada una de las evaluaciones, se van a agregar y analizar su distribución, y son estos estadísticos generados los que se van a utilizar para tomar decisiones sobre las mejores transformaciones y características a elegir.

Esta es la metodología elegida ya que es el método utilizado por otras implementaciones anteriores que han realizado proyectos similares, y esto permite comparar mejor las distintas implementaciones. No obstante, cabe destacar que el utilizar una metodología basada en una única división entre train y test tiene desventajas aparentes:

- Solo se tiene una muestra de una evaluación por dataset, por lo que se incurre en una varianza grande a la hora de estimar los resultados reales
- Al no tener otra fuente de datos para evaluar, el utilizar estos resultados del test para tomar decisiones puede incurrir en *overfitting* en la librería a favor de buenos resultados en este benchmark
- Además, en este caso concreto, no se sabe cómo se ha realizado la división, por lo que puede haber algún problema o sesgo en la propia división que falsee resultados

Estas son críticas válidas para esta metodología. Sin embargo, tiene la ventaja de que permite realizar una comparación con otras implementaciones y obtener resultados rápido, que es un requisito importante en el proyecto. Además, al combinar los resultados de 128 evaluaciones con 128 datasets distintos al mismo tiempo, esta variabilidad en los resultados se verá reducida.

Finalmente, es cierto que se corre un riesgo de *overfitting* con esta metodología, por lo que se propone como una mejora y un trabajo futuro el realizar una validación del diseño pero utilizando una metodología utilizando técnicas más aceptadas. En particular, se propone el uso de un conjunto de validación, que sólo se utilice para una prueba de la librería final, y que se utilicen técnicas de *Cross Validation* para tomar las decisiones intermedias sobre las transformadas a elegir. De esta forma se consiguen mejores estimaciones de la capacidad de generalización de los modelos, y por tanto de las transformaciones, y con esto se consigue una evaluación más realista del proyecto.

4.4. Planificación

En la Figura 4.2 se muestra un *Diagrama de Gannt* que ilustra la planificación inicial, donde se pueden observar los rasgos de la metodología detallada en el apartado anterior (Sección 4.3).

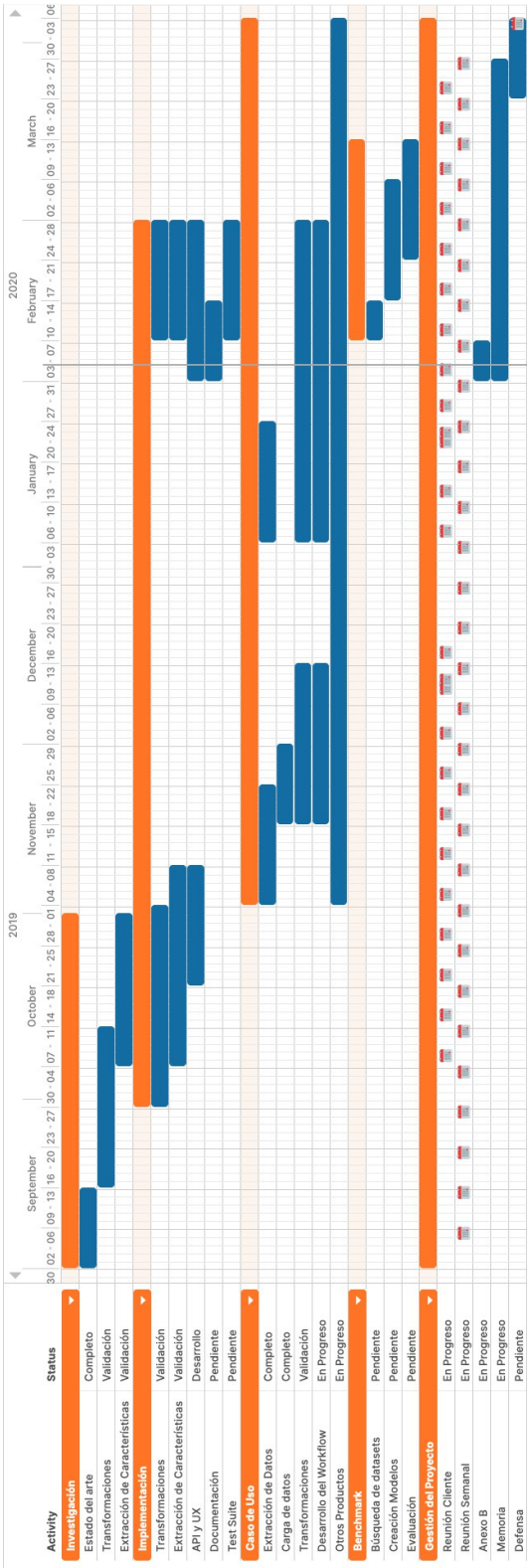

Figura 4.2: Diagrama de Gaunt con la planificación inicial

Debido a la naturaleza de la metodología, basada en una organización ágil e iterativa, es de esperar que se produzcan adaptaciones de las tareas y los tiempos a lo largo del proyecto.

Siguiendo esta organización, el proceso de desarrollo sigue una estructura definida. En primer lugar, se descubre una necesidad. En una reunión, se definen unos requisitos y se discuten posibles soluciones y funcionalidades que respondan a esta necesidad.

Tras un proceso de diseño, el que puede conllevar una revisión profunda del estado del diseño actual y de cómo encaja la nueva funcionalidad, se realiza una primera implementación. La implementación se revisa a través de un proceso de *code review*, por el que un desarrollador que no ha intervenido en la implementación directamente revisa el código, y esto puede conllevar sugerencias de cambios y mejoras.

Una vez incluidos estos cambios, y previa aprobación de los revisores, se incluye la nueva funcionalidad en el producto, se prueba, y en la siguiente reunión de seguimiento se enseña al cliente, que dará la última aprobación, o nuevas sugerencias de mejora. Este proceso se ilustra en la Figura 4.3.

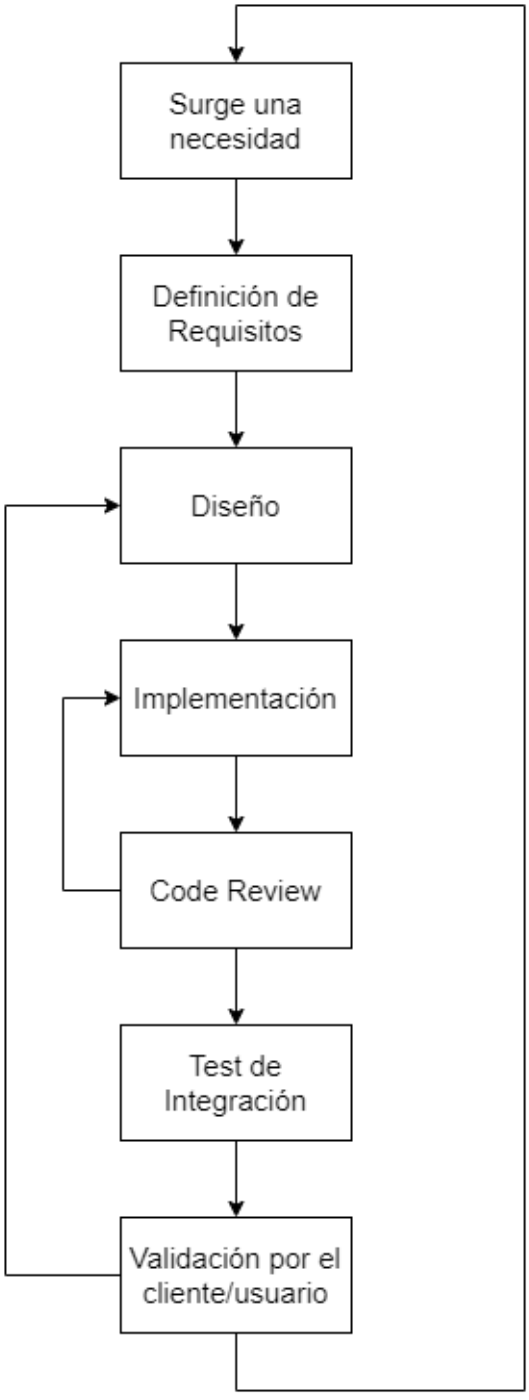

Figura 4.3: Proceso iterativo de desarrollo

Capítulo 5

Sistema desarrollado

En este capítulo se va a hacer una descripción del sistema desarrollado. En primer lugar, se va a realizar un análisis del sistema, desde el punto de vista de la arquitectura y el diseño de más alto nivel. Después, se va a explicar el diseño del sistema, comentando los objetivos y los requisitos de la librería. Por último, se van a exponer algunos elementos considerados destacables sobre el proceso de implementación del sistema, comentando técnicas utilizadas para cumplir con éxito los objetivos marcados.

5.1. Análisis del sistema

Al más alto nivel de abstracción, el objetivo de este proyecto es desarrollar una librería que permita extraer características numéricas de una señal, que es un tipo de datos como se ha descrito en Sección 3.3. Esto es, una librería que permita hacer *Feature Engineering* sobre señales.

Más concretamente, a la entrada se recibe una secuencia de valores numéricos, y a la salida se va a devolver un conjunto de valores numéricos con nombres asociados, correspondientes a las características generadas para describir la señal de entrada. Gráficamente, esto se ilustra en la Figura 5.1.

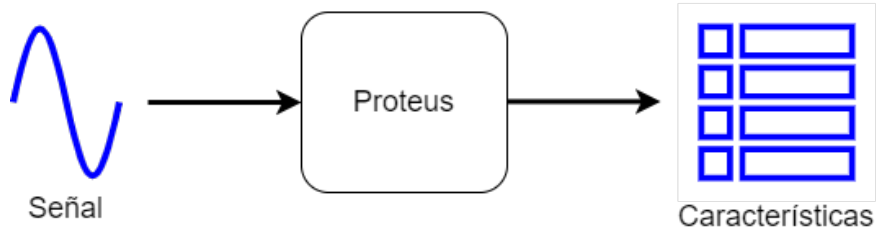

Figura 5.1: Descripción de alto nivel de la librería

La librería expone una función principal que transforma una lista de N valores en un conjunto de K características. Esta función se puede ver como una función matemática que establece una relación entre una lista numérica y sus características, donde para una misma entrada se obtiene siempre la misma salida, es decir, es determinista.

Para hacer *Feature Engineering* sobre señales, se ha seguido un procedimiento similar al que se siguen otras soluciones existentes. Este proceso se basa en hacer transformaciones sobre la señal original, y aplicar técnicas de extracción sobre estas transformaciones con el objetivo de concentrar la mayor cantidad de información posible en el menor número de características. Este es un proceso intuitivo y que ya ha sido probado para soluciones específicas y librerías más genéricas.

El primer paso es realizar una transformación sobre la señal original. Esto es, aplicar una función matemática (que generalmente es no lineal) sobre la señal original, para obtener una representación de la señal diferente. El objetivo de este paso es buscar distintas representaciones de la señal original, que expliciten ciertos rasgos característicos de la misma de forma que sea fácil extraerlos en el siguiente paso.

Este concepto es análogo a lo que se busca al realizar *Feature Engineering* sobre cualquier otro tipo de datos, y la única diferencia son los algoritmos concretos que se utilizan. Además, muchas de las transformaciones que se aplican sobre señales generan otras señales a la salida, y por lo tanto es necesario añadir una operación más para resumir esta nueva señal en unos pocos valores numéricos representativos.

Y este es el objetivo del siguiente paso del proceso, que representa la aplicación de técnicas que resumen una distribución de datos en unos pocos valores que tengan alguna propiedad o representen una propiedad de la propia distribución. Un ejemplo sencillo de una de estas técnicas es el cálculo de estadísticos, como máximo, media, mediana, etc. que representan propiedades estadísticas de la distribución de las muestras de la señal o de una transformación. Las transformadas se explican más en detalle en la Subsección 5.2.3, y técnicas de extracción en la Subsección 5.2.4.

La arquitectura del sistema refleja este proceso, y su objetivo es hacer independientes los dos procesos, lo que permite diseñarlos, desarrollarlos y modificarlos de forma independiente. La arquitectura se muestra en la Figura 5.2.

En la siguiente sección (Sección 5.2) se explican en detalle los componentes y el proceso de diseño.

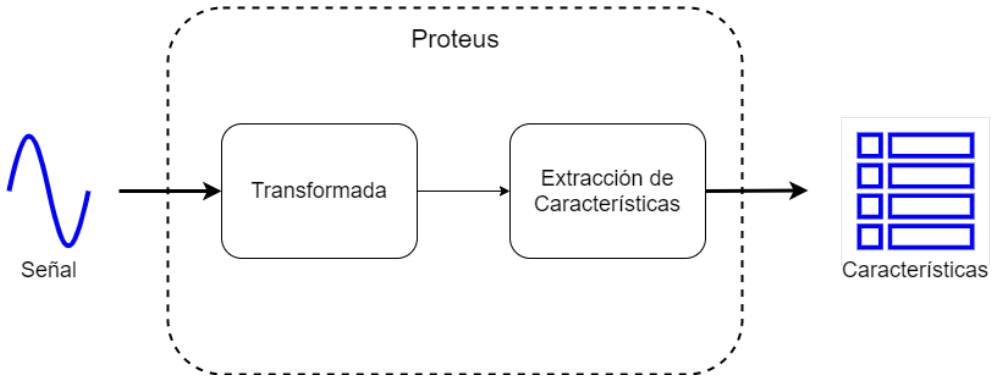

Figura 5.2: Arquitectura de la librería

5.2. Diseño

En esta sección se va a explicar más en detalle el proceso descrito en el apartado anterior, y el diseño que se ha elegido.

Para ello, se va a comenzar explicando los requisitos del proyecto. Después se van a comentar los distintos componentes que componen el sistema, sus relaciones, y cómo reflejan el proceso que se ha descrito en la arquitectura. Aquí se van a detallar las transformaciones y algoritmos de extracción elegidos para la librería. Por último, se van a exponer algunos detalles relevantes sobre la implementación concreta de una prueba de concepto del diseño explicado en los otros dos apartados.

5.2.1. Requisitos

Los requisitos están estrechamente relacionados con los objetivos detallados en la Sección 4.2.

El requisito principal de la aplicación tiene que ver con maximizar la utilidad de la librería. Es necesario que esta solución aporte valor a los usuarios, y que se justifique el coste en tiempo, y por lo tanto dinero, que conlleva realizar las operaciones. Esta librería puede ser utilizada en aplicaciones que pueden tener un gran impacto en la sociedad, como en aplicaciones médicas, prevención de desastres, control de calidad de productos, etc. y como tal, se debe buscar maximizar la utilidad de la misma.

Este requisito se va a medir con relación a los objetivos del caso de uso, es decir, que se medirán por la capacidad de detectar errores, y la importancia asociada a las características generadas por esta librería durante el proceso de *Feature Engineering*.

A la hora de buscar la utilidad, se busca también que se generalice a distintos tipos de aplicaciones. Se debe evitar optimizar la estructura de la librería con un

único caso de uso en mente.

Para hacer una estimación de la utilidad de la librería en una o varias aplicaciones, se va a crear un benchmark, que permite aplicar las transformaciones de la librería sobre un conjunto de más de 100 datasets provenientes de distintos orígenes, con distintos objetivos, cuyo objetivo es una tarea de clasificación (ver Subsección 5.3.4).

Para cada conjunto de datos, se va a realizar el *Feature Engineering* con una configuración determinada de la librería, se va a crear un modelo de *Random Forest* en BigML, se va a evaluar el modelo con un conjunto de test determinista, y siempre el mismo, y se van a extraer parámetros de la evaluación de la clasificación. Al final del benchmark, se extraen estadísticos que describen la distribución de los resultados de las evaluaciones, y estos serán los valores que se usen para comparar.

Este sistema se va a utilizar también para medir importancias de transformaciones, y así tomar decisiones sobre la utilidad de una transformación concreta.

Relacionado con el objetivo de la utilidad, se debe permitir que un usuario avanzado o experto en un dominio concreto pueda utilizar sus conocimientos para configurar y optimizar el comportamiento de las transformaciones. De esta forma, se debe ofrecer una forma de configurar el sistema.

El segundo objetivo importante en el sistema es la usabilidad. El sistema tiene que ofrecer su utilidad de la forma más accesible posible. Para ello, el sistema debe tener incorporados unos valores por defecto, de forma que sin ningún tipo de esfuerzo se puedan conseguir resultados.

En esta misma línea, se debe intentar que los usuarios puedan interpretar los resultados de la librería. Para conseguir esto, se va a priorizar transformaciones sencillas e intuitivas sobre otras más complejas, siempre sin perder de vista la utilidad. Sin embargo, debido a la naturaleza compleja que conlleva el extraer relaciones de secuencias numéricas, que suelen conllevar transformaciones no lineales, en ocasiones no es posible evitar elegir una transformación intuitiva. Para solventar este problema, se debe añadir documentación que ofrezca la información necesaria, o enlaces directos a la información relevante a las técnicas incluidas en la librería.

Los dos últimos objetivos están relacionados con la agilidad a la hora de desarrollar, tanto el primer prototipo como en el futuro. Así, el diseño debe permitir que el desarrollo del prototipo sea rápido, y al mismo tiempo permita modificaciones y mejoras en un futuro sin necesidad de hacer cambios grandes, y especialmente evitar cambios que afecten a la interfaz pública de la librería.

5.2.2. Estructura

A la hora de realizar el diseño del sistema, se han tenido en cuenta los requisitos antes mencionados.

En primer lugar, se ha decidido realizar un diseño *modular*. Este diseño se basa en dividir el sistema en componentes independientes, con fronteras e interfaces bien definidas. De esta forma, el flujo de datos va de un componente a otro de forma controlada, y se elimina cualquier interdependencia entre los módulos, evitando depender de detalles de implementación entre unos y otros.

Este tipo de diseños ofrecen libertad a la hora de diseñar y desarrollar cada uno de los componentes, y bien implementado permite cambiar por completo la implementación de un módulo, sin necesidad de cambiar la implementación en cualquier otro componente del sistema. Este diseño también permite la reutilización de componentes.

La estructura de los componentes responde a la arquitectura que se ha comentado en la Sección 5.1, es decir, se implementa un flujo que se encarga de procesar la señal a la entrada, realizar una serie transformaciones, y para cada una de ellas, aplicar un algoritmo de extracción de características. Esta estructura se muestra en la Figura 5.3

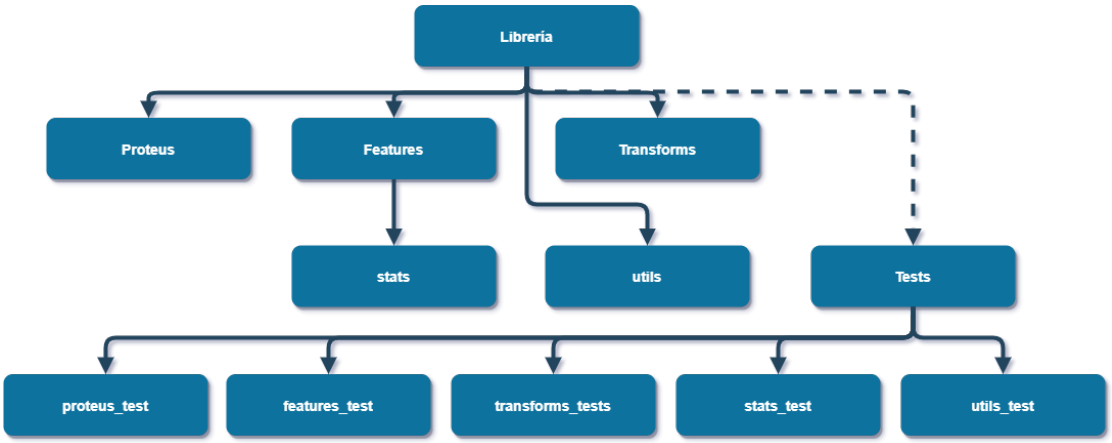

Figura 5.3: Módulos de la librería

La división de los módulos de la librería tiene una estructura que refleja este proceso. Se ha dividido el problema en 4 componentes, que cada uno se encarga de las tareas principales.

En primer lugar, un componente se encarga de procesar la entrada a la librería, y gestiona la configuración y en función de esta, el flujo de los datos. En segundo lugar, otro componente se encarga de realizar las transformaciones. Después, los resultados de las transformaciones se mandan a un componente que se encarga de extraer características de una transformación. Por último, se unen todas las características en una única estructura, que se devuelve como salida al usuario.

La separación de estos componentes permite modificar las técnicas de extracción sin afectar a las transformaciones, y viceversa. A continuación se describe

cada uno de ellos:

- proteus: Es el módulo principal. Este es el módulo que importa el usuario de la librería. Este módulo contiene la función de entrada, cuya función principal es aceptar la entrada, asegurarse de que tiene el formato correcto, y controlar el flujo de datos entre el resto de módulos y funciones.
- transforms: Es el módulo que contiene las funciones que implementan las distintas transformaciones de las señales.
- features: Es el módulo que contiene las implementaciones de las técnicas de extracción de características.
- tests: Es el módulo (o conjunto de módulos) que contiene los distintos tests para los distintos módulos y funciones.
- otros
    - stats: contiene algunas funciones que implementan el calculo de estadísticos.
    - utils: contiene funciones de utilidad en general.
    - plotting: contiene funciones extra para visualizar algunas de las características extraídas por la librería

En la Figura 5.4 se ilustra con un diagrama de actividad cómo interactúan los módulos principales. En ella sólo se ha detallado a bajo nivel cómo se extraen las características relacionadas con el espectro (*FFT* y *PSD*), pero el proceso es similar para cualquier transformación que se añada a la librería.

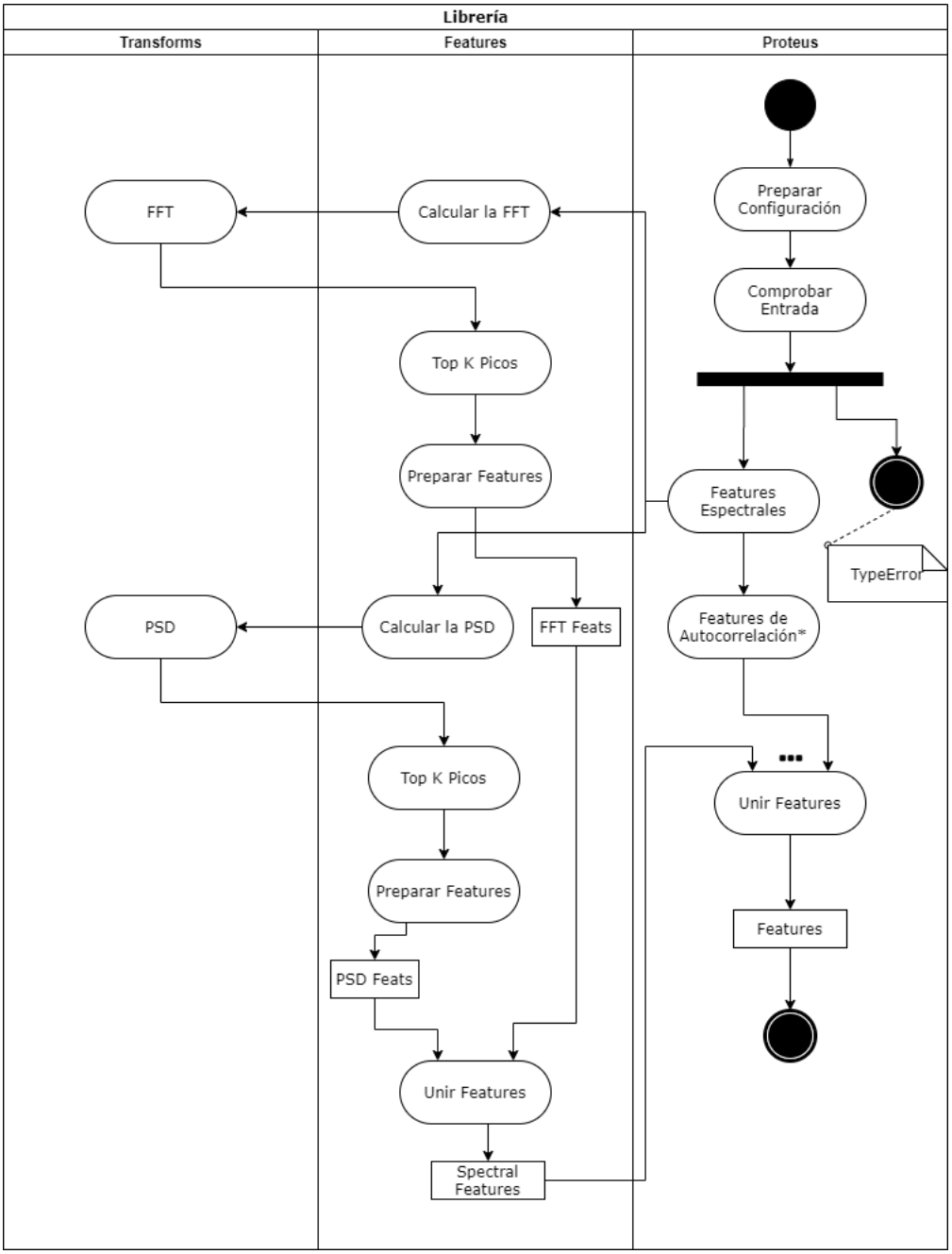

Figura 5.4: Diagrama de Actividad

5.2.3. Transformadas

La elección de las transformaciones correctas es un paso clave que marcarán el éxito o el fracaso del proyecto.

Estas transformaciones consisten en aplicar operaciones matemáticas sobre una señal, de forma que se consigue una representación diferente que resalte alguna propiedad de la misma. El objetivo de estas transformaciones es facilitar el siguiente paso, que es la extracción de características propiamente dicha.

En el campo del procesado de señales tradicionalmente se hace buen uso de este tipo de técnicas, principalmente para codificar y compartir información. Por ejemplo, en aplicaciones de radio como la radio *FM* (*Frecuencia Modulada*) se codifica información de audio en las variaciones de las componentes espectrales (frecuencias) del campo eléctrico, y la forma de extraerlas es mediante el uso de filtros y la *Transformada de Fourier*, que devuelve descomposición de una señal en sus componentes espectrales.

En esta librería se van a aprovechar algunas de las técnicas tradicionales de procesamiento digital de señales para generar características de una señal con el fin de aplicarlas a un workflow de *Machine Learning*.

No obstante, es necesario tener en cuenta los objetivos y requisitos a la hora de elegir las transformaciones que incluir en la librería. Estas operaciones son complejas y muchas de ellas requieren de conocimientos avanzados para poder interpretarlas. Además, aunque algunas de ellas disponen de implementaciones eficientes, en general son operaciones que son costosas de calcular.

Con esto en mente, se van a detallar las transformaciones que se han elegido para la implementación de la prueba de concepto.

Estas son:

- FFT: *Transformada Rápida de Fourier*, da información sobre las principales componentes de frecuencia de la señal
- PSD: *Densidad Espectral de Potencia*, da información sobre la distribución de la energía en las componentes en frecuencia, similar a la *FFT* aunque no idéntica
- ACF: *Función de Autocorrelación*, que da información sobre la correlación de una señal con muestras de sí misma, retrasadas en el tiempo.
- PACF: *Función de Autocorrelación Parcial*, similar a la *ACF*, aunque eliminando relaciones indirectas entre muestras.
- CWT: *Transformada de Ondícula Continua*, que es una generalización sobre la *FFT* que permite analizar el espectro de una señal y su evolución a lo largo del tiempo. Esto es similar a lo que ofrece la *Short Time FFT*, aunque

es más genérica, y tiene una resolución dinámica. De esta transformada se extraen los K mayores picos.

A continuación se van a explicar más fondo cada una de ellas.

Transformada Rápida de Fourier

La *Transformada Rápida de Fourier*, o *FFT*, es un algoritmo que calcula de forma eficiente la *Transformada Discreta de Fourier*, o *DFT*, de una secuencia, que es la versión discreta de la *Transformada de Fourier*. Esta transformada convierte una señal en el dominio del tiempo, donde cada muestra representa un valor concreto en el tiempo, al dominio de la frecuencia, donde cada muestra representa un valor a una frecuencia concreta.

La *Transformada de Fourier* se basa en la propiedad por la que, mediante una combinación lineal de suficientes señales sinusoidales con distintas frecuencias y amplitudes, se puede representar cualquier señal real, sin importar lo compleja que esta sea. A la serie de frecuencias que representan una señal se le denomina el *espectro* de frecuencias. Esto se ilustra en la Figura 5.5.

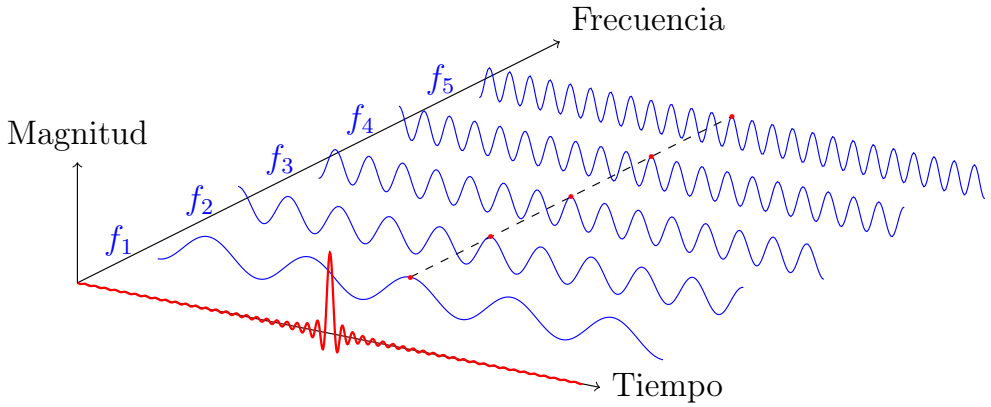

Figura 5.5: Visualización de la *Transformada de Fourier*

Si la señal es periódica, es decir, que se repite tras un periodo de tiempo, se puede representar con un número limitado de componentes, y si no es periódica, se necesitan infinitas componentes espectrales para aproximar la señal perfectamente. Esta transformada es la más utilizada en entornos relacionados con las comunicaciones, y se utiliza para aplicaciones de filtrado, codificación y modulación de señales.

Formalmente, la *Transformada de Fourier* se define por la siguiente ecuación:

$$\hat{f}(\xi) = \int_{-\infty}^{\infty} f(x)\ e^{-2\pi i x \xi}\, dx,$$

Y la *FFT*, que es un algoritmo diseñado para que su implementación, especialmente en hardware, sea muy eficiente:

$$X_k = \sum_{n=0}^{N-1} x_n e^{-i2\pi kn/N} \qquad k = 0, \ldots, N-1,$$

Esta transformación era un candidato ideal, ya que ha sido utilizada con éxito para extraer características en aplicaciones de *Machine Learning*, es muy eficiente de calcular, y al mismo tiempo permite describir señales periódicas complejas con pocos valores. Esta transformada permite la información espectral de la señal y comparar señales (preferentemente estacionarias, es decir, que no varían sus características espectrales en la ventana de tiempo que se analiza) comparando las frecuencias principales de la señal.

En análisis espectral es muy útil, ya que en las componentes espectrales se encuentran información clave de muchos fenómenos físicos, por ejemplo, el timbre y tono de una nota musical o el color de la luz. Así, se puede utilizar esta técnica para detectar y comparar por ejemplo ritmos cardiacos en electrocardiogramas, o detectar una nota musical que se está tocando en un instrumento.

En un caso ideal, en el que por ejemplo la señal de entrada fuera una combinación sencilla de señales sinusoidales, podríamos comprimir toda la información de la señal la localización de las componentes espectrales.

Por ejemplo, con una señal generada con la siguiente ecuación:

$$y(x) = 1 + \sin(2 \times \pi \times 50) + \sin(2 \times \pi \times 100)$$

La señal, y el resultado de aplicar la *FFT* se muestra en la Figura 5.6. En ella se puede ver que la transformada toma valores elevados (1 para la constante, y 1/2 para el resto) en las muestras correspondientes a las frecuencias de las componentes de la señal. Con estos valores, podríamos obtener una descripción perfecta de la señal original. Este es un caso simplificado, y las señales reales tienen más componentes y ruido añadido, pero aquí se muestra la utilidad de la transformada.

Densidad Espectral de Potencia

La *Densidad Espectral de Potencia*, o *PSD*, describe como se distribuye la potencia de una señal en sus distintas componentes espectrales. En un proceso aleatorio estacionario, que es un proceso aleatorio que no cambia sus características estadísticas, se puede definir la *PSD* como la *Transformada de Fourier* de la *Función de Autocorrelación* de la señal. En estos casos, la *PSD* se define mediante la ecuación:

$$S_{xx}(\omega) = \int_{-\infty}^{\infty} R_{xx}(\tau) e^{-i\omega\tau}\, d\tau = \hat{R}_{xx}(\omega)$$

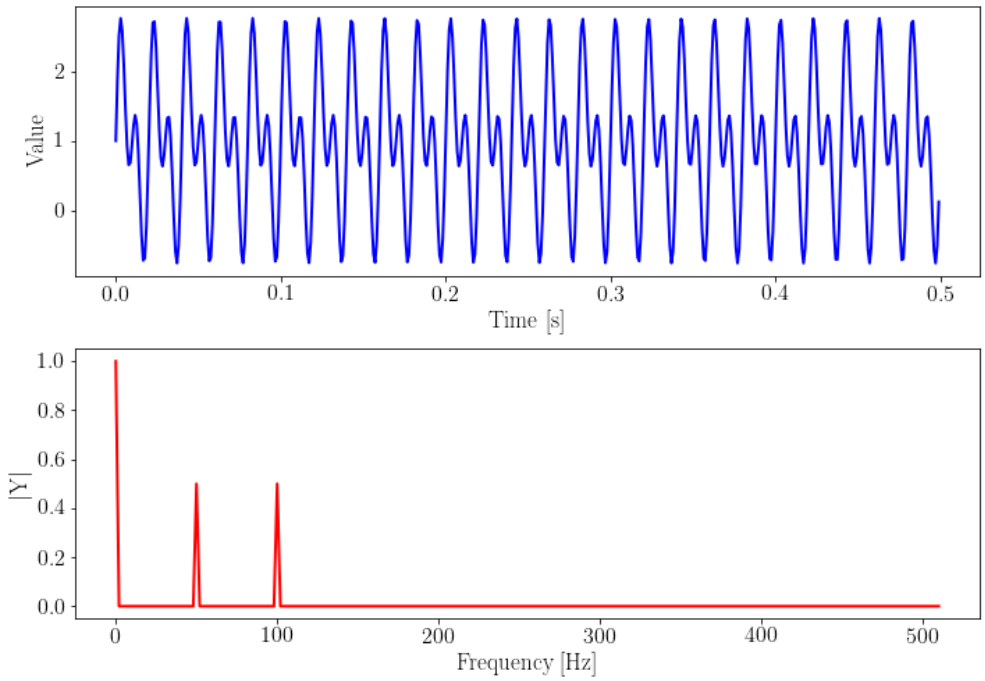

Figura 5.6: Señal de ejemplo y su *FFT*

donde:

$$R_{xx}(\tau) = \langle X(t)X(t+\tau)\rangle = \mathbf{E}[X(t)X(t+\tau)]$$

Esta transformación es otra forma eficiente de hacer análisis espectral, y en algunos casos puede aportar información adicional que complementa a la conseguida por la *FFT*. Al igual que la *FFT*, esta transformación se ha utilizado para extraer características en proyectos de *Machine Learning* con éxito.

Función de Autocorrelación

La *Función de Autocorrelación*, o *ACF*, describe cómo la correlación entre dos valores de la señal cambia conforme su separación cambia, es decir, representa la correlación de una señal con versiones de sí misma retrasadas en el tiempo. Matemáticamente, se define por la ecuación:

$$\mathrm{R}_{XX}(t_1, t_2) = \mathrm{E}[X_{t_1}\overline{X_{t_2}}]$$

Esta transformación permite obtener información sobre señales que son resultado de procesos *autorregresivos*, es decir, procesos en los que el valor de una

muestra depende de los valores anteriores de la misma. Con esta técnica, se pueden analizar propiedades como la periodicidad de una señal. Esta transformada se utiliza en aplicaciones tan diversas como óptica (espectroscopia), análisis músical (determinar el tempo de una base, por ejemplo), para visualizar el flujo sanguíneo en imágenes médicas de ultrasonidos. Existen métodos eficientes para calcular esta transformada, y también se ha utilizado con éxito con el fin de calcular características para modelado de *Machine Learning*.

Existe también una variación de esta transformación, llamada *Función de autocorrelación Parcial*, que permite complementar el análisis de la *ACF*, eliminando correlaciones intermedias entre muestras. Esta transformación es mucho menos eficiente que las anteriores, y en la Subsubsección 6.2.2 se ha demostrado que no se puede justificar su uso en esta librería, por lo que finalmente no se ha incluido para calcular características.

Transformada de Ondícula Continua

Una ondícula, o *wavelet*, es una función acotada en tiempo, es decir, que su amplitud empieza y termina en 0, que se puede ver como una "breve oscilación". En la Figura 5.7 se muestra un ejemplo de una ondícula.

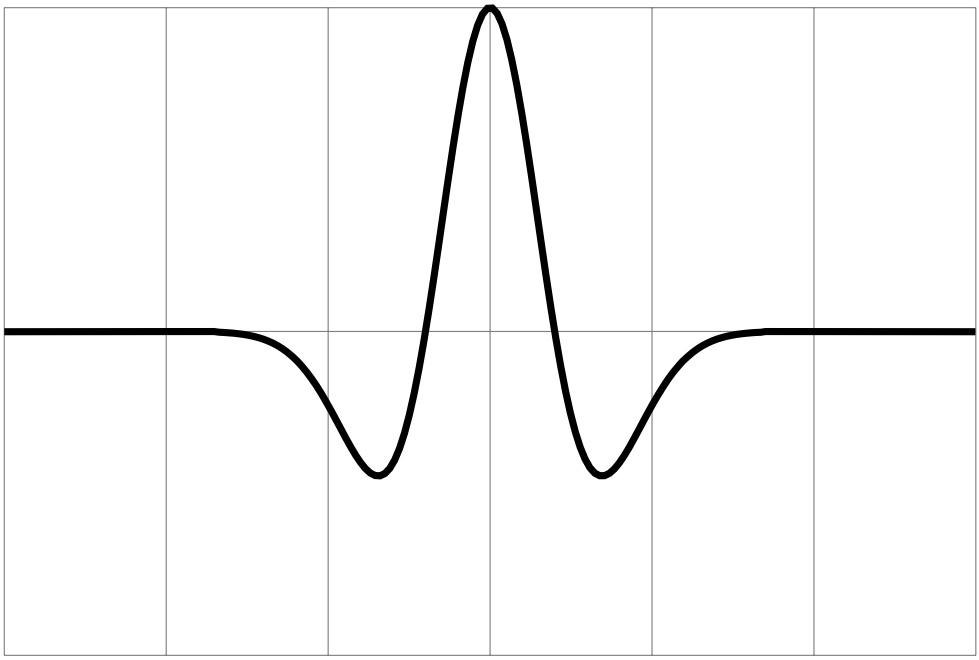

Figura 5.7: Ejemplo de ondícula

La *Transformada de Ondícula Continua*, o *CWT*, se define como la representación de una función de cuadrado integrable (una señal con valores reales o

complejos en la que la integral o suma del cuadrado de su módulo es finita) en una base ortonormal calculada en relación con una ondícula. Esta transformación es una generalización de la teoría de la *Transformada de Fourier*, que devuelve una representación de una señal en una representación en 2 dimensiones de tiempo/frecuencia. Es decir, con esta transformación se obtiene una visión de cómo evoluciona el espectro de una señal a lo largo del tiempo, conocido como análisis tiempo-frecuencia.

Formalmente, la *CWT* se calcula según la ecuación:

$$X_w(a,b) = \frac{1}{|a|^{1/2}} \int_{-\infty}^{\infty} x(t)\overline{\psi}\left(\frac{t-b}{a}\right) dt$$

Donde ψ es la ondícula elegida, llamada ondícula "madre". Para calcular la *CWT*, se calcula la convolución de la señal con deformaciones de la ondícula "madre". Estas deformaciones son de 2 tipos:

- Desplazamiento temporal: se mueve el comienzo de la ondícula a lo largo del tiempo, para así hacer la comparación con distintas partes de la señal
- Escala: son dilataciones y contracciones de la ondícula, que varían su frecuencia.

Este proceso se ilustra en la Figura 5.8

De esta forma se consigue una función de 2 dimensiones, que da una noción de cómo coincide cada sección de la señal original con la versión deformada de la ondícula, y esto se traduce en una representación de las propiedades de tiempo y frecuencia de la señal original. Variando la ondícula "madre", se puede conseguir modificar las propiedades de la transformada, detectando con mejor precisión unos tipos de vibraciones u otros. Esto hace que esta transformada sea más flexible.

En el campo del análisis de tiempo-frecuencia, se puede utilizar la *Short Time Fourier Transform* o *STFT*, que consiste en calcular una *Transformada de Fourier* dividiendo una señal en ventanas de tiempo. La *STFT* tiene una limitación en cuanto a la resolución que tiene tanto en tiempo como en frecuencia, y es que se realiza una division del espacio de tiempo-frecuencia uniforme, que no es lo ideal en muchas aplicaciones.

La *CWT* permite hacer un análisis espectral más exhaustivo de señales que no son estacionarias, ya que tiene la propiedad de ofrecer una resolución de tiempo-frecuencia dinámica, es decir, que la separación entre las muestras que se consiguen del espectro es diferente a bajas frecuencias y a altas. Debido al *Principio de Incertidumbre*, es imposible tener una buena resolución en tiempo y frecuencia a la vez, y con la *CWT* se consigue un equilibrio, por el que a frecuencias bajas se consigue una mayor resolución en frecuencia, a cambio de una baja resolución en

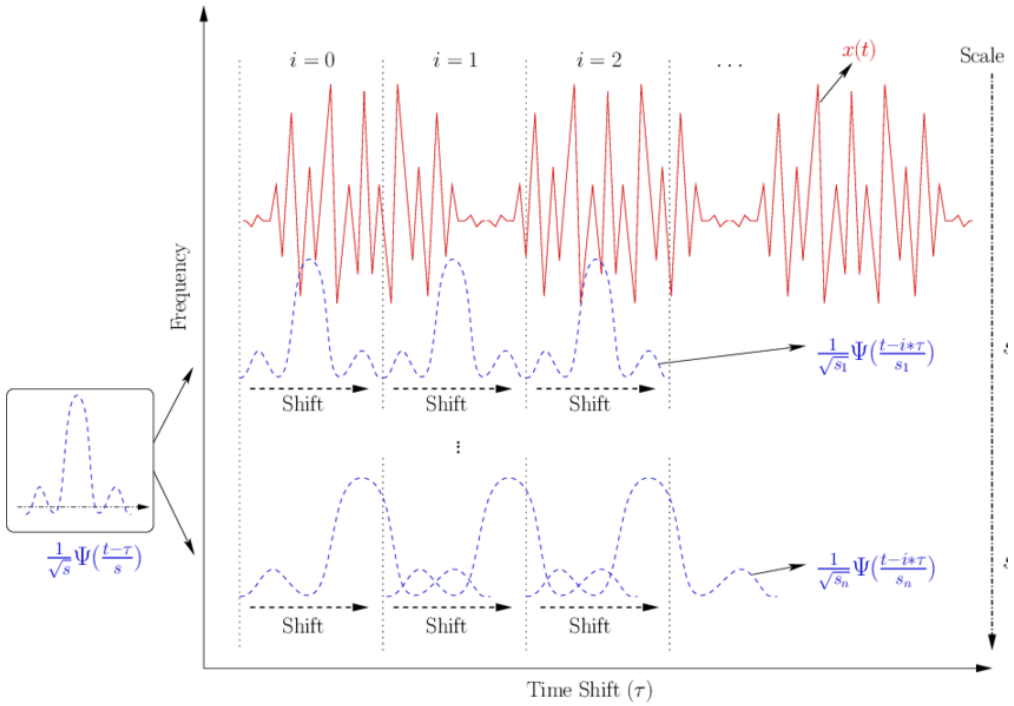

Figura 5.8: Visualización del cálculo de la *CWT* [28]

tiempo, y viceversa. En términos prácticos, esto quiere decir que con la *CWT* se puede detectar una vibración localizada en el tiempo con alta precisión.

En la Figura 5.9 se muestra una comparación de las resoluciones en tiempo-frecuencia de las distintas representaciones de la señal que se han visto hasta ahora. En ella se puede ver:

- Time Series: es la señal original, que sólo muestra información temporal sobre la señal
- Fourier Transform: es la *Transformada de Fourier*, que muestra únicamente información sobre las frecuencias
- Short Time Fourier Transform: que muestra información de tiempo-frecuencia, pero con una resolución uniforme en ambas dimensiones
- Wavelet Transform: que ofrece una resolución que ofrece una resolución de tiempo más alta para frecuencias más bajas, y viceversa

Estas características hacen que la *CWT* sea una buena transformación para obtener información sobre la evolución de las características espectrales de la señal de entrada a lo largo del tiempo, y en particular permite detectar variaciones en el

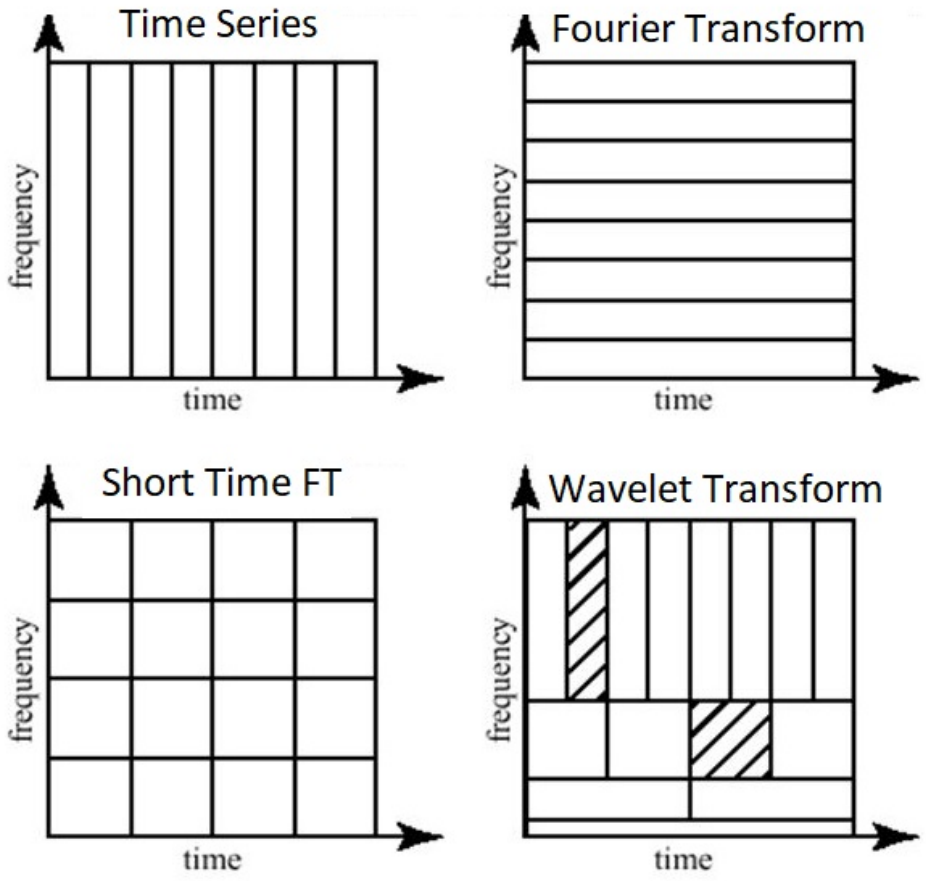

Figura 5.9: Comparación de la resolución de tiempo-frecuencia de distintas transformadas [29]

mismo localizadas en el tiempo. Esta transformada se ha utilizado con éxito en distintas aplicaciones de procesamiento de señales para fines médicos, procesamiento de imágenes (eliminar ruido, detección de bordes), detección de patrones de audio, y muchas otras más. Al igual que las anteriores, esta transformada también se ha utilizado con éxito en aplicaciones de *Machine Learning*, como en la detección y prevención de terremotos o detección de arritmias en electrocardiogramas.

5.2.4. Extracción de características

La extracción de características es otro paso esencial para el éxito de la librería. Las transformaciones que se han comentado hasta ahora producen todas una

señal a la salida, y esto quiere decir que por si solas no generan características compatibles con los algoritmos de *Machine Learning*.

Para solucionar este problema, es necesario aplicar técnicas de extracción de características sobre los resultados de las transformaciones. Estas técnicas son algoritmos que reducen una señal a uno o varios valores numéricos, que representan alguna propiedad sobre la señal. A continuación se describen las técnicas de extracción de características que se han utilizado en este proyecto.

Detección de Picos

El primer algoritmo que se ha utilizado es la detección de picos. Este algoritmo se encarga de localizar en la señal picos que cumplen unas características concretas. Un pico, o máximo local, se define como una muestra que tiene un valor mayor que sus dos muestras adyacentes.

Ya que las señales pueden contener ruido y otras componentes de alta frecuencia que provocan pequeñas oscilaciones en la señal y generan gran cantidad de máximos locales, es necesario filtrar los máximos y buscar los que sean más relevantes. Para ello, se estudian determinadas propiedades sobre cada uno de estos máximos:

- Altura: los picos tienen que tener una altura mínima, que puede ser relativa o absoluta
- Distancia: separación en muestras con respecto al pico más cercano
- Prominencia: es una medida de cuánto destaca un pico sobre la base que le rodea. Para calcular la prominencia, el algoritmo es:

La prominencia es un valor importante a la hora de filtrar picos que no son relevantes, o picos secundarios que surgen como "artefactos" cerca de picos reales. Esto se ilustra en la Figura 5.10 La prominencia que se utiliza en la detección de picos se basa en el concepto de *Prominencia Topográfica*, y el algoritmo es el siguiente:

1. Extender una línea horizontal desde la localización del pico actual hacia izquierda y derecha hasta llegar a un tamaño de ventana w_{size}, o hasta que cruce con la pendiente de otro pico más alto
2. En cada lado, encontrar el mínimo valor de la señal dentro del intervalo definido por el paso anterior. Estos puntos serán las bases de los picos.
3. La prominencia es la distancia vertical entre la altura del pico y la base más alta de las dos

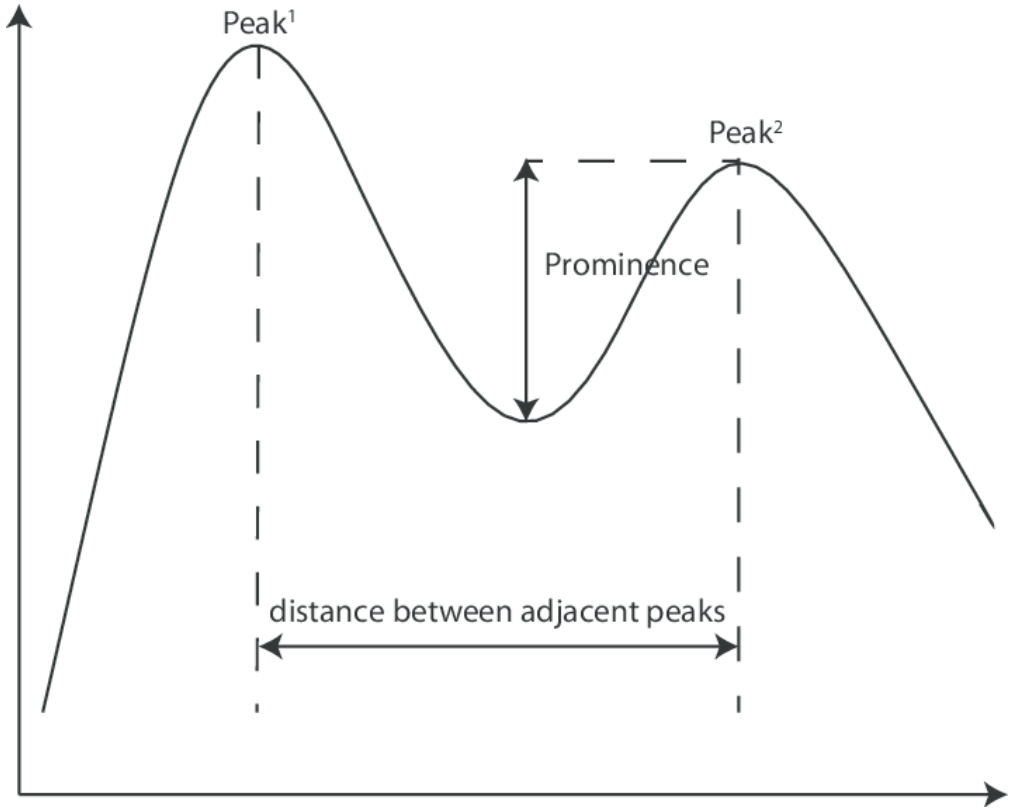

Figura 5.10: Ilustración de la prominencia de un pico [30]

Con estas propiedades, se detectan los picos en las distintas transformadas. En el caso de la *CWT*, como la transformación tiene 2 dimensiones (tiempo y frecuencia), la definición del algoritmo de detección se extiende de forma análoga a N-dimensiones. El último paso del algoritmo es ordenar los picos por su altura, de forma que se ordenan de mayor a menor. De ellos, se van a escoger K picos, que se utilizarán para generar las características.

Una vez elegidos los K picos, para cada uno se generan las siguientes características:

- Localización del pico en el tiempo. Por ejemplo: *Test.FFT_1_LOC*, que corresponde a la coordenada en muestras del segundo pico de la *FFT* de la señal *Test*.
- Localización del pico en Frecuencia (sólo en el caso de *CWT*). Por ejemplo: *Test.CWT_3_FREQ*, que corresponde a la coordenada frecuencia (relativa) del tercer pico de la *CWT* de la señal *Test*.
- Valor del pico Por ejemplo: *Test.PSD_1_PSD*, que corresponde al valor del

pico.

Esta técnica se aplica para extraer características de la *FFT*, *PSD* y de la *CWT*, esta última en su versión 2D.

A modo de ejemplo, siguiendo con el caso simplificado mencionado en la Subsubsección 5.2.3, con una señal de ejemplo generada con la fórmula:

$$y(x) = 1 + \sin(2 \times \pi \times 50) + \sin(2 \times \pi \times 100)$$

Donde su *FFT* era la mostrada en la Figura 5.6. Aplicando la detección de picos con $K = 3$ a la salida de la *FFT*, se obtiene el resultado mostrado en la Figura 5.11. Ahí se puede ver que el algoritmo detecta los 3 primeros picos, que corresponden con las frecuencias de las componentes que se han usado para generar la señal.

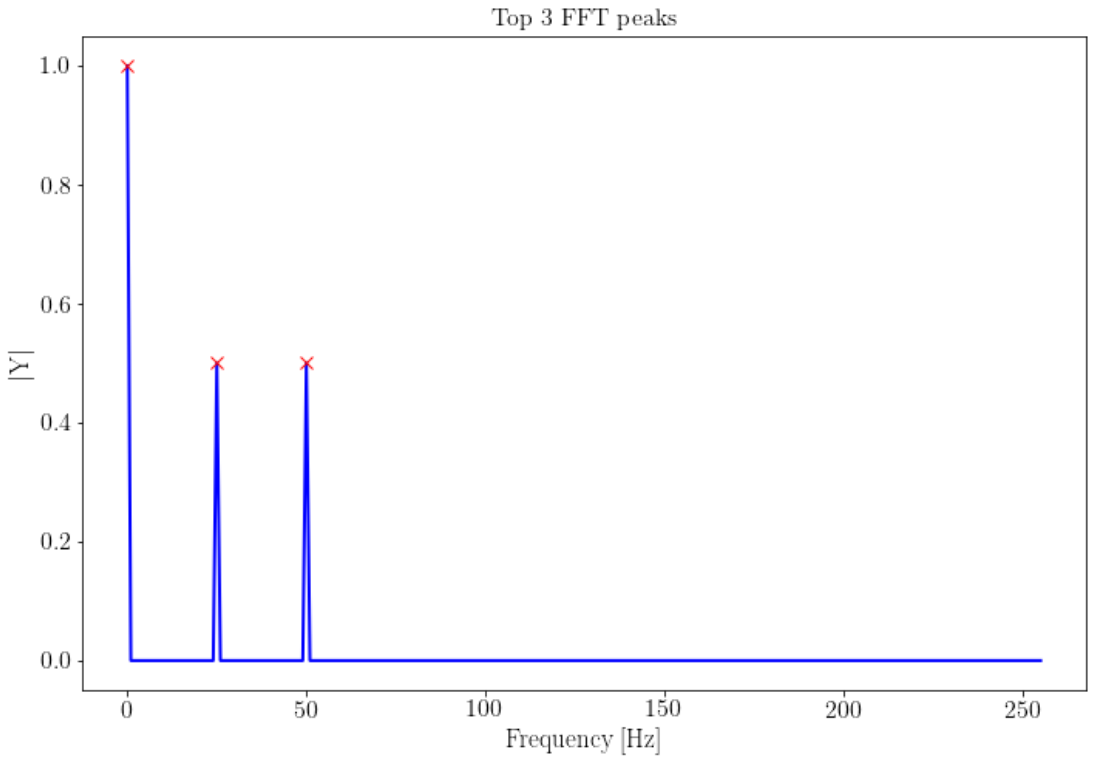

Figura 5.11: Visualización de los picos detectados en la *FFT* de la señal de prueba

Y de esta forma se conseguirían unas características que describen la señal original de forma precisa. Este es un caso simplificado, pero demuestra la utilidad de este algoritmo para extraer valores significativos de una señal o de una transformación.

Aun así, hay casos en los que la detección de picos no es útil, ya que la información esta distribuida a lo largo de toda la señal, o en un conjunto amplio de valores. Para estos casos, es necesario utilizar otras técnicas, como la que se explica a continuación.

Estadísticos

El otro principal algoritmo de extracción de características que se ha utilizado es la descripción estadística de señales o transformaciones. El objetivo de esta técnica es describir las propiedades de la señal a través de parámetros estadísticos bien estudiados. Los estadísticos que se han generado son los siguientes:

- Entropy: entropia de Shannon de la señal, según la formula

$$\mathrm{H}(X) = -\sum_{i=1}^{n} \mathrm{P}(x_i) \log_b \mathrm{P}(x_i)$$

- ZeroCrossings: número de cruces por 0 de la señal
- MeanCrossings: número de cruces por la media de la señal
- Quantile5: percentil 5
- Quantile25: percentil 25
- Quantile75: percentil 75
- Quantile95: percentil 95
- Median: mediana
- Mean: media
- StdDeviation: desviación típica
- RMS: raíz de la media cuadrática, calculado con la fórmula

$$x_{\mathrm{RMS}} = \sqrt{\frac{1}{n}\left(x_1^2 + x_2^2 + \cdots + x_n^2\right)}$$

Y con estos estadísticos se generan características como por ejemplo:

- Mediana de la *ACF*. Por ejemplo: *Test.ACF_Median*
- Número de cruces por la media de la señal original. Por ejemplo: *Test.RAW_MeanCrossings*

Esta técnica se aplica a la señal original (muestras originales de la señal, o RAW), y en conjunto con la *ACF*.

Distancias

Por último, se ha incluido una característica adicional, que por defecto no se calcula. Esta característica corresponde a una medida de distancia. Sin embargo, para calcular la distancia es necesario disponer de otra señal con la que comparar, y ya que en esta librería no se dispone de información suficiente como para calcular una referencia, se deja a discreción del usuario. Si este provee una señal que utilizar como referencia, se puede calcular una característica que devuelva una medida de la distancia entre las dos señales.

Esta característica tiene la ventaja de que es muy interpretable, ya que se es muy fácil explicar cuándo una señal se ha desviado demasiado de una señal de referencia. Sin embargo, la utilidad de este tipo de características depende en gran manera de la forma de calcular esta señal de referencia.

En la librería se incluyen 4 opciones diferentes de calcular la distancia, y la elección depende del problema en cuestión. Las distancias disponibles son las siguientes:

- Error cuadrático medio, que se calcula con la fórmula

$$\mathrm{ECM} = \frac{1}{n}\sum_{i=1}^{n}(\hat{Y}_i - Y_i)^2$$

- Error absoluto medio, que se calcula con la fórmula

$$\mathrm{EAM} = \frac{\sum_{i=1}^{n}|y_i - x_i|}{n} = \frac{\sum_{i=1}^{n}|e_i|}{n}$$

- Correlación cruzada, que se calcula con la fórmula

$$(f \star g)[n] \triangleq \sum_{m=-\infty}^{\infty} \overline{f[m]}\, g[m+n]$$

- Dynamic Time Warping, que se explica más en detalle a continuación

Las 3 primeras distancias miden la distancia que hay entre una señal y una referencia, asumiendo que las dos señales tienen las mismas características, con la misma longitud. Sin embargo, hay una opción más compleja, pero más flexible, que es utilizar el algoritmo de *Dynamic Time Warping*.

Este algoritmo permite medir la similaridad entre una señal y una referencia que pueden tener velocidades diferentes. Para ello, se permite hacer transformaciones no lineales sobre la señal, basadas en dilataciones en el tiempo y contracciones, y

se busca la combinación de modificaciones de la señal que minimicen la distancia entre las dos.

Para esto, el algoritmo recorre la señal muestra por muestra, y busca una muestra en la señal de referencia, aunque no sea en el mismo índice, que minimice la distancia. Además, se permite que varias muestras correspondan a una misma muestra de referencia. De esta forma, se consigue un efecto de dilatación y contracción temporal en la señal. En la práctica, se establecen límites sobre estas transformaciones, que hacen que este *warping* tenga sentido lógico y matemático en el problema en cuestión, y limitan el espacio de búsqueda para hacer más eficiente el algoritmo, ya que de otra manera el problema se convierte en un ineficiente problema de optimización cuadrático. Esto se ilustra en la Figura 5.12.

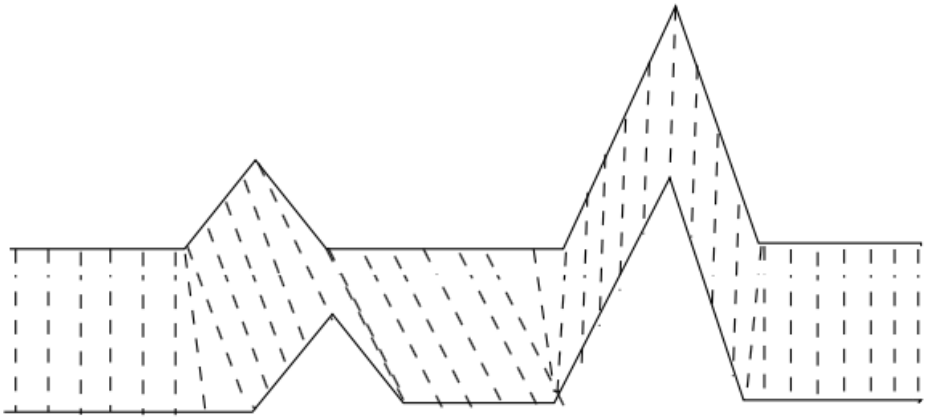

Figura 5.12: Dynamic Time Warping

De esta forma, el algoritmo busca una correspondencia óptima entre las muestras de la señal y la referencia, con las siguientes restricciones:

- Todas las muestras tienen que tener una correspondencia
- El primer y último índice de la señal tienen que corresponder con el primero y último de la referencia, respectivamente (aunque más muestras pueden tener la misma correspondencia)
- Las correspondencias tienen que ser monotónicamente crecientes, es decir, si se ha hecho una correspondencia $i \rightarrow k$ donde i es una muestra de la señal original y k es una muestra en la referencia, entonces para todo $j > i$ en la señal original, su correspondencia l debe cumplir $l \geq k$

Esta medida de distancia se ha usado con éxito en aplicaciones de reconocimiento de habla, donde se pueden comparar discursos de personas que hablan a velocidades diferentes, o también para estudiar patrones de movimiento.

5.3. Implementación

En esta sección se van a explicar los detalles más relevantes sobre la implementación de los diseños comentados en el apartado anterior, las técnicas utilizadas y las decisiones que se han tomado a la hora de implementar esta librería. En el Apéndice B se han extendido algunas de las explicaciones, incluyendo detalles de más bajo nivel y con *snippets* de código.

La implementación de la prueba de concepto de la librería se ha hecho en *Python*. Las razones de esta decisión se han explicado en detalle en el Sección 2.1, pero se pueden resumir en:

- Velocidad de desarrollo: este lenguaje, debido a su naturaleza dinámica, diversa librería estándar, y tersa sintaxis, facilita el desarrollo ágil
- Acceso a un gran ecosistema de librerías para programación numérica, con una gran base de conocimiento accesible en cuanto a la resolución de problemas típicos.
- Popularidad, lo que da acceso a una gran base de usuarios potenciales para el producto
- Integración con otras herramientas del ecosistema, en particular BigML y el caso de uso
- Familiaridad con el lenguaje, lo que permite acelerar aún más el desarrollo

Estas razones hacen de este lenguaje la mejor decisión para la implementación de esta librería.

Con el diseño descrito en los apartados anteriores en mente, se ha estructurado la implementación de forma que coinciden con los módulos descritos en la Figura 5.3. Estos módulos se corresponden con módulos de *Python*, que son unidades de código independientes que además permiten ser importados por separado, y ser compartidos y reutilizados. Esta estructura permite además realizar *testing* de forma independiente para cada uno de los componentes.

Siguiendo con la filosofía modular, la implementación de cada uno de los módulos se ha hecho de forma que desde ellos, el resto de módulos se ven como cajas negras de las que sólo se conoce su interfaz, la forma pasar y recibir datos de ellos. De esta manera, se consigue una implementación de módulos desacoplados unos

de otros, y esto permite la reutilización de código, ya que estos módulos se pueden mover de un sitio a otro y seguir funcionando, al mismo tiempo que permite cambiar un módulo por otro (introducir mejoras, como incluir nuevas transformaciones o eliminar otras) sin requerir modificaciones en ningún otro componente del sistema. A lo largo del desarrollo se ha hecho buen uso de esta propiedad del diseño y de la implementación, introduciendo cambios en función de los resultados obtenidos en el benchmark, como se describe más adelante en la Sección 6.2.

A continuación se describen algunos detalles de la implementación que son relevantes, y que han sido decisiones concretas que aprovechan las ventajas y características de *Python*.

5.3.1. Configuración

Uno de los requisitos de la librería es que, aunque se ofrecen unos valores por defecto, debe darse la posibilidad a un usuario experto de que pueda modificar el comportamiento las transformaciones y de la extracción de características para adaptar su funcionamiento, aprovechando así el conocimiento del dominio.

Permitir esto es clave para poder utilizar esta herramienta en un amplio rango de casos de uso. Aunque se busca ofrecer el conjunto de valores por defecto que de mejores resultados en general, los mejores resultados para cada uno de los casos se consiguen personalizando el comportamiento adaptándolo al dominio concreto, y para eso es necesario permitir al usuario experto configurar la librería.

Desde un punto de vista de implementación, en *Python* este problema se resuelve añadiendo argumentos opcionales a las funciones, lo cual funciona muy bien para la mayoría de situaciones. Esto es algo que en particular en este lenguaje es popular, gracias a los *keyword arguments* con los que se puede mantener el código fácil de leer al mismo tiempo que se le añaden parámetros.

No obstante, en el caso de esta librería, se van a combinar numerosas funciones, que cada una por separado ya son suficientemente complejas como para que la configuración por argumentos de la función sea compleja. Esto quiere decir que, aunque esto funcionaría, sería un problema a la hora de realizar el mantenimiento del código, y es una carga para los usuarios de la librería.

Además, desde un punto de vista de usabilidad, este es un método que no da mucha libertad a los usuarios. Codificando la configuración en los propios argumentos, se acopla la configuración, que son datos, al código, y eso quiere decir que si se quiere realizar pruebas en las que se cambia la configuración, es necesario modificar el código.

Es por esto que en este caso se ha decidido utilizar un método diferente. En esta librería, la configuración se va a pasar mediante un único parámetro opcional en forma de diccionario.

Esto permite que la configuración y el código vayan por separado. La configuración se puede describir en un archivo separado del código, por ejemplo en *JSON*, que tiene una buena compatibilidad con los diccionarios de *Python*. Así, para ejecutar una nueva prueba, solo hay que cambiar este archivo y no el código.

Facilitar la rápida experimentación es una cualidad muy útil, especialmente en tareas de *Feature Engineering* que son tan experimentales, donde la prueba y error, y rápida iteración son claves a la hora de seleccionar el mejor conjunto de características.

Con este diseño, se busca un equilibrio entre una herramienta que puede ofrece utilidad en el mayor número de casos de uso, manteniendo al mismo una interfaz sencilla gracias a unos valores por defecto y el uso de parámetros opcionales.

5.3.2. Testing

El facilitar el *testing* es uno de los objetivos del diseño. A pesar de que mantener una *test suite* conlleva un coste, ya que obliga al desarrollador a escribir más código, al final tiene un beneficio neto, que hace que el desarrollo sea más rápido y de mayor calidad que uno sin tests.

Tener un buen conjunto de tests, y desarrollar con este objetivo en mente, tiene numerosas ventajas:

- Detección rápida de errores
  - Menos tiempo resolviendo errores, *debugging*
  - Más confianza para introducir cambios en la librería
  - Menos errores llegan a la fase de producción, lo que aumenta la percepción de calidad del código y la experiencia del usuario
- Código de mejor calidad
  - Incita a un diseño más modular, con las ventajas que esto conlleva
  - Obliga al desarrollador a pensar en la interfaz y guía el proceso de desarrollo
  - La confianza para introducir cambios incita a introducir mejoras más frecuentes

A la hora de diseñar las transformadas, se va a seguir la misma filosofía, manteniendo el mayor nivel de desacoplo posible entre la implementación elegida de una transformada concreta, y su uso. Para esto, se han separado las transformadas en su propio módulo, y a su vez, cada una de ellas se encapsulan en una

función *wrapper* que abstrae la interfaz concreta de la librería que implementa la transformada.

Lo mismo se ha buscado con la implementación de las características, que se encapsulan en su propio componente, que en este caso es una función que es reutilizable y *testeable*.

5.3.3. Gestión de nulos

Uno de los problemas con los que hay que lidiar es con nulos en los datos. Cuando se da este caso, hay dos soluciones posibles.

Por un lado, la opción más directa y sencilla es la de levantar una excepción o un error, y obligar al usuario a gestionar este error. Esta opción tiene más ventajas aparte de la simplicidad, y es que obliga al usuario a gestionar estos errores, lo que lleva un código más robusto.

Pero por otro lado esto presenta un problema desde el punto de vista de usabilidad, y es que en ocasiones esto no es la opción que desea el usuario, y es la librería la que se encuentra en mejor posición para ofrecer una solución en caso de que se encuentre una entrada nula.

De esta forma, se ha decidido que a la hora de recibir una entrada nula, el comportamiento por defecto que ofrece la librería es el de devolver un diccionario con la misma estructura que se habría devuelto si la entrada fuera correcta (que dependerá de la configuración), y como valores, se rellenan todos con "None".

Esto tiene la ventaja de que un error puntual en los datos no interrumpa por completo el *pipeline* de *ETL* del usuario debido a problemas con la estructura. Esta propiedad se ha aprovechado a la hora de integrar la librería con el caso de uso. Al mantener la misma estructura, los registros vacíos se pueden almacenar en las mismas tablas u otras similares, lo que da persistencia a estos errores y permite su futura consulta y análisis.

Esta decisión en cualquier caso obliga al usuario a detectar antes de las transformaciones los posibles problemas con sus datos, lo cual es una buena práctica.

5.3.4. Benchmark

Con el fin de poder obtener una estimación válida de la utilidad de las características generadas con esta librería, se ha implementado un sistema que de forma automática haga una evaluación de un modelo de ejemplo sobre una gran variedad de datasets, de orígenes distintos.

La tarea principal para la que se ha diseñado (aunque no hay razón por la que este sistema no funcione en cualquier otro tipo de tarea) es la de la clasificación basada en señales. De esta forma, se han buscado conjuntos de datos que contengan registros con señales como características y como objetivo una categoría.

Existe un repositorio que recopila este tipo de datasets, y que ha sido utilizado como referencia en otros artículos en este espacio [31]. Este repositorio está recopilado es el *The UCR time series archive* [32], y contiene 128 datasets como el que se busca para este proyecto.

Los datos se encuentran cada uno en una carpeta con el nombre del dataset, y dentro de cada una se encuentran 2 archivos, uno con datos para el entrenamiento, y otro para el testing. De esta forma, al utilizar siempre los mismos datos para entrenamiento y test, podemos hacer comparaciones entre ejecuciones con diferentes transformaciones. Los datos vienen en formato *.ts*, que se pueden procesar con la librería *sktime* en *Python*.

El sistema de benchmark se encarga de recorrer la estructura de directorios y archivos de este repositorio de datos, y para cada dataset, realiza los siguientes pasos.

- Leer los archivos *.ts* con los datos de entrenamiento y test.
- Aplicar una función de transformación que se obtiene como entrada. Esta función de transformación contiene el código que cambia entre distintas ejecuciones del benchmark, y se ofrece un valor por defecto que simplemente transforma la señal con la configuración por defecto de la librería. Para implementar una versión personalizada de esta función, sólo es necesario implementar una función que recibe un *pandas DataFrame* de entrada y devuelve otro con las características ya generadas. Esta función se ejecuta una vez con los datos de entrenamiento, y otra con los datos de test.
- Crear un modelo predictivo de clasificación con los datos transformados. En particular, se va a crear un modelo de *Ensemble* de BigML con los valores por defectos, que crea un *Random Forest* de 100 árboles.
- Evaluar el modelo con los datos de test.
- Guardar un resumen con los resultados de la evaluación, así como información sobre la importancia de las distintas características generadas.

Este proceso se ha ilustrado en la Figura 5.13.

Gracias su estructura el benchmark es altamente paralelizable. Cada iteración, las operaciones sobre cada conjunto de datos es completamente independiente del resto del programa, por lo que se pueden ejecutar como una función en otro proceso. Se ha aprovechado esta propiedad para reducir significativamente el tiempo de ejecución, y esta mejoría se observa más de forma más acusada en máquinas con múltiples núcleos.

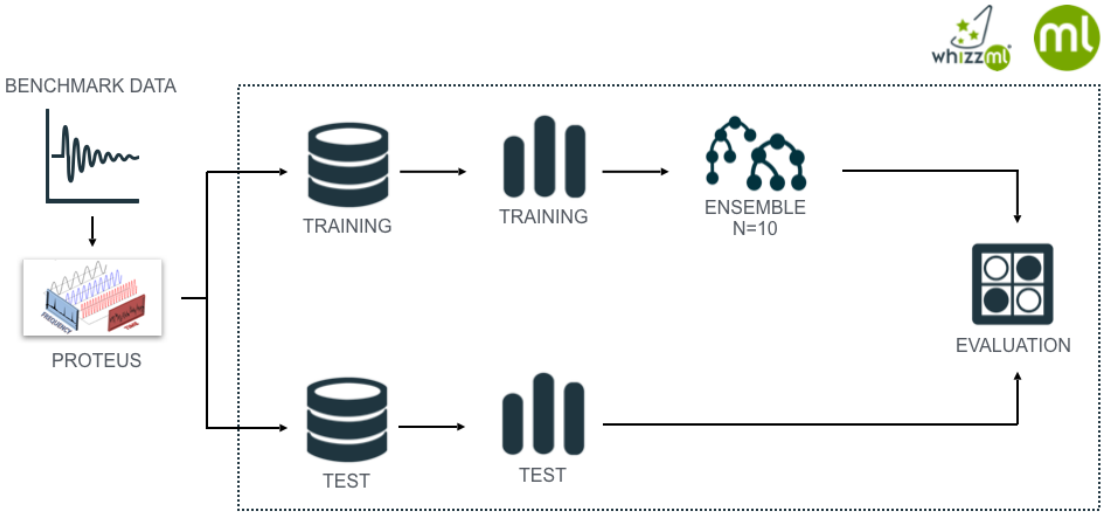

Figura 5.13: *Machine Learning workflow* para el benchmark

Se puede acelerar también el proceso de modelado. Al estar utilizando una plataforma externa para el modelado como es BigML, se pueden aprovechar las facilidades que estas ofrecen a la hora de acelerar la ejecución del benchmark.

En particular, se ha aprovechado el uso de *WhizzML*, *DSL* para la automatización de procesos de *Aprendizaje Automático* dentro de la plataforma de BigML. Este es un caso de uso ideal para esta herramienta.

Este lenguaje permite definir las distintas tareas que componen el proceso de *Machine Learning* que se ha definido para el benchmarking, de una forma concisa. Con este lenguaje se puede crear un *Script*, que es un recurso más que se puede gestionar desde la interfaz de la plataforma, o en este caso, a través de los *bindings* de *Python*. El código del *script* es el siguiente:

```
;; Very simple script to create an ensemble
;; and run the evaluation

;; Helper to prepare the dataset that makes sure
;; the objective_field is correct
(define (prepare-dataset src-id)
  (let (ds-id (create-and-wait-dataset src-id)
        target-id
            ((find-field (resource-fields ds-id)
                         target-name)
                         "id"))
    (update ds-id {"objective_field" {"id" target-id
      }})))
```

```
;; Create and make sure the objective_field is
   configured
(define train-ds (prepare-dataset train-src))
(define test-ds (prepare-dataset test-src))

;; Create an ensemble and evaluate
(define ensemble (create-ensemble train-ds))
(define evaluation
    (create-and-wait-evaluation ensemble test-ds))
```

Snippet 5.1: WizzML del workflow para el benchmark

Este script realiza las siguientes tareas:

- Crear una función cuyo objetivo es asegurarse de que los datos en origen tienen la estructura correcta, y la variable objetivo bien asignada
- Crear dos datasets, uno de train y uno de test, partiendo de las fuentes originales, ejecutando antes la función definida para preparar los datasets
- Crear un ensemble con la configuración por defecto de la plataforma, entrenándolo con el dataset de entrenamiento
- Crear una evaluación del ensemble que se acaba de crear, utilizando el dataset de test

Cabe destacar la simplicidad de este código. Con una ejecución de este script se obtienen como resultado dos recursos que son el ensemble y la correspondiente evaluación, que se pueden recuperar en caso de que se desee. A pesar de no especificarlo explícitamente en el código, la plataforma automáticamente va a optimizar la ejecución del mismo, paralelizando las tareas cuando sea posible. Este script se va a ejecutar una vez por cada dataset que forma parte del benchmark.

Para incluirlo en la plataforma es necesario incluir el código *WhizzML*, junto a un simple archivo de *JSON* que describe el script, declarando sus entradas y salidas, da acceso a un script utilizable en cualquier momento.

El extraer esta lógica del código *Python* del benchmark simplifica sensiblemente el código del cliente del benchmark, y permite concentrarse en cargar los datos e imprimir los resultados.

Con el *Script* definido, se importa en la plataforma, y ya se tiene acceso a él a través de la interfaz gráfica (que ofrece asistencia a la hora de ejecutarlo gracias a

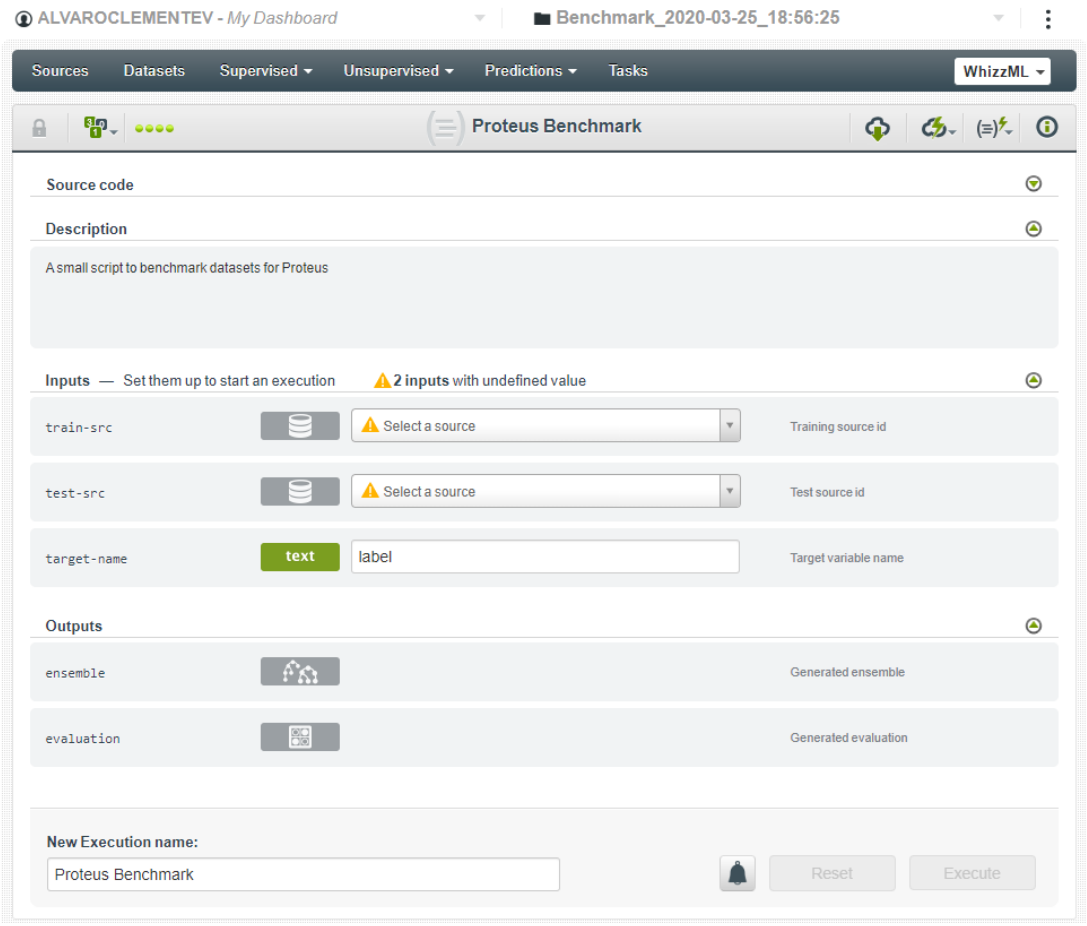

Figura 5.14: Script en BigML

que sabe los tipos de los datos, como sugerencias de autocompletado), o la API a través de los *bindings*. La interfaz de la plataforma se muestra en la Figura 5.14.

Otras ventajas que ofrece el uso de BigML y *scripts* es la trazabilidad. Con los métodos de auditoría que ofrece esta plataforma se dispone de un registro que permite mantener un histórico de las ejecuciones del benchmark, de los pasos intermedios de los procesos, y de los resultados.

En la Figura 5.15 se muestra una captura de la organización de los benchmark en *proyectos* de BigML. Esto permite tener todos los recursos relacionados con este benchmark agrupados, cosa que simplifica su acceso, intercambio y en caso de que se requiera, permite eliminar todos los recursos en una sola acción.

En la Figura 5.16 se muestra una captura de la vista de las ejecuciones para un benchmark concreto. Así podemos hacer un seguimiento de los resultados intermedios generados por cada una de las tareas.

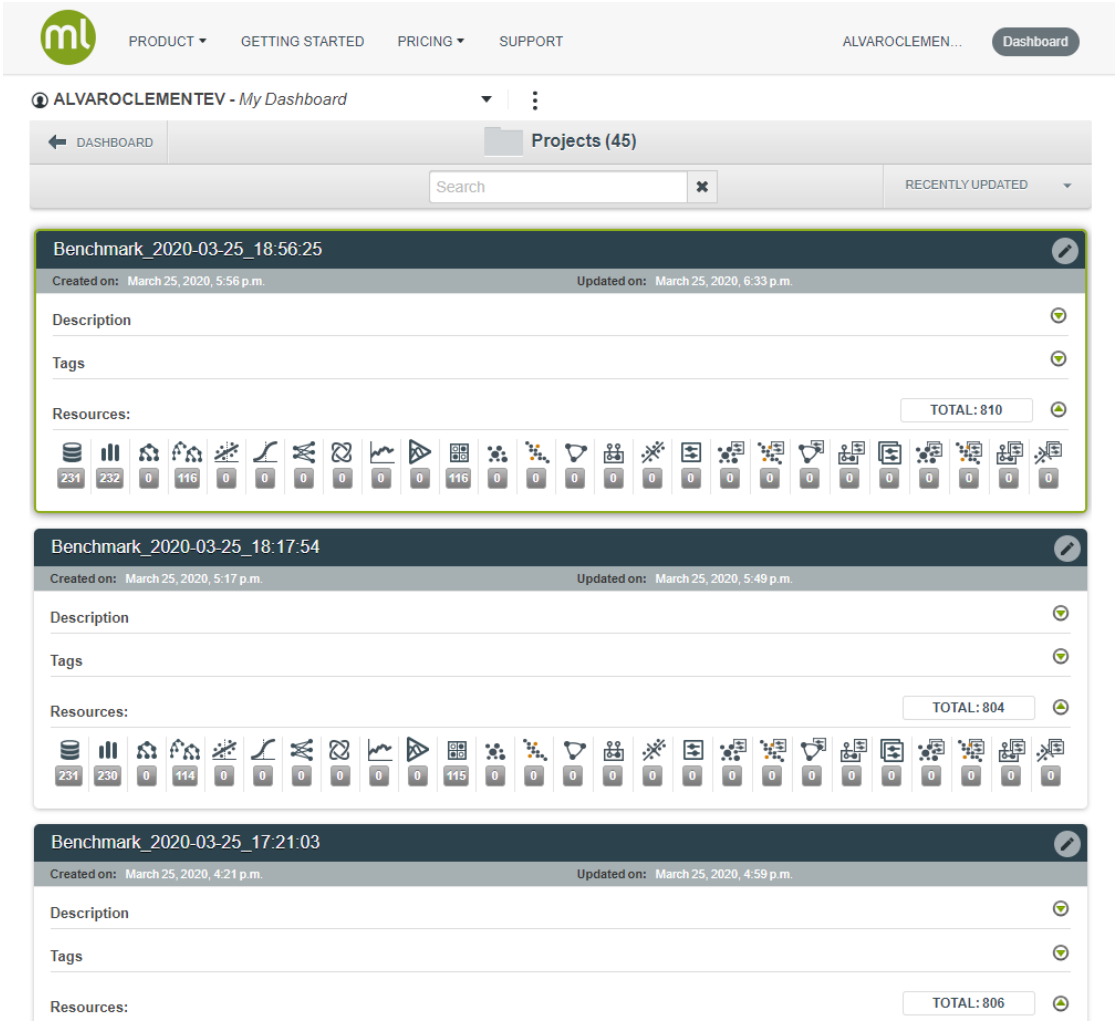

Figura 5.15: Proyectos en BigML

Haciendo uso de esta técnica se tiene acceso a los resultados de las evaluaciones de cada uno de los datasets creados, sin necesidad de hacer ninguna acción específica en el cliente. Esto, junto con las visualizaciones que vienen incluidas en la plataforma, son herramientas de gran ayuda a la hora de interpretar los resultados obtenidos. En la Figura 5.17 se muestra un ejemplo de una evaluación en uno de los benchmark ejecutados.

Una vez ejecutado el benchmark, se imprime un resumen en pantalla de la ejecución, y se generan unos archivos en formato *CSV* que contienen detalles de los resultados de cada una de las evaluaciones que componen el dataset. Estos *CSV* son útiles a la hora de analizar individualmente cada uno de los resultados, y permiten explorarlos y encontrar resultados interesantes, que luego se pueden

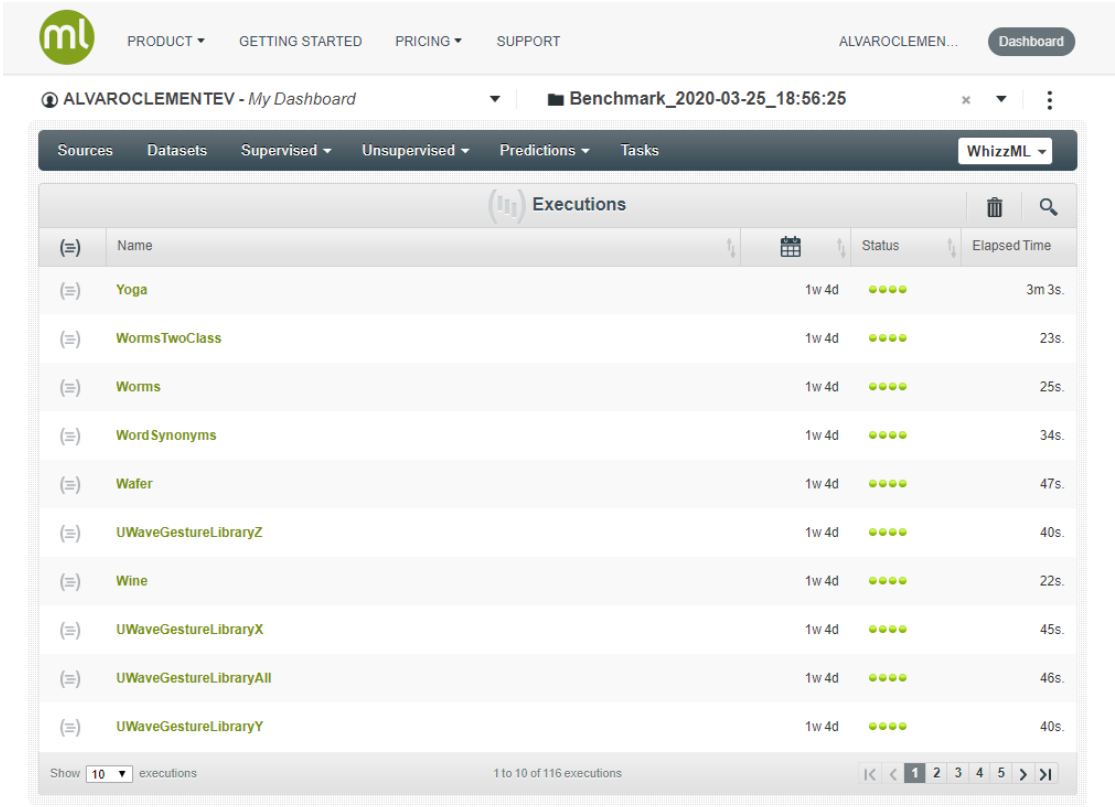

Figura 5.16: Ejecuciones de un benchmark en BigML

explorar en detalle y visualizar en BigML.

Por último, cabe destacar que, a pesar de que el repositorio original utilizado como base para el benchmark contiene, a fecha de escritura de este documento, 128 datasets, en la práctica alguno de estos datasets tiene irregularidades en el formato, que para algunas pruebas, plantea un problema.

En particular, 12 de ellos tienen señales que tienen tamaños diferentes, y esto es un problema a la hora de utilizar transformaciones basadas en distancia a señales de referencia. Ya que el conjunto original tiene 128 datasets, se ha considerado que aún ignorando estos datasets problemáticos se obtiene suficiente variedad, por lo que estos datasets no se van a utilizar para el benchmark. Así, el benchmark en la práctica consta de 116 datasets.

La lista concreta de datasets que se han utilizado y los que se han ignorado, se detalla en el Apéndice A.

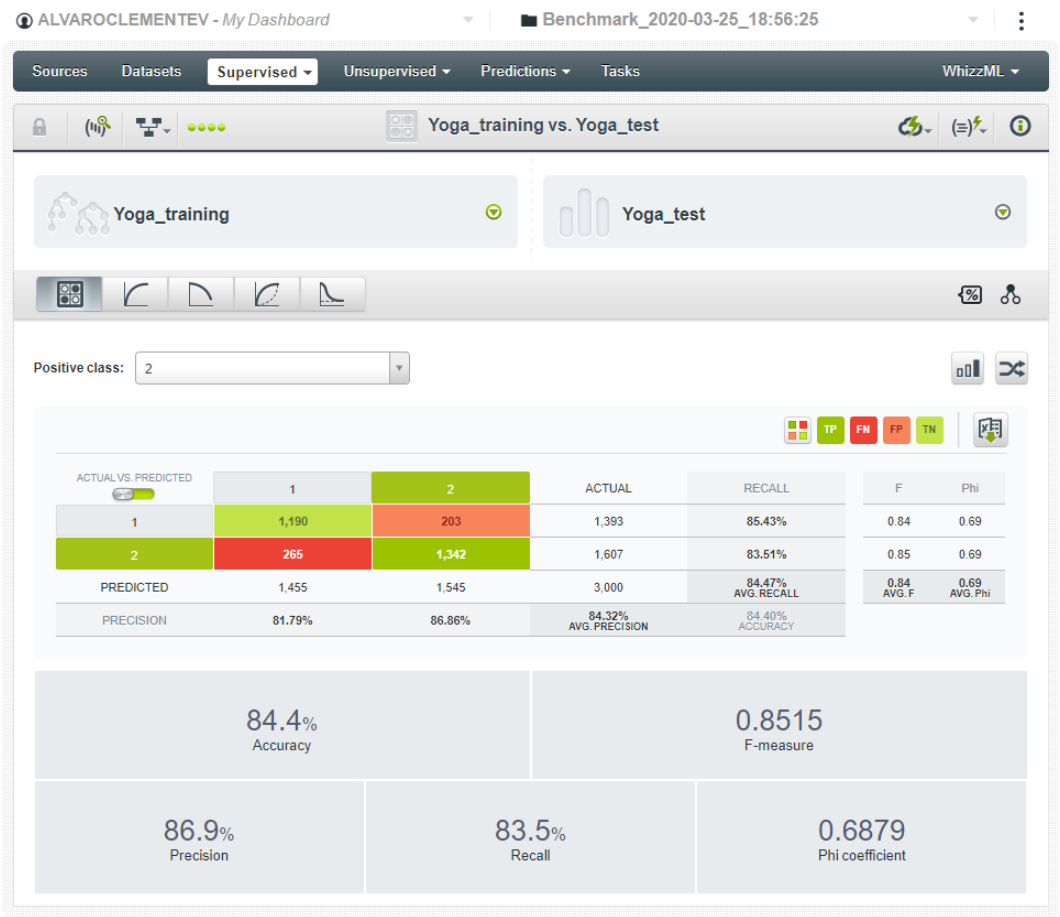

Figura 5.17: Evaluación de un modelo de un benchmark

Capítulo 6

Análisis de resultados

En este capítulo se va a realizar un análisis de los resultados obtenidos. En primer lugar, se va a explicar los resultados obtenidos tras la implementación de la prueba de concepto del caso de uso. A continuación, se va a comentar los resultados de los experimentos realizados basados en el benchmark, que introducen mejoras en la versión de la librería implementada para el caso de uso. Por último, se va a analizar la configuración final que se ha considerado la que mejor cumple con los objetivos marcados al inicio del proyecto.

En primer lugar, el resultado principal de este proyecto es la producción de una librería que sirve como herramienta para automatizar el proceso de *Feature Engineering* con señales.

Se ha implementado una librería que expone una única función, que por defecto se puede utilizar con apenas dos líneas de código, como se muestra a continuación:

```
>>> from bigml.proteus import proteus
>>> features = proteus.generate_features(signal,
                         signal_name='Test')
>>>
>>> features
{
    'Test.RAW_Entropy': 4.9281976097986595,
    'Test.RAW_ZeroCrossings': 20,
    'Test.RAW_MeanCrossings': 39,
    'Test.RAW_Quantile5': -0.549903355691758,
    'Test.RAW_Quantile25': 0.5033459903311247,
    'Test.RAW_Quantile75': 1.4966540096688727,
    'Test.RAW_Quantile95': 2.5499033556917503,
    'Test.RAW_Median': 1.0,
    'Test.RAW_Mean': 0.9999999999999999,
```

```
    'Test.RAW_StdDeviation': 1.0000000000000002,
    'Test.RAW_RMS': 1.4142135623730951,
    'Test.FFT_1_LOC': 0,
    'Test.FFT_1_VALUE': 0.7812499999999998,
    'Test.FFT_2_LOC': 13,
    'Test.FFT_2_VALUE': 0.36295089815546944,
    'Test.FFT_3_LOC': 26,
    'Test.FFT_3_VALUE': 0.3323023053090839,
    'Test.PSD_1_LOC': 0,
    'Test.PSD_1_VALUE': 72.52249582307844,
    'Test.PSD_2_LOC': 13,
    'Test.PSD_2_VALUE': 33.92925300844974,
    'Test.PSD_3_LOC': 26,
    'Test.PSD_3_VALUE': 31.18786415340304,
    'Test.ACF_Entropy': 5.298317366548037,
    'Test.ACF_ZeroCrossings': 24,
    'Test.ACF_MeanCrossings': 21,
    'Test.ACF_Quantile5': -0.4046850374777887,
    'Test.ACF_Quantile25': -0.17783580474379412,
    'Test.ACF_Quantile75': 0.09749986759987081,
    'Test.ACF_Quantile95': 0.6160257287345352,
    'Test.ACF_Median': -0.039060868528057055,
    'Test.ACF_Mean': 0.0025,
    'Test.ACF_StdDeviation': 0.2937472799873641,
    'Test.ACF_Variance': 0.08628746449997489,
    'Test.ACF_RMS': 0.29375791819111,
    'Test.CWT_1_TIME': 56.0,
    'Test.CWT_1_FREQ': 0.04924242424242424,
    'Test.CWT_1_VALUE': 24.34388493256909,
    'Test.CWT_2_TIME': 136.0,
    'Test.CWT_2_FREQ': 0.04924242424242424,
    'Test.CWT_2_VALUE': 24.342335541256954,
    'Test.CWT_3_TIME': 76.0,
    'Test.CWT_3_FREQ': 0.04924242424242424,
    'Test.CWT_3_VALUE': 24.341635400184003
}
```

Snippet 6.1: Ejemplo de uso de la función principal

Esto es un ejemplo de la usabilidad que se buscaba, ya que no ha sido necesario código para interactuar con la librería, ni tener ningún conocimiento específico

sobre cuáles son las transformaciones que se generan.

A continuación se muestran el resto de pruebas que se han hecho, primero con la prueba de concepto para el caso de uso, y después las mejoras que se han introducido con experimentación y benchmarking.

6.1. Caso de uso

Debido a restricciones con el calendario, ha sido necesario desarrollar una prueba de concepto inicial de la librería para poder comprobar la viabilidad del uso de *Machine Learning* para la detección de fallos en entornos de fabricación de automóviles. De esta forma, y tal y como se describió en la Sección 4.3, se ha hecho una primera implementación basada en una investigación previa, y pruebas con los datos disponibles. El diseño e implementación que surgen como resultado de esta primera prueba de concepto, serán la base sobre la que se realicen cambios y mejoras una vez se disponga del sistema de benchmarking, cuyos resultados se exponen en la Sección 6.2.

Como se comentó en la Sección 4.2, el caso de uso que se va a implementar tiene como objetivo principal la detección rápida de errores en un proceso de soldadura, para permitir así solucionar el problema en la misma estación de soldadura, donde es barato y rápido de solucionar.

En este caso se dispone de un conjunto de datos que contienen información sobre las máquinas de soldadura, y a los que se le añade características sobre señales recogidas de sensores durante el proceso de soldadura y transformadas con la librería.

6.1.1. Solución

Este es un problema de clasificación, donde el objetivo es identificar una soldadura como error o éxito, donde el objetivo es la clase de error ("clase positiva"). Debido a limitaciones en el sistema de información existente, no se tiene acceso a datos sobre cuáles de las soldaduras provocaron un error real en las siguientes fases de producción, por lo que este es un problema que se tiene que plantear utilizando modelos de *Machine Learning* no supervisado. Además, por la naturaleza del proceso, este es un problema de clasificación con unas clases extremadamente desequilibradas, donde los casos donde no se ha producido un error son más de mil veces más frecuentes.

Como solución, se ha planteado un *workflow* que se basa en detección de anomalías. La hipótesis es que, los errores deben resultar en medidas de los sensores que son diferentes a las de no errores, y en un dataset tan desequilibrado, deberían ser detectables como anomalías. Esta es una técnica que se utiliza con éxito

en problemas similares, como en detección de fraude en operaciones bancarias, o detección de intrusiones en redes.

De esta forma, se va a realizar *Feature Engineering* sobre las señales resultantes de los procesos de soldadura, y con las características extraídas, se genera una *anomaly score* utilizando *Isolation Forest* (el algoritmo detrás del *Anomaly Detector* de BigML), que devuelven un valor entre 0 y 1 que describe cómo de anómalo es un registro en comparación con lo observado durante el entrenamiento. Al mismo tiempo, para cada detector, se definen unos umbrales que son valores límite que marcan la frontera entre un registro común y uno anómalo.

Ya que se están analizando procesos en distintos robots, que hacen procesos diferentes y se encuentran en condiciones diferentes, se va a entrenar un modelo distinto con diferentes fronteras por cada entidad en el sistema.

Con este objetivo en mente, y como resultado de una investigación previa, se ha implementado una primera versión de la librería, que se basa en las siguientes transformaciones, que se han explicado en la Subsección 5.2.3:

- FFT: *Transformada Rápida de Fourier*, sobre la que se extraen los 3 mayores picos
- PSD: *Densidad Espectral de Potencia*, sobre la que se extraen los 3 mayores picos
- ACF: *Función de Autocorrelación*, sobre la que se extraen estadísticos
- PACF: *Función de Autocorrelación Parcial*, sobre la que se extraen estadísticos
- CWT: *Transformada de Ondícula Continua*, sobre la que se extraen los 3 mayores picos
- DTW distance: *Distancia mediante Dynamic Time Warping*

Para poder calcular la *DTW distance*, es necesario tener una señal de referencia con la que comparar, y como primera aproximación se ha calculado la señal media para cada entidad, es decir, para cada una de las máquinas realizando soldaduras. Para facilitar la comparación entre valores, el valor de la distancia se normaliza dividiendo por la distancia de la referencia a la señal nula (señal con todos los valores a 0). Con esto se intenta asegurar que la distancia está en un rango limitado de valores, y permite comparar distancias entre señales con magnitudes diferentes y visualizarlas.

De estas transformaciones cabe destacar que existe cierta redundancia entre la información que devuelven las características, con algunas de ellas con correlaciones altas, y otras que pueden no tener ninguna relación con el objetivo. Sin embargo,

esto no es un problema grave ya que los modelos que se están utilizando están basados en árboles, que son buenos ignorando variables que no son útiles.

6.1.2. Resultados

Se ha tenido acceso a una muestra reducida de ejemplos de soldaduras recopilada de forma manual. Estos datos se han cruzado con los de fases posteriores, y se ha podido aislar casos de errores que se debieron detectar. Estos son ejemplos de los casos que se está buscando detectar con esta solución, y por tanto puede servir para hacer una primera evaluación de los resultados que se pueden obtener.

Se han cruzado los identificadores de estos registros recibidos con la muestra de datos de entrenamiento a la que se ha tenido acceso para desarrollar modelos, y se han hecho evaluaciones de los resultados que se habría conseguido si estos modelos hubiesen estado generando predicciones. Esto se va a utilizar como una primera prueba de la utilidad de la solución escogida para este problema, y en particular, de las características generadas por la librería.

Con esta pequeña muestra se han podido comprobar la utilidad de algunos de los modelos. Para hacer la evaluación no tiene sentido fijarse en la *accuracy*, ya que es un problema de clases desequilibradas, y no aporta información útil. En su lugar, se van a utilizar la *precision* y *recall*, que nos permiten medir el volumen de falsos positivos, y la capacidad de detectar todos los errores, respectivamente. A continuación se muestran algunos de los resultados obtenidos.

Entidad A

A continuación se muestran los resultados obtenidos al entrenar un *Anomaly Detector 2459* muestras, correspondiente a la *Entidad A*, y que se ha evaluado con muestras de 6 errores reales de producción. Este es un ejemplo de un modelo que de haber estado funcionando, habría sido capaz de detectar todos los errores, aunque habría generado número elevado de falsas alarmas también. Las scores asignadas por este modelo son las siguientes:

En la Figura 6.1 se puede ver una visualización de la distribución de los resultados, separados en el eje X por un indicador que muestra si el error hubiera sido detectado por el sistema experto que existe actualmente (False quiere decir que el sistema actual lo ha marcado como error, True lo contrario), en el eje Y se muestra el *anomaly score* asignado, y el color muestra el nivel de urgencia de la potencial alerta generada donde verde es alerta, naranja representa sospecha de error, y azul es normal. Aquí se puede ver cómo hay se han marcado numerosos registros como alertas, entre ellos, todos los errores que no habían sido detectados por el sistema experto.

Las métricas de la evaluación se pueden ver en la Tabla 6.2.

	Anomaly Score	Nivel de Alerta
Error 1	0.56	Urgente
Error 2	0.52	Urgente
Error 3	0.52	Urgente
Error 4	0.59	Urgente
Error 5	0.52	Urgente
Error 6	0.49	Urgente

Tabla 6.1: *Anomaly score* de los errores de la Entidad A

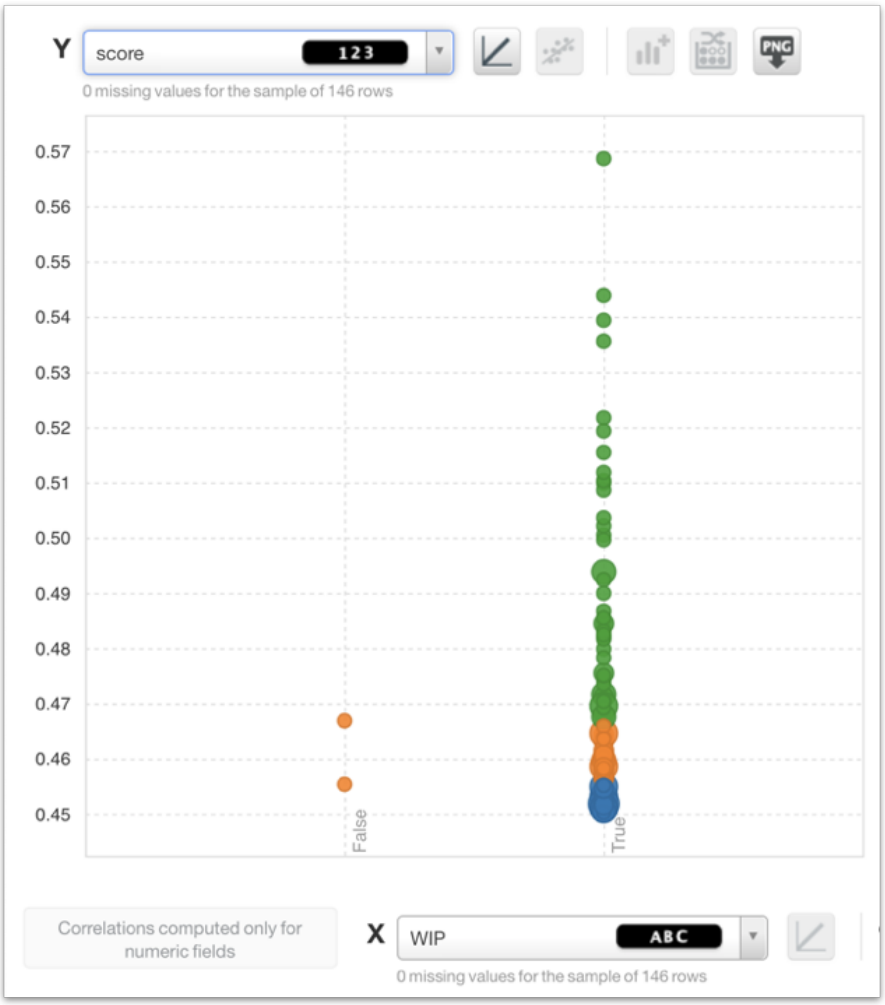

Figura 6.1: Resultado Entidad A

	Registros	Errores	Positivos Reales	Falsos Positivos
Entidad A	2459	6	6	103

Tabla 6.2: Evaluación Entidad A

De esta forma, *precision* y *recall* son:

$$\text{recall} = \frac{TP}{TP + FN} = \frac{6}{103} = 5{,}82\,\%$$

$$\text{precision} = \frac{TP}{TP + FP} = \frac{6}{6} = 100\,\%$$

Los resultados de esta prueba se han considerado satisfactorios, ya que se ha conseguido detectar todos los errores con un modelo entrenado sin ninguna información sobre errores (es un modelo no supervisado). Con resultados como este, se puede considerar la prueba de concepto como un éxito, ya que esto muestra que se pueden utilizar este tipo de modelos para detectar errores en la producción, a pesar de todas las dificultades inherentes a este problema y los datos de los que se disponen. Aún así, en esta prueba se puede ver que hay margen de mejora, ya que hay un porcentaje alto de falsos positivos.

Con resultados como este, ya sería suficiente para validar la prueba de concepto. Aun así, se van a mostrar otros 2 casos que ilustran otros tipos de resultados que se han obtenido.

Entidad B

Este siguiente resultado, correspondiente a la *Entidad B*, se ha conseguido con un modelo entrenado con *4811* registros, y se van a evaluar con 1 error real. Este es un ejemplo de un modelo que de haber estado funcionando, habría sido capaz de detectar el error real, al mismo tiempo que alertar de los mismos registros que el sistema experto. Las scores asignadas por este modelo son las siguientes:

	Anomaly Score	Nivel de Alerta
Error 1	0.65	Alto

Tabla 6.3: *Anomaly score* de los errores de la Entidad B

En la Figura 6.2 se puede ver una visualización similar a la del apartado anterior, que muestra de forma gráfica la distribución de los resultados. Aquí se puede ver cómo se ha marcado el único error como un valor anómalo, así como los dos

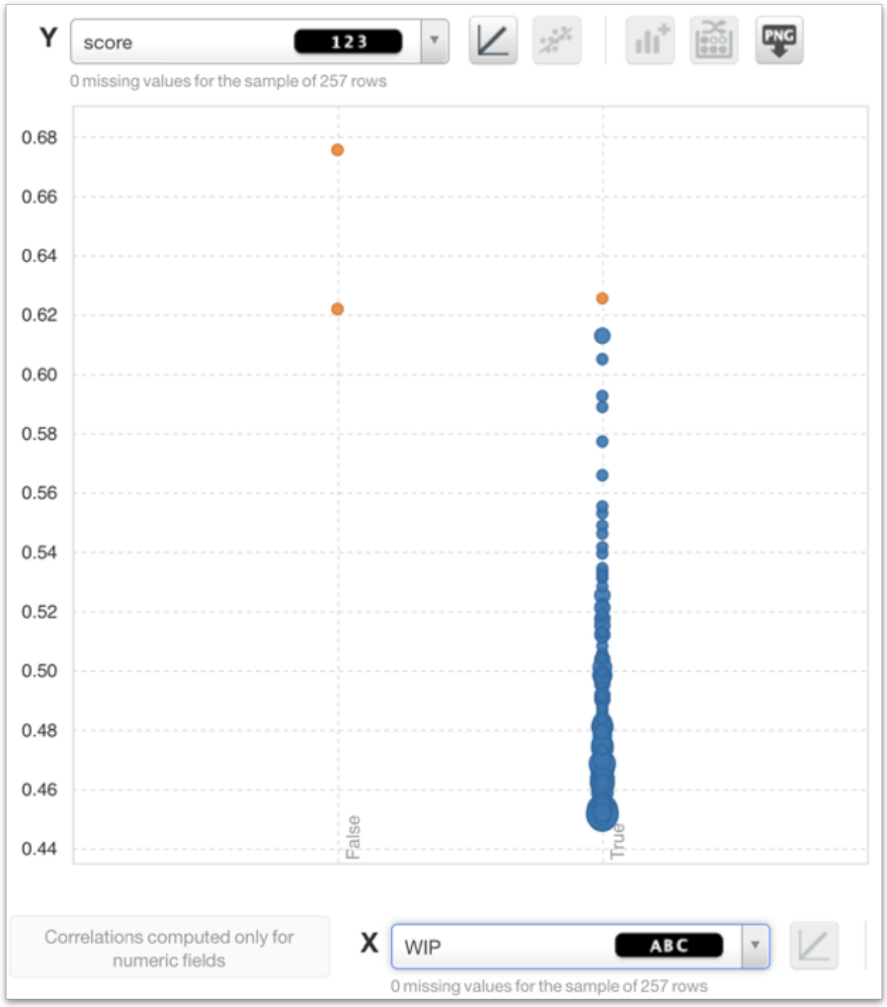

Figura 6.2: Resultado Entidad B

posibles errores que fueron detectados por el sistema experto (columna izquierda, False).

Las métricas de la evaluación se pueden ver en la Tabla 6.4.

	Registros	Errores	Positivos Reales	Falsos Positivos
Entidad B	4811	1	1	0

Tabla 6.4: Evaluación Entidad B

De esta forma, *precision* y *recall* son:

$$\text{recall} = \frac{TP}{TP + FN} = \frac{1}{1} = 100\,\%$$

$$\text{precision} = \frac{TP}{TP + FP} = \frac{1}{1} = 100\,\%$$

Esta prueba, aunque es cierto que tiene una muestra reducida de 1 único error, es un ejemplo del tipo de solución que se buscaba a este problema, donde el modelo permite a los trabajadores revisar los sistemas y solucionar los problemas de forma eficiente.

Este es el tipo de resultados que idealmente se esperan obtener, ya que es un modelo que sería capaz de detectar todos los errores, sin incurrir en muchas falsas alarmas (falso positivo). Con resultados como el del apartado anterior y este se tiene confianza de que la solución planteada en esta prueba de concepto puede servir para solucionar el problemas como el planteado en este caso de uso, y con esto se considera que merece la pena continuar con esfuerzos para refinar estos modelos.

Entidad C

Por último, se muestra también un ejemplo de un modelo que no cumple con los objetivos deseados, y que muestra que, aunque los resultados son prometedores, hay aún mucho margen de mejora. Este es un ejemplo de un modelo que de haber estado funcionando, no habría sido capaz de detectar ningún error. Este es el resultado obtenido entrenado un modelo con *2249* ejemplos, correspondientes a la *Entidad C*, y se van a evaluar 3 errores reales. Las scores asignadas por este modelo son las siguientes:

	Anomaly Score	Nivel de Alerta
Error 1	0.48	Normal
Error 2	0.43	Normal
Error 3	0.45	Normal

Tabla 6.5: *Anomaly score* de los errores de la Entidad A

En la Figura 6.3 se muestran los resultados en un gráfico similar a los de los apartados anteriores, y en ella se puede ver que se han detectado valores no erróneos como anomalías (en verde).

Las métricas de la evaluación se pueden ver en la Tabla 6.6.

De esta forma, *precision* y *recall* son:

$$\text{recall} = \frac{TP}{TP + FN} = \frac{0}{3} = 0\,\%$$

$$\text{precision} = \frac{TP}{TP + FP} = \frac{0}{8} = 0\,\%$$

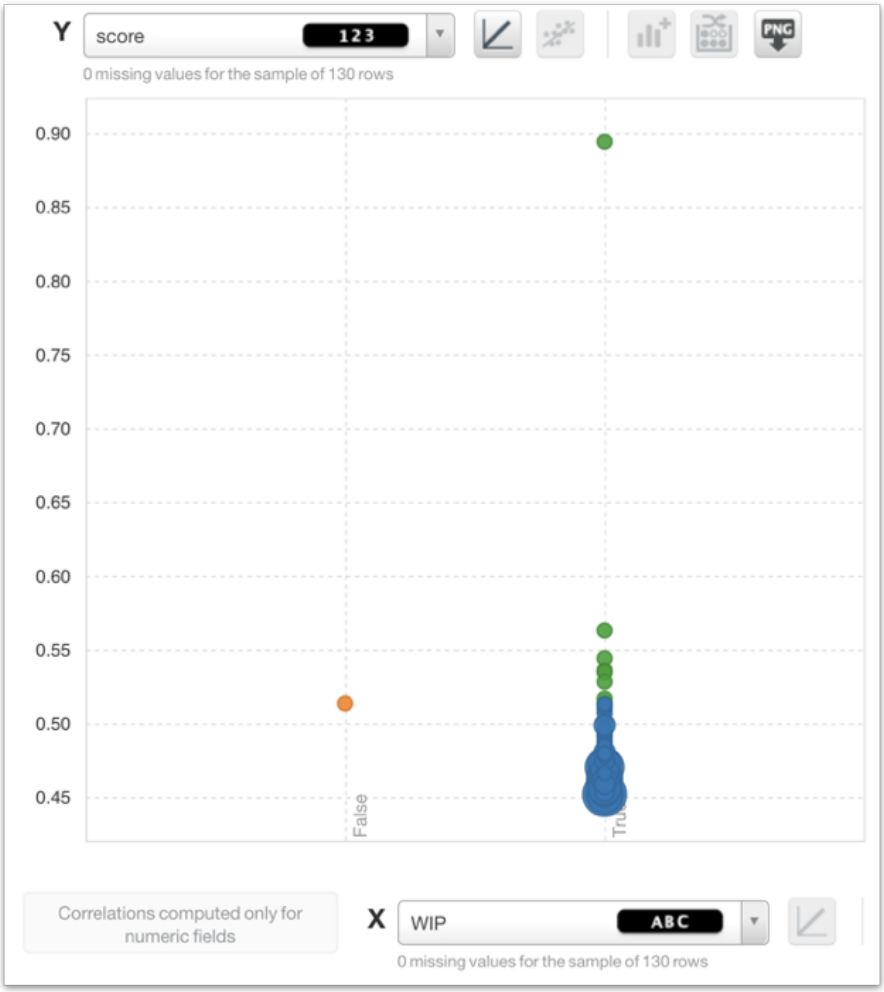

Figura 6.3: Resultado Entidad C

	Registros	Errores	Positivos Reales	Falsos Positivos
Entidad C	2249	3	0	8

Tabla 6.6: Evaluación Entidad C

En esta prueba los resultados no son los esperados, ya que no se han detectado ninguno de los errores. Sin embargo, se puede ver que aun así a los errores se les ha asignado valores que están en la parte alta de la distribución de las *anomaly score*, lo cual hace pensar que con trabajo se puede conseguir mejorar este comportamiento. Este es un ejemplo de un mal resultado, pero como ya se ha comentado, se han visto mejores con otros modelos, con los que se da por validada la hipótesis de

que un workflow basado en *Anomaly Detectors* puede servir para detectar errores.

Este caso de fracaso se utiliza como un ejemplo para motivar los esfuerzos que se van a realizar en el futuro para solucionar los problemas de la metodología actual. Basado en este workflow, se va a optimizar el proceso aprovechando los nuevos datos que van llegando, así como las mejoras que se puedan incluir en la librería.

6.1.3. Importancia de las características

Por último, se va a comprobar la influencia que tienen las características generadas con la librería en los modelos.

Para esto, se va a aprovechar una de las funcionalidades de BigML, que permite la introspección sobre los modelos que genera. Concretamente, esta funcionalidad devuelve una medida de la "importancia" que tiene una característica concreta para la generación de una predicción, o en general en el entrenamiento.

A continuación se expone una muestra de los valores más anómalos de uno de estos *Anomaly Detectors*. Teniendo un *Anomaly Detector* en BigML, se puede generar un informe como el que se muestra en la Figura 6.4, que muestra los 10 registros más anómalos, junto con un registro de qué variables se han tenido en cuenta en cada caso para determinar su condición de anomalía.

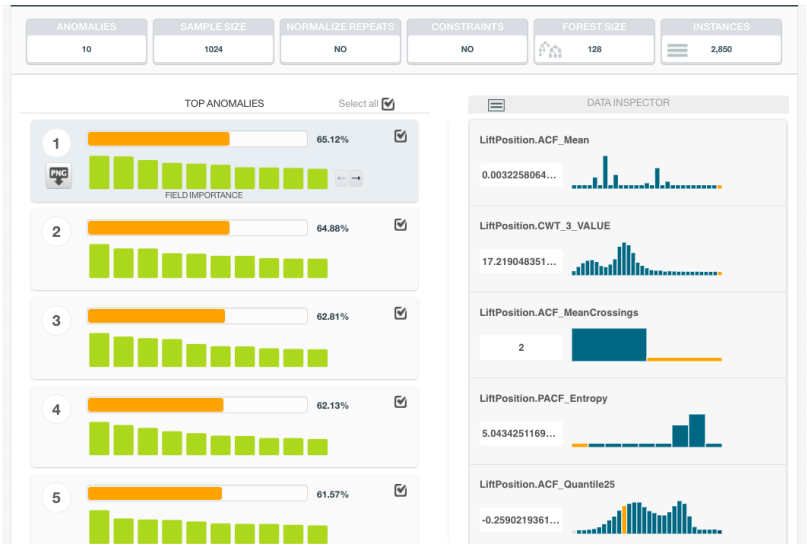

Figura 6.4: Reporte de principales anomalías

En la siguiente tabla se muestran los 10 registros más anómalos encontrados durante el entrenamiento de uno de estos modelos, y para cada uno se muestran las 5 variables más importantes.

	Variable 1	Variable 2	Variable 3	Variable 4	Variable 5
1	LiftPosition .ACF_Mean	LiftPosition .CWT_3_VALUE	LiftPosition .ACF_MeanCrossings	LiftPosition .PACF_Entropy	LiftPosition .ACF_Quantile25
2	LiftPosition .ACF_Mean	LiftPosition .ACF_Quantile75	LiftPosition .ACF_Quantile25	LiftPosition .ACF_MeanCrossings	LiftPosition .PACF_Entropy
3	LiftPosition .ACF_Mean	LiftPosition .PSD_1_LOC	LiftPosition .PACF_Entropy	LiftPosition .ACF_Quantile25	LiftPosition .ACF_MeanCrossings
4	LiftPosition .ACF_MeanCrossings	LiftPosition .PACF_Entropy	LiftPosition .ACF_Mean	LiftPosition .ACF_Quantile25	LiftPosition .PSD_2_LOC
5	Voltage .FFT_2_VALUE	Voltage .ACF_Quantile25	Voltage .DTW_Distance	Var1	Voltage .ACF_Quantile95
6	Var2	Current .PACF_StdDeviation	Voltage .FFT_3_LOC	Voltage .FFT_1_LOC	Voltage .PSD_1_LOC
7	Voltage .FFT_2_LOC	Current .PSD_3_LOC	Voltage .CWT_1_TIME	Current .ACF_Variance	Var3
8	LiftPosition .ACF_RMS	Voltage .FFT_1_LOC	Voltage .ACF_Quantile95	Var4	LiftPosition .DTW_Distance
9	Voltage .ACF_Entropy	Current .ACF_Variance	Current .PSD_2_VALUE	Current .PSD_1_LOC	Voltage .ACF_RMS
10	Var5	Voltage .FFT_1_LOC	LiftPosition .DTW_Distance	Voltage .CWT_1_TIME	Voltage .ACF_StdDeviation

Tabla 6.7: Variables más importantes en anomalías

En esta tabla se puede observar claramente como las variables más importantes a la hora de detectar las anomalías son las variables relacionadas con las señales, y que las características que se extraen con la librería son una parte importante de esta prueba de concepto que con estos resultados se da por válida.

Dados todos estos resultados, se han cumplido los requisitos para dar por válida la prueba de concepto, considerándose un éxito la primera parte de este proyecto. Esto demuestra que se puede utilizar un método basado en detección de anomalías y características de señales para detectar fallos en procesos industriales. Estos resultados han servido también para resaltar algunas de las flaquezas de este enfoque, y se han apuntado los puntos a mejorar para las siguientes fases.

A continuación se van a exponer los resultados relacionados con el benchmark.

6.2. Benchmark

El siguiente paso del proceso de validar los requisitos de la librería consiste en comprobar su capacidad de generalizar los éxitos que se han conseguido con el caso de uso. Esto se hace utilizando el benchmark descrito en el capítulo anterior (ver Subsección 5.3.4), como base para optimizar el proceso de *Feature Engineering* para señales de la librería, eligiendo transformadas útiles y técnicas de extracción más apropiadas. Además, se va a utilizar este benchmark para hacer experimentos que comprueben la utilidad de posibles mejoras a introducir en la librería.

Este benchmark cuenta con 116 datasets, que contienen una señal y una etiqueta, que pueden tener 2 o más clases. Cada dataset pertenece a un "tipo", que

indica el tipo de señal que incluye. Con esto, se puede hacer una estimación de la capacidad de generalizar que tiene, y los tipos de señales que procesa mejor y peor. En la Figura 6.5 La distribución de los tipos en el benchmark, y en ella se ve que hay tipos de datasets de los que se tienen pocas muestras, y esto se tendrá en cuenta a la hora de analizarlo.

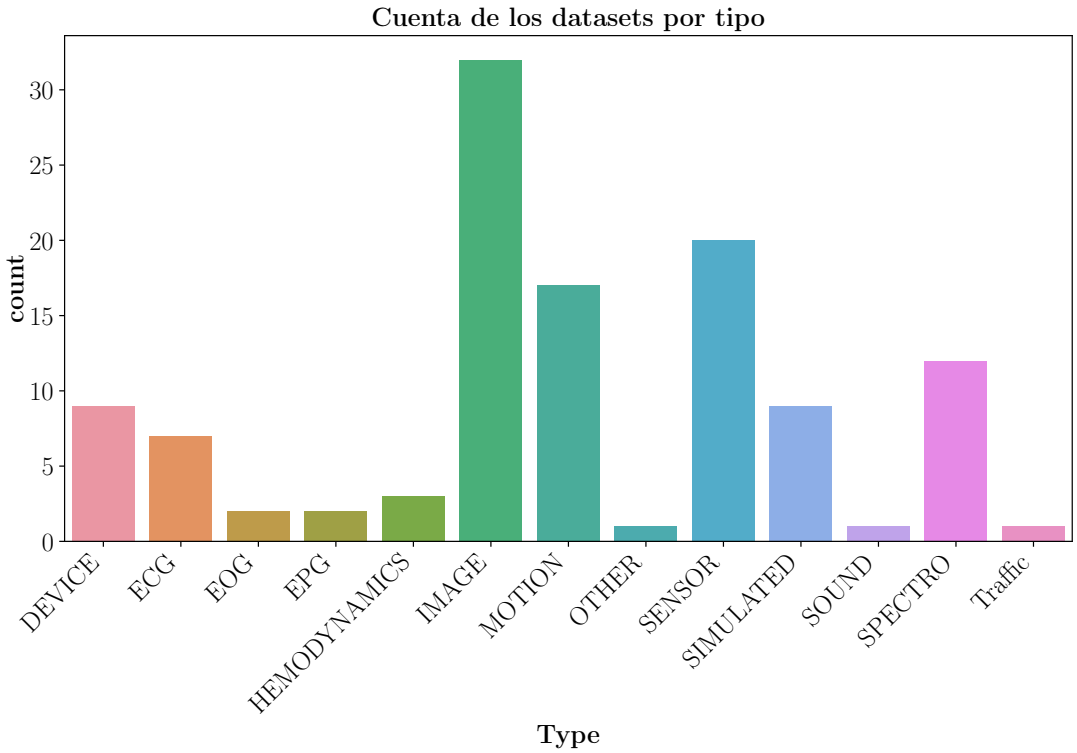

Figura 6.5: Distribución de los tipos de datasets en el benchmark

Los resultados obtenidos en este benchmark se resumen en la Figura 6.6. En ella se puede ver la evolución de la distribución de *accuracy* en los distintos experimentos que se han hecho, y la configuración final.

En ella se puede ver cómo, con las distintas pruebas se ha conseguido una distribución de *accuracy* que tiene menos variabilidad y en media mejores resultados. En las siguientes secciones se explica cada experimento con el razonamiento que se ha seguido, y se muestran los resultados obtenidos más en detalle.

6.2.1. Base

En primer lugar se va ha hecho una evaluación del comportamiento de la implementación de la librería hecha para el caso de uso, sobre la que se va a iterar para

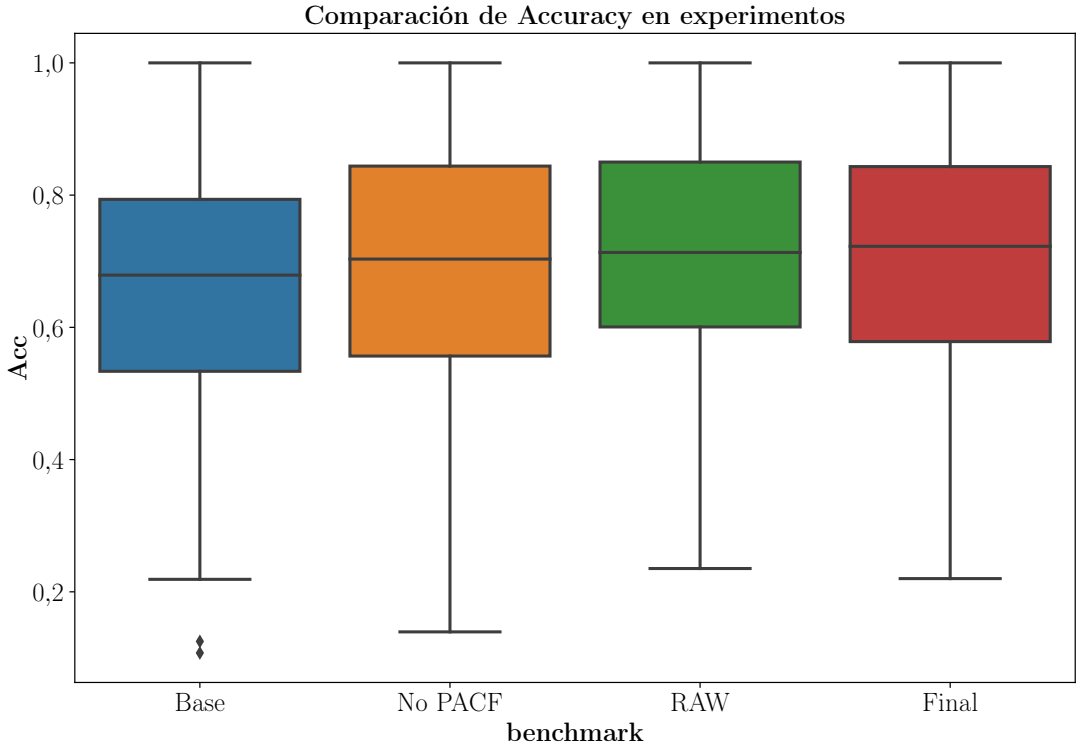

Figura 6.6: Distribuciones de *accuracy* en distintos experimentos

hacer mejoras. En la Tabla 6.8 se incluyen los resultados de la evaluación de todos los datasets del benchmark, incluyendo distintos parámetros y con estadísticos que muestran cómo se distribuyen.

Estadístico	Accuracy	F_measure	ROCAUC	Phi
mean	0.660645	0.624818	0.850502	0.501102
std	0.191454	0.206626	0.116489	0.232987
min	0.107530	0.093000	0.487090	0.007550
25 %	0.533530	0.488880	0.801650	0.348225
median	0.679010	0.655610	0.874090	0.484080
75 %	0.793600	0.773915	0.936895	0.661135
max	1.000000	1.000000	1.000000	1.000000

Tabla 6.8: Resultado de Evaluación de Base

Como se puede ver, con este conjunto de features en función del dataset se pueden conseguir buenos resultados. Este resultado es comparable con otras so-

luciones existentes y más complejas como *tsfeatures* de *R*, que en un benchmark similar [31] consigue *69.4 %* de *accuracy*.

En cuanto a los resultados por tipos, estos se muestran en la Figura 6.7. Ahí se puede ver que hay tipos para los que se consiguen buenos resultados, y tipos con los que no se consiguen demasiado buenos resultados.

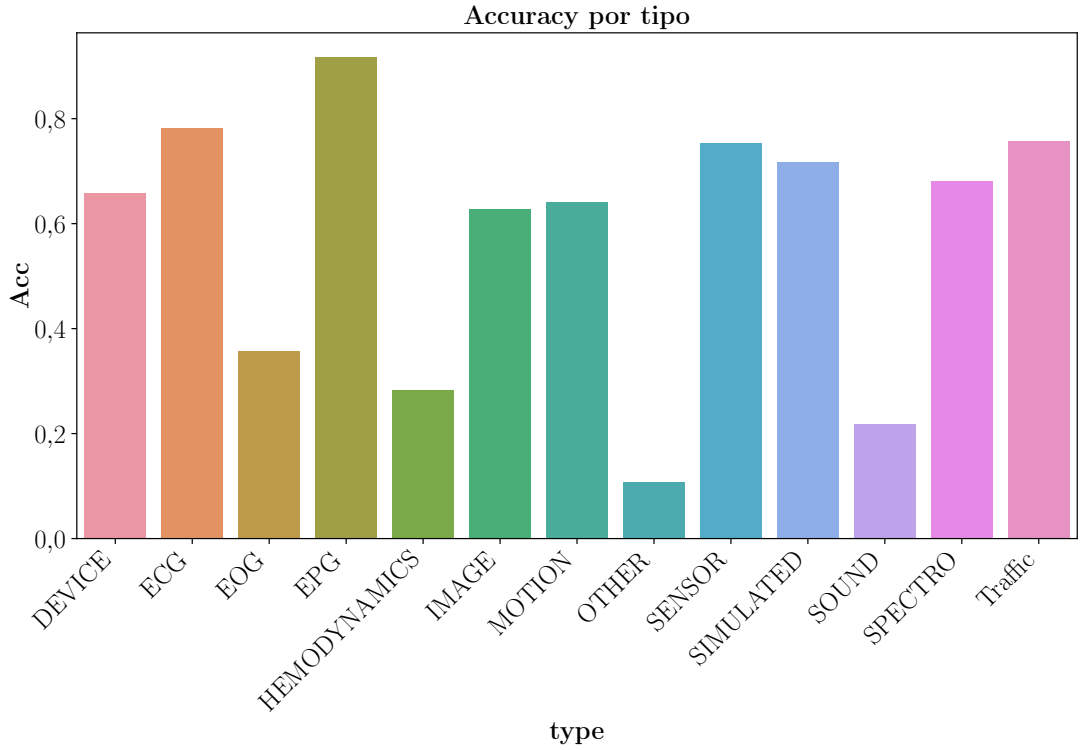

Figura 6.7: Accuracy media por tipo de dataset

Como punto positivo cabe destacar los resultados con señales de sensores y señales espectrales, lo que confirma la hipótesis del caso de uso, y tiene sentido porque se ha diseñado con estas tareas principales en mente. También funciona bien en algunas aplicaciones médicas (*ECG*, *EPG*) y de tráfico, aunque de estos datasets se tienen menos muestras.

Como punto negativo, el tipo *OTHER*. Aunque analizando en profundidad, se ve que este tipo sólo tiene 1 dataset, *Fungi*, que tiene 18 clases y 1 única muestra por clase. Por lo tanto, no es una muestra representativa de un dataset de *Machine Learning*. Otro que tampoco da buenos resultados es *SOUND*, y es que el dataset sufre de un problema similar: sólo tiene 214 muestras, pero 39 clases. Este no es el tipo de datasets con el que este workflow funcione correctamente. Para este tipo de problemas sería necesario usar técnicas como un modelo especializado en detectar

cada una de las clases, pero esto queda fuera del alcance de este benchmark.

A continuación se van a explicar los cambios, y alguno de los experimentos llevados a cabo para mejorar estos resultados.

6.2.2. Selección de características

A continuación se van a presentar los experimentos realizados para seleccionar el mejor conjunto posible de características. Esto incluye probar distintas transformadas, y distintas combinaciones de técnicas de extracción de características sobre estas. También se ha buscado eliminar algunas de las características redundantes.

Para elegir las características más útiles se van a utilizar una técnica similar a la que se ha utilizado en el apartado, aprovechando las medidas de la importancia de las variables que BigML devuelve en sus modelos.

Ignorar PACF

La primera prueba que se ha querido hacer es asegurarse de que la *PACF* es una transformación útil. Esto es importante, ya que como se ha comprobado en la Subsección B.1.3, esta es la transformación más lenta, y cuya implementación tiene una complejidad de ejecución $O(n^2)$, lo cual le hace imposible de utilizar con señales con muchas muestras, típicas en aplicaciones de audio o imágenes.

Para esto, se va a estudiar la importancia de las características generadas por cada una de las transformadas durante el benchmark. Como las transformadas generan múltiples características, se va a estudiar el máximo de la importancia en cada caso.

En primer lugar, se muestran las importancias máximas en el benchmark base. En la Tabla 6.9 se muestran estadísticos sobre la distribución de la importancia a lo largo de todo el benchmark.

Estadístico	FFT	PSD	PACF	ACF	CWT
mean	0.101782	0.148810	0.065927	0.163490	0.130263
std	0.131415	0.156579	0.095344	0.138579	0.153611
min	0.002160	0.008040	0.002820	0.011800	0.012190
25 %	0.025855	0.045870	0.015815	0.061520	0.045175
median	0.050045	0.095630	0.030125	0.126390	0.075330
75 %	0.109680	0.205250	0.063680	0.222790	0.144185
max	0.600000	0.869560	0.558960	0.798710	1.000000

Tabla 6.9: Importancias en el benchmark base

En los estadísticos se puede ver que la *PACF* tiene una importancia sensiblemente menor que el resto de las transformaciones. En la Figura 6.8 se muestra gráficamente la importancia de cada transformada, resaltando la importancia que tiene la *PACF*. Como se puede ver, en la mayoría de casos la importancia es menor que la de otras transformadas (a excepción de un *outlier*, que corresponde con el dataset *Fungi*, que no es una buena muestra), y en muchos de los casos es 0, que quiere decir que la señal era demasiado larga y la *PACF* ni se ha calculado.

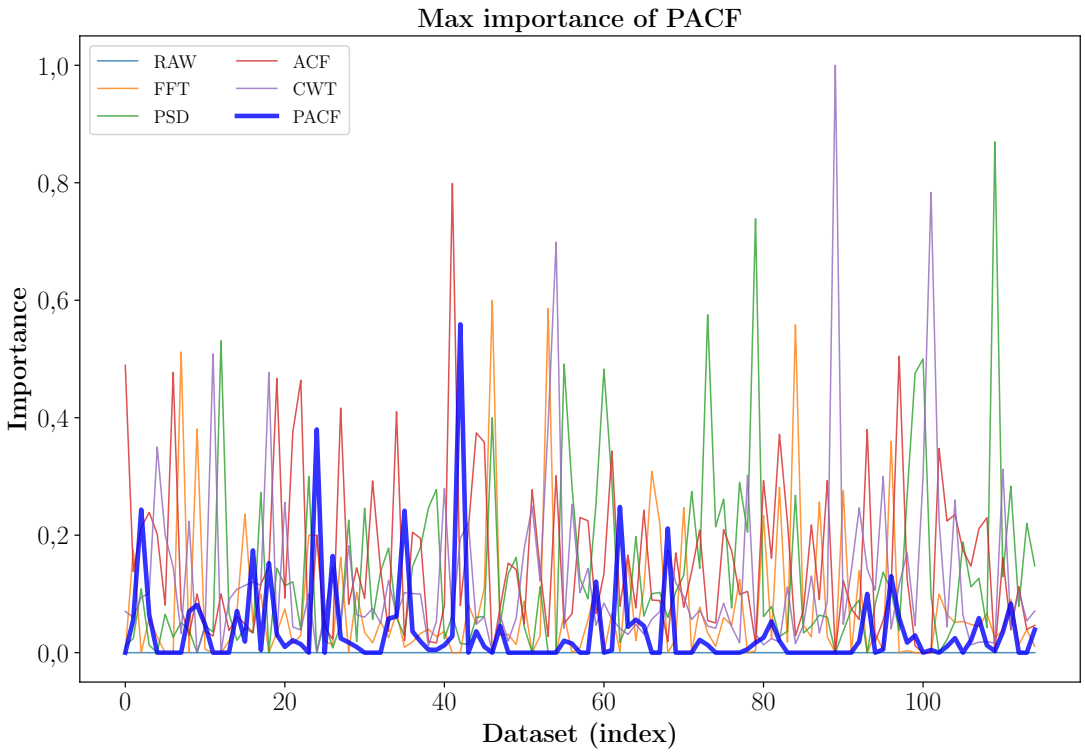

Figura 6.8: Importancia de las transformadas

De esta forma, se ha probado a hacer una ejecución eliminando esta transformada mediante la configuración de la librería. Con esto, se han conseguido los siguientes resultados:

Al comparar con los resultados de la Tabla 6.8, se observa que los resultados son mejores en todos los parámetros de la evaluación. De esta forma, se ha decidido que eliminar esta transformación es una decisión aceptable, ya que no tiene un coste importante en el comportamiento medio, mientras que tiene un coste muy grande en cuanto al tiempo de ejecución.

El resto de experimentos se han ejecutado incluyendo este cambio.

Estadístico	Accuracy	F_measure	ROCAUC	Phi
mean	0.679509	0.644683	0.857109	0.529652
std	0.193644	0.208667	0.111927	0.231792
min	0.139420	0.081360	0.513150	0.009770
25 %	0.556575	0.484970	0.807625	0.350715
median	0.703260	0.669385	0.869450	0.537250
75 %	0.843892	0.787275	0.951970	0.702973
max	1.000000	1.000000	1.000000	1.000000

Tabla 6.10: Resultado de Evaluación sin *PACF*

Estadísticos de la señal original

Uno de los conjuntos de características que se pasaron por alto en la primera implementación fue el utilizar la señal original, sin transformaciones, y extraer características directamente de ella. Esto sigue encajando con el planteamiento de la librería, viendo la señal original como el resultado de una transformación "identidad", que devuelve la entrada. Así, se han incluido características extraídas utilizando la extracción de estadísticos.

En la Tabla 6.11 se muestran los resultados del experimento. En primer lugar se muestra una tabla con los resultados de la evaluación:

Estadístico	Accuracy	F_measure	ROCAUC	Phi
mean	0.706580	0.671104	0.874296	0.568599
std	0.182313	0.203272	0.105605	0.237171
min	0.235230	0.103910	0.569880	-0.099860
25 %	0.600600	0.524685	0.820398	0.382265
median	0.713270	0.681185	0.899580	0.571865
75 %	0.850000	0.839835	0.958305	0.742365
max	1.000000	1.000000	1.000000	1.000000

Tabla 6.11: Evaluación con estadísticos de la señal original

Se observa que, como cabía esperar, el incluir características de la propia señal mejoraría los resultados. Esto se puede ver más claro en la Figura 6.9, donde se muestra la diferencia entre la media de la *accuracy* del benchmark base (azul) y el benchmark "raw", incluyendo las nuevas características.

En aras de la exhaustividad, en la Tabla 6.12 se muestra la importancia de las variables. En ella se puede comprobar que los estadísticos de la señal original son importantes a la hora de crear los modelos.

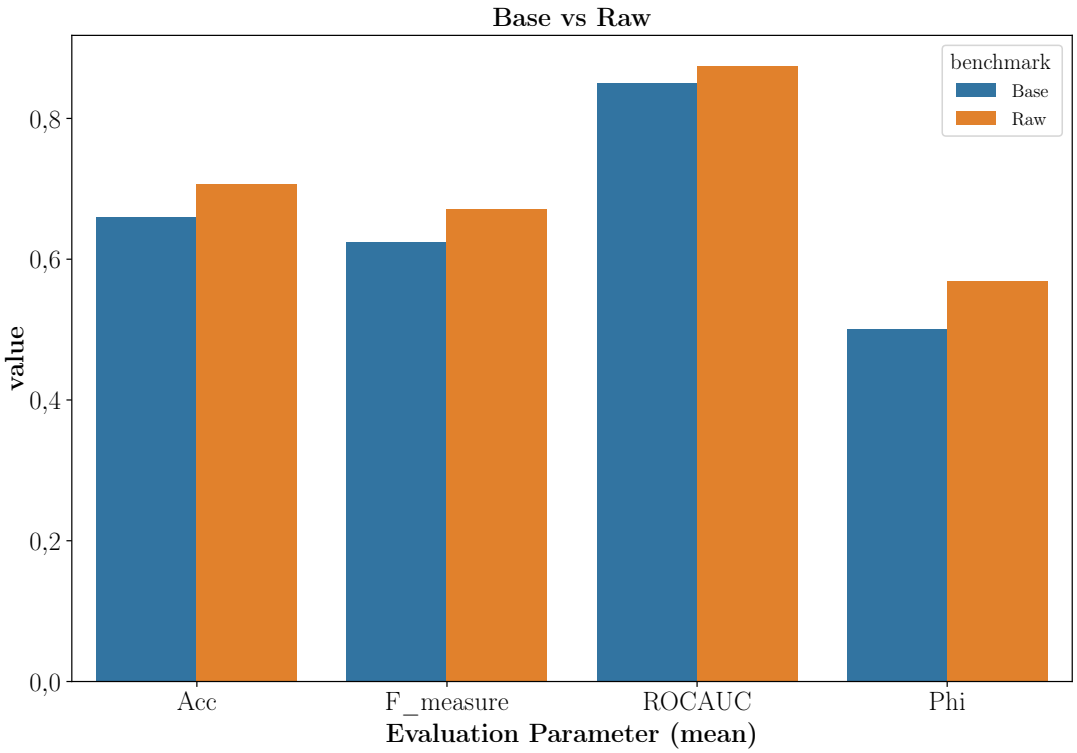

Figura 6.9: Comparación de la *accuracy* media con el benchmark base

Estadístico	RAW	FFT	PSD	ACF	CWT
mean	0.196038	0.058206	0.104165	0.142649	0.125441
std	0.205046	0.080601	0.110628	0.141305	0.180797
min	0.005320	0.000290	0.009790	0.002890	0.007900
25 %	0.079695	0.013110	0.029130	0.040912	0.031450
median	0.133540	0.029510	0.064570	0.087955	0.050000
75 %	0.228797	0.066450	0.120765	0.188240	0.136960
max	1.000000	0.475670	0.552880	0.658490	1.000000

Tabla 6.12: Importancias con las características de la señal original

Redundancia en estadísticos

Una redundancia que se ha detectado en las características generadas es el cálculo tanto de la *varianza* como de la *desviación típica*. Estos dos estadísticos están perfectamente correlados, ya que la desviación típica se puede definir como la raíz cuadrada de la varianza.

De esta forma, es innecesario calcular las dos al mismo tiempo, ya que ofrecen la misma información. Se va a eliminar una de las dos del conjunto de características.

Para esto, se han hecho un benchmark del tiempo que se tarda en calcular las dos, y se ha visto que la diferencia es despreciable entre las dos. Se ha elegido mantener la *desviación típica*, ya que en general va a resultar en valores menores, y además deshacer el camino es más eficiente (una multiplicación es más rápido que una raíz, computacionalmente hablando). Se ha ejecutado un benchmark igual que el de la sección anterior, pero sin generar las características **_ Variance*. El resultado de la ejecución se muestra en la Tabla 6.13.

Estadístico	Accuracy	F_measure	ROCAUC	Phi
mean	0.704211	0.671295	0.872069	0.564724
std	0.178466	0.197685	0.109402	0.233178
min	0.220000	0.121260	0.567330	0.026150
25 %	0.578442	0.527173	0.821325	0.399382
median	0.722565	0.685865	0.902315	0.592630
75 %	0.843152	0.832383	0.955230	0.725863
max	1.000000	1.000000	1.000000	1.000000

Tabla 6.13: Evaluación sin la varianza

Se puede ver que se consiguen los mismos resultados que cuando estaba incluida.

6.2.3. Evaluación

A continuación se han hecho evaluaciones con workflows más complejos, que incluyen algunos pasos extra que se pueden encontrar en un *pipeline* de *Feature Engineering* normal. Estas variaciones utilizan otros algoritmos o recursos externos para intentar mejorar los resultados de las características extraídas.

Distancias

En el conjunto de características, se ha incluido además característica opcional que se calcula como una distancia a una señal de referencia, que es necesario proveer como entrada a la hora de ejecutar las transformaciones. Estas características tienen la ventaja de que son muy intuitivas. Si se define una señal de referencia, calculada con los valores esperados de la señal, cuando se encuentra un registro con una distancia elevada, se puede interpretar fácilmente.

Aprovechando el benchmark, se ha probado la utilidad de estas características. Ya que no se tiene una señal de referencia, se ha simulado una mediante un cálculo de la señal media en el dataset.

En la Figura 6.10 se visualiza una comparación de los parámetros de la evaluación de la librería sin distancias, como en los apartados anteriores, y con los distintos algoritmos de distancia.

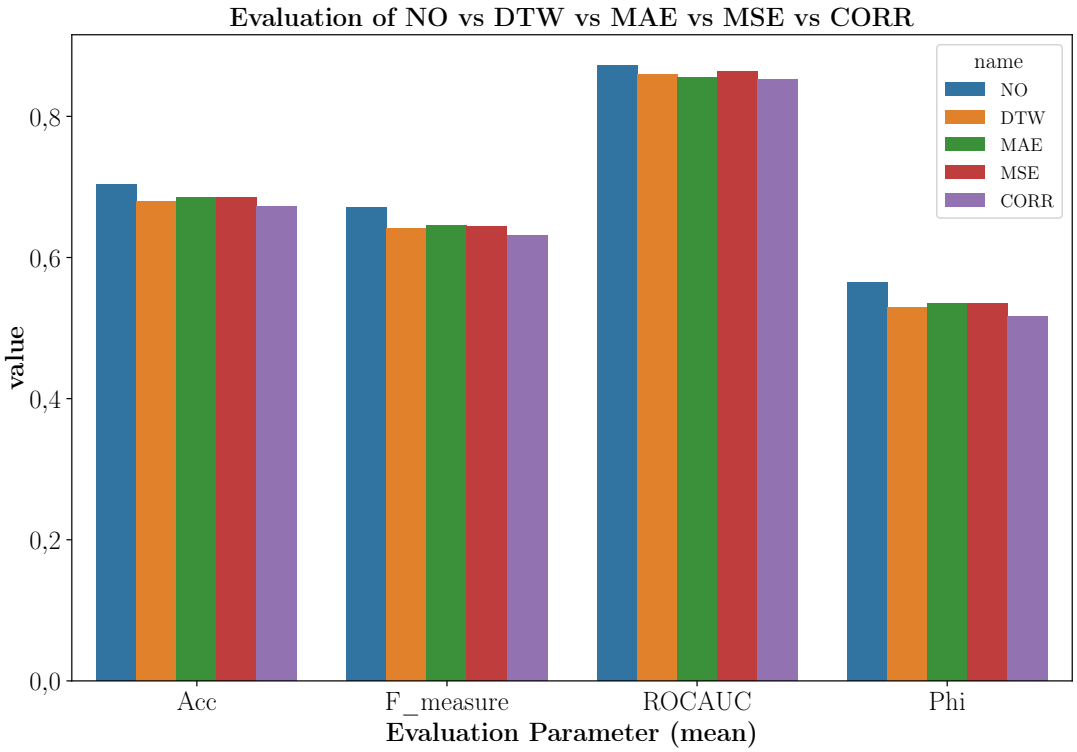

Figura 6.10: Comparación de la *accuracy* media con las distancias

Al ver este que todos los algoritmos de media ofrecen unos resultados similares entre sí, no parece haber una gran diferencia entre ellos. También es destacable que los resultados que ofrecen son ligeramente peores que los resultados sin utilizarlos.

El segundo fenómeno se puede explicar debido a la implementación de la señal de referencia que se ha escogido para este benchmark. En estos problemas, lo ideal es, aprovechando el conocimiento del dominio, poder elegir los registros correctos de cada una de las clases que se consideren "normales", para con ellos generar una referencia. En este caso, se está generando una única señal de referencia por datasets, y esto se comporta peor cuantas más clases haya en el dataset. Además, implementaciones como la *DTW* con la configuración por defecto pueden fallar, y en este caso introducir nulos en los datos, y esto ensucia más los datos.

El caso por el que todos parecen comportarse de forma similar, al no ser una característica que aporte buena información porque la referencia no es valiosa, no se le está dando importancia a la variable, y los resultados no varían. Para ello se

ha hecho una comparación de la importancia media de estas características para sus respectivos benchmarks, que se muestra en la Figura 6.11. En ella se puede ver que en todos los casos la importancia de las distancias son bajas, lo que explica que sus comportamientos sean similares.

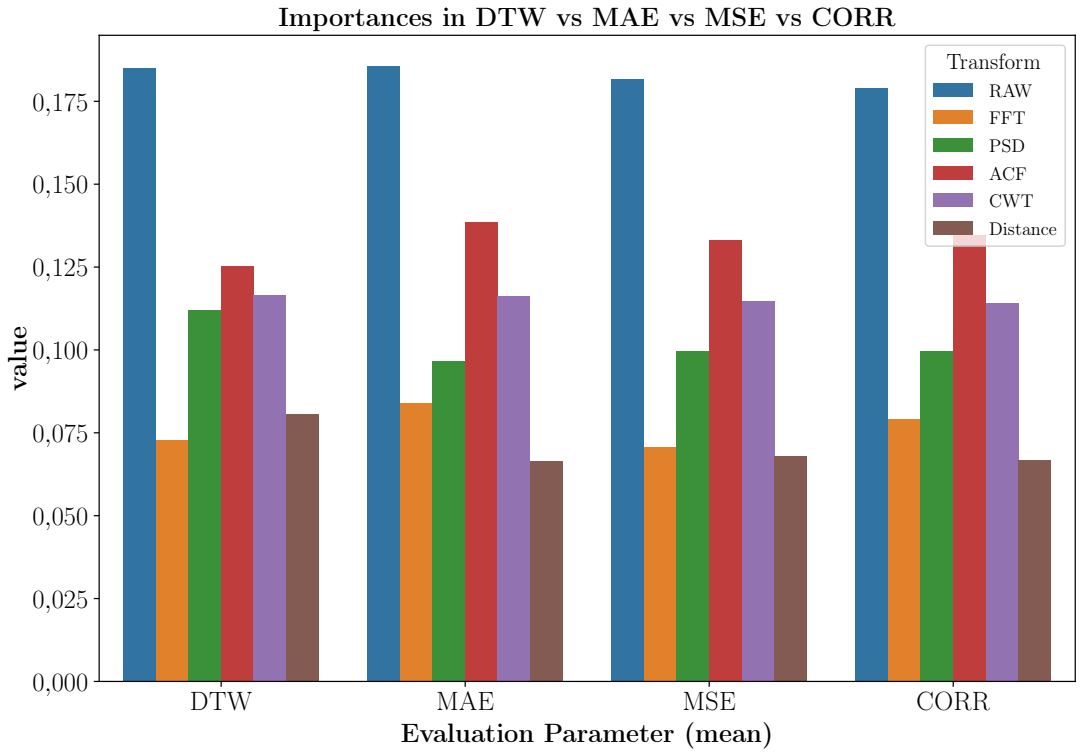

Figura 6.11: Importancias de las distancias distancias

Aunque este no es un resultado favorable a estas características, se ha comprobado en el caso de uso que estas pueden ser de gran ayuda, y pueden ayudar a la explicabilidad de anomalías. Sin embargo, es muy importante el proceso de definición de la referencia, y para eso se requiere de conocimiento del dominio y de los datos, cosa que no se tiene en este benchmark.

Este requisito es lo que ha motivado que estas características sean opcionales, pero que se sigan incluyendo como una posible mejora sobre los procesos por defecto.

PCA

Otra de las técnicas típicas en este tipo de casos en los que se tiene una combinación grande de características más o menos complejas, y con posibles relaciones

entre ellas y redundancias, es hacer *Principal Component Analysis* (*Análisis de Componentes Principales*) o *PCA* con las variables.

Este proceso realiza una proyección de los datos sobre combinaciones lineales de las características originales, de forma que se maximice la varianza de los datos en las primeras componentes. De esta forma, en función de la distribución de los datos, es posible realizar una reducción de las dimensiones del dataset original sin perder mucha información. Esto es siempre una buena idea, especialmente en entornos donde no se disponen de suficientes datos para cubrir un espacio de muchas dimensiones.

Sin embargo, esta transformación tiene un gran inconveniente, y es que las características resultantes pueden ser complicadas de interpretar, aunque no siempre es el caso.

BigML ofrece la opción de incluir un paso de proyección en el workflow. Este paso se puede añadir al código de *WhizzML* de forma sencilla. En la Tabla 6.14 se muestran los resultados de la evaluación añadiendo *PCA* como un paso anterior al modelado:

Estadístico	Accuracy	F_measure	ROCAUC	Phi
mean	0.613236	0.576831	0.802675	0.423496
std	0.201002	0.209191	0.120843	0.230750
min	0.081730	0.026580	0.481480	-0.049380
25 %	0.506790	0.441485	0.721275	0.268782
median	0.640335	0.588960	0.823065	0.435385
75 %	0.747035	0.708540	0.884242	0.531215
max	0.981340	0.974570	0.999070	0.956930

Tabla 6.14: Evaluación con PCA

En la tabla se puede ver que se han conseguido resultados sensiblemente peores a los del resto de benchmarks. A primera vista puede ser sorprendente. Sin embargo, es un fenómeno que tiene sentido.

PCA es una técnica que utiliza combinaciones lineales sobre las características originales. Esto es algo que puede ser beneficioso, si se está trabajando con un problema con propiedades que son lineales. No obstante, no todos los problemas con los que se trabaja son de estas características, y esto se espera que se manifieste en mayor manera en problemas donde se están utilizando señales, con transformaciones no lineales como características.

Aun así, haciendo un estudio pormenorizado se ha encontrado que aplicando *PCA* se han mejorado los resultados sobre la base en 34 de 116 datasets que forman el benchmark (29, 3 % del dataset). Se observa que este dataset existen algunos

problemas para los que esta transformación es beneficiosa. En la Tabla 6.15 se muestran los datasets con mayor mejoría en *F_measure* al aplicar *PCA*.

name	Δ Accuracy	Δ F_measure	Δ ROCAUC	Δ Phi
Worms	0.07792	0.21101	-0.02181	0.22177
PhalangesOutlinesCorrect	0.06527	0.15685	0.04945	0.17252
Wine	0.14815	0.15004	0.17284	0.29556
FordA	0.12576	0.12783	0.06929	0.24565
EthanolLevel	0.09200	0.10288	0.08334	0.12066

Tabla 6.15: Top 5 mejoras con *PCA*

En esta tabla se ven algunos resultados prometedores, como en son el caso de *Worms* que muestra una mejoría de 0.21101 en *F_measure*, o *Wine*, que tiene una mejora de 0,14815 % en *accuracy*.

La conclusión a la que se ha llegado es que, como con otras muchas técnicas en este campo, los beneficios de aplicarla o no dependen de las propiedades de los datos con los que se está trabajando, y es necesario hacer pruebas antes de elegir una opción como *PCA*. También es importante tener en cuenta las desventajas que conlleva aplicarlas, como son en este caso la complejidad añadida en el preprocesado, así como la pérdida de una parte importante de la interpretabilidad del modelo.

6.3. Resultado final

Tras todas las mejoras introducidas con la ayuda de los experimentos de benchmarking expuestos en el apartado anterior, se ha llegado a un conjunto de características cuya evaluación se ha considerado favorable, y que cumplen con los requisitos y objetivos marcados al inicio de este proyecto.

En la Tabla 6.16 se muestra un resumen que explica las características elegidas:

Transformación	Extracción	Ejemplo
RAW	Estadísticos	Test.RAW_Median
FFT	Top K picos	Test.FFT_1_VALUE
PSD	Top K picos	Test.PSD_1_FREQ
ACF	Estadísticos	Test.ACF_StdDeviation
CWT	Top K picos	Test.CWT_1_TIME

Tabla 6.16: Resumen de las características

Con este conjunto de características, se consiguen los siguientes resultados (ver Subsubsección 6.2.2), que se muestran en la Tabla 6.17.

Estadístico	Accuracy	F_measure	ROCAUC	Phi
mean	0.704211	0.671295	0.872069	0.564724
std	0.178466	0.197685	0.109402	0.233178
min	0.220000	0.121260	0.567330	0.026150
25 %	0.578442	0.527173	0.821325	0.399382
median	0.722565	0.685865	0.902315	0.592630
75 %	0.843152	0.832383	0.955230	0.725863
max	1.000000	1.000000	1.000000	1.000000

Tabla 6.17: Evaluación conjunto final

Las diferencias con la base se muestran en la siguiente Tabla 6.18.

Estadístico	Δ Accuracy	Δ F_measure	Δ ROCAUC	Δ Phi
mean	0.043566	0.046478	0.021567	0.063622
std	-0.012988	-0.008941	-0.007087	0.000191
min	0.112470	0.028260	0.080240	0.018600
25 %	0.044912	0.038293	0.019675	0.051157
50 %	0.043555	0.030255	0.028225	0.108550
75 %	0.049552	0.058468	0.018335	0.064728
max	0.000000	0.000000	0.000000	0.000000

Tabla 6.18: Comparación conjunto final vs base

Estos resultados significan una mejora de la *accuracy* media de 66,06 % a 70,42 %. Se observan mejoras similares en el resto de métricas. Además, cabe destacar que, observando el resto de estadísticos se puede ver que esta configuración es más estable, consiguiendo una desviación típica menor en las métricas. Estas mejorías se muestran gráficamente en la Figura 6.12.

En la Figura 6.13 se muestran cómo se distribuyen los resultados de la evaluación en función del tipo de dataset. En ella se ve una distribución similar a la que se mostró en la Figura 6.7, aunque con mejorías en general. En particular, se han conseguido mejoras notables en los tipos que peor se comportaban, destacando la categoría *OTHER*, aunque sigue sin dar demasiada información real ya que al haber pocas muestras se espera una gran varianza en los resultados. Aun así es una muestra de que este proceso ha mejorado los resultados y la capacidad de generalizar de la librería, y es un buen proceso a seguir explorando en el futuro.

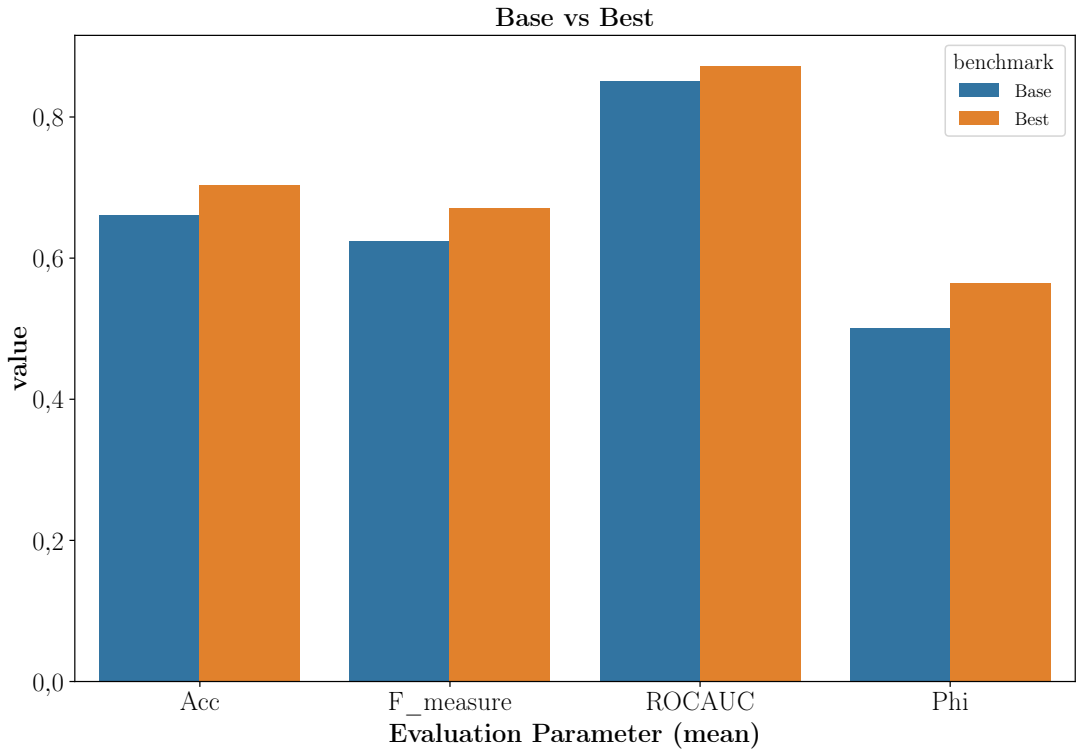

Figura 6.12: Comparación gráfica de conjunto final vs base

Estos resultado se consideran satisfactorios, ya que se ha conseguido una librería que ha demostrado por un lado tener utilidad en un caso de uso real industrial, y al mismo tiempo demostrar que generaliza a otros casos de uso, obteniendo con su configuración más básica resultados que compiten con previos benchmarks de otras soluciones (*tsfeatures*, 69 % *accuracy media* en un benchmark similar).

Con los resultados expuestos hasta ahora, se da por validado el objetivo de utilidad que se marcó en el Sección 4.2. El segundo objetivo importante es el de la usabilidad. Este, aunque es más difícil de medir ya que es una propiedad subjetiva, se puede aproximar observando el proceso que se requiere para utilizar la librería.

En el momento realizar esta evaluación, la librería no está disponible en un repositorio público, ni está integrada en una plataforma como BigML, por lo que el acceso a la librería aún no es tan sencillo como se espera.

Sin embargo, si se tiene acceso a la librería, su instalación es sencilla, con un simple *pip install bigml-proteus*. Además, una vez instalada, para poder utilizarlo no requiere más que importarlo y llamar a una función, aportando como entradas una señal y un nombre. Un ejemplo de este uso se muestra en el Snippet 6.1. Además de esto, se permite la extensión mediante un diccionario de configuración,

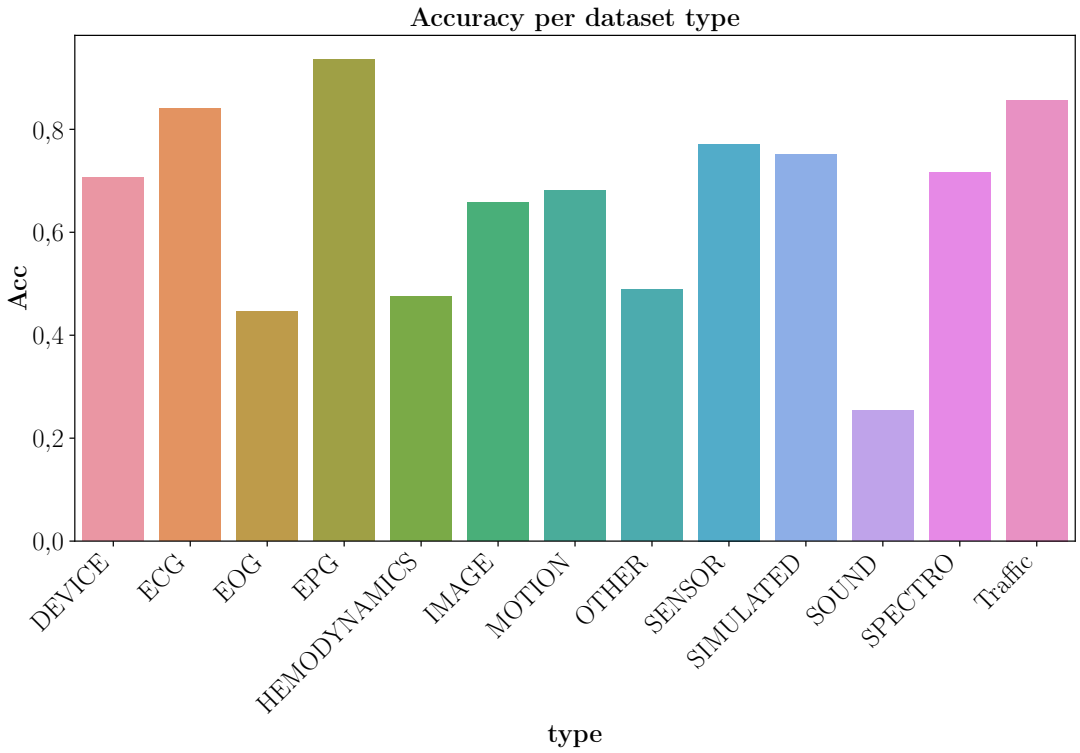

Figura 6.13: Comparación gráfica de la evaluación por tipo de dataset

lo que permite extender y optimizar el comportamiento de la librería.

Con esto, este objetivo se considera validado. Desde el punto de vista de la eficiencia, es necesario hacer un benchmarking exhaustivo de los tiempos de la ejecución con perfilados. Para esta prueba de concepto, se ha conseguido mantener unos niveles de eficiencia que mantenían la ejecución del benchmark de evaluación en un tiempo razonable, (por debajo de 30 minutos en total). Algunos de estos datasets contenían señales de audio u otras señales con miles de muestras, estas se han procesado sin problema. De todas maneras, este siempre será una propiedad que se puede mejorar, encontrando optimizaciones y eliminando redundancias en el proceso.

Por último, cabe destacar que se ha mantenido un nivel aceptable de interpretatividad en las características generadas, ya que, aunque en su formulación son complejas, todas tienen una interpretación que las acerca al usuario menos experto. Además, son transformaciones bien conocidas, sobre las que hay documentación para poder ayudar a la explicación. Aun así, se va a incluir en la distribución de la librería una recopilación de recursos para facilitar la labor de encontrar documentación que facilite la interpretación de los resultados.

Capítulo 7

Conclusiones

Al estudiarse el estado del *Machine Learning* en la actualidad, en especial el estado de su implantación a lo largo de la industria, se han encontrado varios fenómenos que se han considerado dignos de mención.

En primer lugar, se puede observa que las técnicas tradicionales como *Árboles de Decisión*, *Ensembles* o *Regresiones lineales y logísticas* están a la orden del día. Estas soluciones ofrecen unos resultados en una gran variedad de aplicaciones, en las que no se dispone de millones de muestras, tiempo o recursos para entrenar complejas *Redes Neuronales* u otras técnicas de *Deep Learning*, a pesar de ser estos últimos los que reciben más atención en entornos académicos y populares.

Además, estas soluciones se llevan estudiando durante décadas y permiten la introspección de las decisiones que ha tomado un modelo, para poder analizarlos en busca de problemas, prejuicios u otras complicaciones inherentes al modelado de datos *Machine Learning*. Conforme se solucionan los más típicos y sencillos (requisito indispensable para conseguir la adopción del *Machine Learning* en entornos empresariales) se van explorando otros casos de uso en los que aprovechar estas técnicas.

En muchos de estos nuevos campos, aparecen tipos nuevos de datos más complejos, entre los que destacan las señales. A la hora de procesar este tipo de datos, fuera de entornos muy específicos, se ha observado una falta de herramientas que faciliten su manejo y lo automaticen para que sean accesibles a gente que no es experta en el procesado de señales.

7.1. Librería

La misión de este proyecto era desarrollar una herramienta para hacer accesible el procesado de señales con fines de modelado, es decir, el *Feature Engineering* de señales. Para maximizar el impacto que pueda tener, el objetivo principal es diseñar

una herramienta que fuera útil, pero fuera accesible a personas no expertas.

7.1.1. Simplicidad

La lección más importance que se ha aprendido durante el desarrollo de este proyecto es que *la simplicidad es difícil*. Esto se observa con especial claridad cuando se está intentando acercar un tema de naturaleza compleja, como es el análisis y modelado de datos complejos como son las señales. A lo largo de este proyecto se han encontrado muchos momentos en los que hay que tomar una decisión que conllevaba elegir entre una u otra.

Por ejemplo, esto ha sido una preocupación constante a la hora de elegir las transformaciones. Hubiera sido fácil a la hora de diseñar la librería cientos de características, mezclando decenas de transformaciones con aplicaciones concretas en mente, y probablemente se hubiesen conseguido mejores resultados. Sin embargo, esto conllevaría sacrificar la capacidad de introspección sobre las características generadas, y las decisiones de los modelos que se creen con estas. Con esta librería se ha escogido un grupo reducido de transformaciones, manteniendo la complejidad del sistema, y al mismo tiempo de la implementación controlada.

Esta elección ha requerido de un seguimiento más exhaustivo de las evaluaciones que con ellos se consiguen, para así mantener unos estándares de calidad y de utilidad que los usuarios esperan a la hora de confiar en una herramienta con estos fines.

7.1.2. Desarrollo

Otro conjunto de objetivos que se oponían era el de mantener una calidad de código elevada, al mismo tiempo que se conseguía un desarrollo ágil.

Durante el desarrollo del proyecto se ha comprobado que el uso de las buenas prácticas es beneficioso incluso con proyectos en fase de rápido desarrollo e iteración. Entre ellas se incluye una fase de diseño previo intensivo, en la que se tomen decisiones sobre la arquitectura, estructura y tecnologías a utilizar. Es importante dedicarle el tiempo necesario a este proceso, hacer un estudio de las opciones disponibles, teniendo en cuenta las necesidades del proyecto, así como de las fortalezas del equipo. Al mismo tiempo, aunque este diseño va a guiar y acelerar el proceso de desarrollo, es importante admitir que es difícil prever todos los requisitos de antemano, y las necesidades del proyecto pueden evolucionar con el tiempo, y el diseño tiene que evolucionar al mismo tiempo.

Otra práctica que ha sido muy beneficiosa es el uso extensivo de *tests*. A pesar de que conlleven un obvio coste inicial a la hora de introducir nuevo código (escribir más código de test), al mismo tiempo la confianza que dan a la hora de detectar

errores hace que introducir mejoras y cambios sea muy sencillo. Esta ha sido una de las claves a la hora mantener la calidad y velocidad en el desarrollo.

La otra clave ha sido el uso de *revisiones de código*. El tener una persona ajena al código particular que se acaba de escribir revisando los cambios, y con ellos, las decisiones de implementación que se toman con cada parche, permite evitar errores, y encontrar soluciones mejores a problemas que con un único desarrollador no sería posible. Con las herramientas de control de versiones actuales como *Git* y *Github*, se puede mantener una cultura en los equipos de uso de *revisiones de código*, al mismo tiempo que no se pierde tiempo en esperas, gracias al uso de ramas y sistemas de unión de cambios automáticos.

7.1.3. Benchmark

Evaluar la capacidad de generalización de esta herramienta ha sido un reto. La transformación de señales para modelado de *Machine Learning* no es un tema popular de estudios académicos, y no existe una solución estándar para probar las distintas herramientas, similar a los que existen en otros campos donde hay un sistema estándar común que se utiliza como vara de medir y comparar las nuevas soluciones e innovaciones, como son *WordNet* [33] o *ImageNet* o [34] para tareas de *NLP* y reconocimiento de imágenes, respectivamente.

La base de datos de *UEA/UCR* [32] hacen esta labor mucho más sencilla, y el hecho de que existan algunos estudios en esta materia con los que hacer una comparación ha sido de gran ayuda a la hora de poner en perspectiva la estimación de la utilidad.

Tener un sistema de benchmarking con el que poder hacer una estimación de los resultados es clave a la hora de tomar decisiones basadas en datos, y que en última instancia tienen la mejor probabilidad de éxito. Aun así, es importante ser conscientes de las limitaciones de estos sistemas, y no tener fe ciega en estas pruebas. El sentido común debe ser en última instancia el que valore la decisión final.

Es necesario más trabajo y más estudio en este campo, para poder tener una buena base de soluciones, planteamientos y resultados sobre la que construir nuevas herramientas y nuevo conocimiento.

7.2. Caso de uso

El caso de uso de este proyecto ha sido el catalizador que ha llevado a la identificación de la necesidad de implementar una solución de *Feature Engineering* basada en señales.

A lo largo del desarrollo de la librería, el disponer de un caso de uso concreto, de un *early adopter* de la tecnología, ha permitido guiar el desarrollo y ha permitido acelerar el proceso y evitar tomar decisiones que lo beneficiaban al usuario.

Esto también a servido para validar en última instancia la utilidad de la herramienta, y su potencial en un caso de uso real, y cuáles son algunos de las dificultades a las que hay que enfrentarse a la hora de desarrollar un *pipeline* de *ETL*, especialmente cuando se tienen que procesar datos complejos.

Algunas de las lecciones que se han aprendido en este aspecto son:

- No hay que fiarse de que los datos vengan limpios de origen. A la hora de desarrollar una librería hay que tener especial atención a irregularidades en los datos de entrada.
- Las integraciones siempre son complejas, y son puntos de fallos. Hay que intentar mantener el número de sistemas externos necesarios para hacer funcionar una solución.
- Mantener una auditoría de los sistemas es importante, para entender cómo se comportan todas las partes de un sistema. Esto es especialmente importante para solucionar errores, y prevenirlos
- Es importante probar los sistemas en un entorno que simule el entorno de producción, pero que esté controlado. De esta forma se pueden detectar errores de forma que no produzcan costes innecesarios, y reduzcan el tiempo necesario para solucionarlos.

7.3. Trabajos futuros

En este proyecto se ha explorado un campo que tiene potencial de tener un impacto real, aunque aún queda mucho trabajo por delante. Este campo requiere del establecimiento de un sistema de benchmarking estandarizado que permita probar no sólo algoritmos, sino técnicas de *Feature Engineering*. Para ello se debe tener acceso abierto a señales de casos de uso variados, distintas propiedades y orígenes, pero que también estén limpias de forma que los desarrolladores e investigadores se puedan centrar en la tarea en cuestión, y no en la limpieza de datos.

El repositorio de *UEA/OCR* es una buena primera solución, aunque no es ideal para tareas de transformación. En este proyecto se ha realizado una implementación que sirve como una primera aproximación a este problema, pero requiere más trabajo, y más estudios que lo utilicen y formen una base de conocimiento.

Se propone adaptar el sistema de benchmarking para utilizar técnicas de evaluación más robustas que reduzcan la probabilidad de incurrir en *overfitting*, es

decir, tomar decisiones que mejoren la evaluación sobre el conjunto de datos del que se dispone pero que no generaliza bien a nuevos casos. Este fenómeno puede ocurrir al tomar decisiones basadas en resultados de modelado con unos conjuntos de datos cuya división no es correcta y está sesgada debido al muestreo realizado a la hora de hacer la división en train y test por los creadores de los datasets. Este es un problema al que con la metodología actual se es susceptible, ya que estos son los conjuntos que se han utilizado para evaluar y tomar decisiones.

Un ejemplo de una técnica que reduce este problema es el uso de *3-way split*, que consiste en dividir los datos en conjuntos de train, test y además separar un conjunto de validación que no se utiliza para tomar decisiones, y sólo sirve para hacer una evaluación final sobre datos no vistos por el modelo, de forma que se obtiene una estimación de su capacidad de generalizar a datos que no se han tenido en cuenta anteriormente para entrenar o tomar decisiones. La técnica más aceptada hoy en día es el uso de *Cross Validation*, que consiste en hacer múltiples divisiones diferentes sobre los mismos datos, y realizar el modelado con todas estas divisiones y agregar los resultados. De esta forma se consigue una estimación más realista de la capacidad de generalización del proceso.

El problema del uso de estas técnicas es que requieren de mayores cantidades de datos para dar estimaciones más precisas, y esto no es algo que algunos de los datasets elegidos cumplía. Además, técnicas como *Cross Validation*, que son no deterministas, dificultan la reproducción de los resultados, y es necesario utilizar y compartir las *seed* que se utilizan para los generadores pseudo-aleatorios.

En cuanto las implementaciones que a este proyecto respectan, a pesar de que tanto la implementación de la librería como el caso de uso se han considerado un éxito, ambos se encontraban en la fase de prueba de concepto. Esto quiere decir que es necesario seguir trabajando y seguir desarrollando las dos soluciones antes de poder utilizarlas en un entorno de producción real.

Desde el punto de vista de la librería, algunos de los siguientes pasos son:

- Integración con BigML, y/o otras herramientas de modelado
- Estudio de nuevas transformaciones y otras técnicas de extracción de características
- Mejorar la eficiencia de las implementaciones
- Estudiar la librería en más casos de uso reales, y ver los posibles casos extremos en los que puedan aparecer errores
- Mejoras la capacidad de configuración de la librería, dando acceso directo a todas las opciones de las librerías escogidas
- Uso de *property based testing* para mejorar la exhaustividad de los tests

Por otro lado, en cuanto al caso de uso:

- Integrar las mejoras hechas a la librería
- Optimizar el workflow de detección de errores, reduciendo el número de falsos positivos al máximo posible. Algunas de las técnicas a explorar
  - Incluir el feedback de los expertos en el entrenamiento de los modelos
  - *Data Augmentation* para mitigar la reducida muestra de los errores
- Escalar la solución para aumentar el *throughput*. Para ello hay que mejorar la eficiencia de los sistemas de procesado, y permitir la división del procesamiento entre workers distribuidos.

Apéndice A

Datasets utilizados en el Benchmark

En la tabla siguiente se muestra una descripción de los datasets utilizados para el benchmark descrito en Subsección 5.3.4.

Dataset	Train Size	Test Size	Classes	Type
ACSF1	100	100	10	DEVICE
Adiac	390	391	37	IMAGE
ArrowHead	36	175	3	IMAGE
Beef	30	30	5	SPECTRO
BeetleFly	20	20	2	IMAGE
BirdChicken	20	20	2	IMAGE
BME	30	150	3	SIMULATED
Car	60	60	4	SENSOR
CBF	30	900	3	SIMULATED
Chinatown	20	345	2	Traffic
ChlorineConcentration	467	3840	3	SIMULATED
CinCECGtorso	40	1380	4	ECG
Coffee	28	28	2	SPECTRO
Computers	250	250	2	DEVICE
CricketX	390	390	12	MOTION
CricketY	390	390	12	MOTION
CricketZ	390	390	12	MOTION
Crop	7200	16800	24	IMAGE
DiatomSizeReduction	16	306	4	IMAGE
DistalPhalanxOutlineAgeGroup	400	139	3	IMAGE
DistalPhalanxOutlineCorrect	600	276	2	IMAGE
DistalPhalanxTW	400	139	6	IMAGE
DodgerLoopDay	78	80	7	SENSOR
DodgerLoopGame	20	138	2	SENSOR

DodgerLoopWeekend	20	138	2	SENSOR
Earthquakes	322	139	2	SENSOR
ECG200	100	100	2	ECG
ECG5000	500	4500	5	ECG
ECGFiveDays	23	861	2	ECG
ElectricDevices	8926	7711	7	DEVICE
EOGHorizontalSignal	362	362	12	EOG
EOGVerticalSignal	362	362	12	EOG
EthanolLevel	504	500	4	SPECTRO
FaceAll	560	1690	14	IMAGE
FaceFour	24	88	4	IMAGE
FacesUCR	200	2050	14	IMAGE
FiftyWords	450	455	50	IMAGE
Fish	175	175	7	IMAGE
FordA	3601	1320	2	SENSOR
FordB	3636	810	2	SENSOR
FreezerRegularTrain	150	2850	2	SENSOR
FreezerSmallTrain	28	2850	2	SENSOR
Fungi	18	186	18	OTHER
GunPoint	50	150	2	MOTION
GunPointAgeSpan	135	316	2	MOTION
GunPointMaleVersusFemale	135	316	2	MOTION
GunPointOldVersusYoung	135	316	2	MOTION
Ham	109	105	2	SPECTRO
HandOutlines	1000	370	2	IMAGE
Haptics	155	308	5	MOTION
Herring	64	64	2	IMAGE
HouseTwenty	34	101	2	DEVICE
InlineSkate	100	550	7	MOTION
InsectEPGRegularTrain	62	249	3	EPG
InsectEPGSmallTrain	17	249	3	EPG
InsectWingbeatSound	220	1980	11	SENSOR
ItalyPowerDemand	67	1029	2	SENSOR
LargeKitchenAppliances	375	375	3	DEVICE
Lightning2	60	61	2	SENSOR
Lightning7	70	73	7	SENSOR
Mallat	55	2345	8	SIMULATED
Meat	60	60	3	SPECTRO
MedicalImages	381	760	10	IMAGE
MiddlePhalanxOutlineAgeGroup	400	154	3	IMAGE

MiddlePhalanxOutlineCorrect	600	291	2	IMAGE
MiddlePhalanxTW	399	154	6	IMAGE
MixedShapes	500	2425	5	IMAGE
MixedShapesSmallTrain	100	2425	5	IMAGE
MoteStrain	20	1252	2	SENSOR
NonInvasiveFetalECGThorax1	1800	1965	42	ECG
NonInvasiveFetalECGThorax2	1800	1965	42	ECG
OliveOil	30	30	4	SPECTRO
OSULeaf	200	242	6	IMAGE
PhalangesOutlinesCorrect	1800	858	2	IMAGE
Phoneme	214	1896	39	SOUND
PigAirwayPressure	104	208	52	HEMODYNAMICS
PigArtPressure	104	208	52	HEMODYNAMICS
PigCVP	104	208	52	HEMODYNAMICS
Plane	105	105	7	SENSOR
PowerCons	180	180	2	DEVICE
ProximalPhalanxOutlineAgeGroup	400	205	3	IMAGE
ProximalPhalanxOutlineCorrect	600	291	2	IMAGE
ProximalPhalanxTW	400	205	6	IMAGE
RefrigerationDevices	375	375	3	DEVICE
Rock	20	50	4	SPECTRO
ScreenType	375	375	3	DEVICE
SemgHandGenderCh2	300	600	2	SPECTRO
SemgHandMovementCh2	450	450	6	SPECTRO
SemgHandSubjectCh2	450	450	5	SPECTRO
ShapeletSim	20	180	2	SIMULATED
ShapesAll	600	600	60	IMAGE
SmallKitchenAppliances	375	375	3	DEVICE
SmoothSubspace	150	150	3	SIMULATED
SonyAIBORobotSurface1	20	601	2	SENSOR
SonyAIBORobotSurface2	27	953	2	SENSOR
StarlightCurves	1000	8236	3	SENSOR
Strawberry	613	370	2	SPECTRO
SwedishLeaf	500	625	15	IMAGE
Symbols	25	995	6	IMAGE
SyntheticControl	300	300	6	SIMULATED
ToeSegmentation1	40	228	2	MOTION
ToeSegmentation2	36	130	2	MOTION
Trace	100	100	4	SENSOR
TwoLeadECG	23	1139	2	ECG

TwoPatterns	1000	4000	4	SIMULATED
UMD	36	144	3	SIMULATED
UWaveGestureLibraryAll	896	3582	8	MOTION
UWaveGestureLibraryX	896	3582	8	MOTION
UWaveGestureLibraryY	896	3582	8	MOTION
UWaveGestureLibraryZ	896	3582	8	MOTION
Wafer	1000	6164	2	SENSOR
Wine	57	54	2	SPECTRO
WordSynonyms	267	638	25	IMAGE
Worms	181	77	5	MOTION
WormsTwoClass	181	77	2	MOTION
Yoga	300	3000	2	IMAGE

Tabla A.1: Datasets utilizados en el benchmark

Apéndice B

Detalles de implementación y código

En este anexo se incluyen algunos detalles sobre la implementación concreta de la librería y el benchmark, añadiendo snippets de código.

B.1. Librería

B.1.1. Función principal

La pieza principal de la librería es la función de entrada, *generate_features*, y se encuentra en el módulo principal que debe ser importado. Esta es la única función pública a la que tiene acceso el usuario. A continuación se muestra un ejemplo de su uso sin ningún tipo de configuración:

```
>>> from bigml.proteus import proteus
>>> features = proteus.generate_features(signal,
                         signal_name='Test')
>>>
>>> features
{
    'Test.RAW_Entropy': 4.9281976097986595,
    'Test.RAW_ZeroCrossings': 20,
    'Test.RAW_MeanCrossings': 39,
    'Test.RAW_Quantile5': -0.549903355691758,
    'Test.RAW_Quantile25': 0.5033459903311247,
    'Test.RAW_Quantile75': 1.4966540096688727,
    'Test.RAW_Quantile95': 2.5499033556917503,
    'Test.RAW_Median': 1.0,
    'Test.RAW_Mean': 0.9999999999999999,
    'Test.RAW_StdDeviation': 1.0000000000000002,
```

```
    'Test.RAW_RMS': 1.4142135623730951,
    'Test.FFT_1_LOC': 0,
    'Test.FFT_1_VALUE': 0.7812499999999998,
    'Test.FFT_2_LOC': 13,
    'Test.FFT_2_VALUE': 0.36295089815546944,
    'Test.FFT_3_LOC': 26,
    'Test.FFT_3_VALUE': 0.3323023053090839,
    'Test.PSD_1_LOC': 0,
    'Test.PSD_1_VALUE': 72.52249582307844,
    'Test.PSD_2_LOC': 13,
    'Test.PSD_2_VALUE': 33.92925300844974,
    'Test.PSD_3_LOC': 26,
    'Test.PSD_3_VALUE': 31.18786415340304,
    'Test.ACF_Entropy': 5.298317366548037,
    'Test.ACF_ZeroCrossings': 24,
    'Test.ACF_MeanCrossings': 21,
    'Test.ACF_Quantile5': -0.4046850374777887,
    'Test.ACF_Quantile25': -0.17783580474379412,
    'Test.ACF_Quantile75': 0.09749986759987081,
    'Test.ACF_Quantile95': 0.6160257287345352,
    'Test.ACF_Median': -0.039060868528057055,
    'Test.ACF_Mean': 0.0025,
    'Test.ACF_StdDeviation': 0.2937472799873641,
    'Test.ACF_Variance': 0.08628746449997489,
    'Test.ACF_RMS': 0.29375791819111,
    'Test.CWT_1_TIME': 56.0,
    'Test.CWT_1_FREQ': 0.04924242424242424,
    'Test.CWT_1_VALUE': 24.34388493256909,
    'Test.CWT_2_TIME': 136.0,
    'Test.CWT_2_FREQ': 0.04924242424242424,
    'Test.CWT_2_VALUE': 24.342335541256954,
    'Test.CWT_3_TIME': 76.0,
    'Test.CWT_3_FREQ': 0.04924242424242424,
    'Test.CWT_3_VALUE': 24.341635400184003
}
```

Snippet B.1: Ejemplo de uso de la función principal

Aquí se puede ver cómo, con una simple llamada a una función, con una señal de entrada (una lista numérica que se encuentra en la variable *signal*) y un nombre, se ha conseguido un diccionario que contiene un conjunto valores numéricos.

Se ha escogido un formato de diccionario a la salida, ya que es casi tan importante el nombre de la variable como su valor. De esta forma, se permite interpretar los valores que se han devuelto.

La interfaz de la función principal se ha mantenido mínima, reduciendo la barrera de entrada para nuevos desarrolladores.

Las tareas de las que se ocupa esta función son:

- leer la configuración
- preparar la entrada para su procesado
- dirigir el flujo de ejecución

El primer paso que ejecuta la función es asegurarse de que se ha recibido una configuración adecuada, y además, proveer de unos valores por defecto. El hecho de proveer unos valores por defecto, permite que el usuario solo tenga que poner los valores que desea cambiar, y el resto de valores que están omitidos se recogen de un valor por defecto. Este valor por defecto es un diccionario que está definido como una constante en la librería.

Para unir los valores por defecto con las modificaciones, *Python* dispone de una *collection*, llamada *ChainMap*, que permite encadenar diccionarios, de forma que a la hora de leer un valor, se busquen de forma secuencial en la lista de diccionarios que se ha pasado en el constructor. Excepto en el constructor, la interfaz es la misma que para un diccionario normal, por lo que el resto de funciones no tienen que tener conocimiento sobre este detalle de implementación.

De esta forma, pasando primero la configuración del usuario, y después la por defecto, conseguimos automáticamente el efecto de leer primero de la configuración de usuario, y si no se encuentra se recurre al valor por defecto. Esto se ve en el siguiente listado:

```
if config is None:
    config = {}
# Merge user config with default
config = ChainMap(config, DEFAULT_CONFIG)
```

Snippet B.2: Lectura de la configuración

La siguiente tarea de *generate_features* es preparar la entrada. Como facilidad para el usuario, a pesar de que la librería utiliza transformaciones basadas en *numpy.ndarray*, se acepta como entrada una lista nativa de *Python*. Por ello, el siguiente paso es el de convertir la entrada al tipo correcto.

Una vez se tiene procesada la configuración, y los datos están en el tipo correcto, esta función se encarga de controlar el flujo de datos, delegando en funciones

de más bajo nivel. Estas funciones, cuyo funcionamiento se explica en el siguiente apartado, requieren como entrada la señal, el nombre y la configuración, y devuelven como resultado un subconjunto de las transformaciones, en un diccionario de *Python*, siguiendo el mismo formato que la función principal. Esto se puede ver en el siguiente listado:

```
raw_feats = raw_features(signal, signal_name, config=
    config)
spectrum_feats =
    spectrum_features(signal, signal_name, pad, N,
        config=config)
autocorrelation_feats =
    autocorrelation_features(signal, signal_name,
        config=config)
cwt_feats = cwt_features(signal, signal_name, config=
    config)
ref_feats = ref_features(signal, signal_name, config=
    config)
```

Snippet B.3: Uso de las funciones de *features*

Por último, sólo queda unir todos estos diccionarios en uno solo, para devolverlo como resultado final.

```
res = {
    **raw_feats,
    **spectrum_feats,
    **autocorrelation_feats,
    **cwt_feats,
    **ref_feats}
return res
```

Snippet B.4: Ejemplo de uso de la función principal

Por último, cabe destacar que al ser una función que se encuentra de cara al usuario, es importante añadir documentación.

```
def generate_features(signal, signal_name, config=None
    ):
    """
    Generate features based on digital signal
        processing
    transforms.
```

```
    This returns a dictionary with the structure:
        {
            'signal_name.feature1': feat_value,
            'signal_name.feature2': feat_value2
        }

    :param signal: input signal, list of numbers
    :type signal: list
    :param signal_name: signal name to prepend to the
       name
        of the features generated
    :type signal_name: string
    :param: config: dictionary with configuration
       options
    :type config: dict, optional
    :return: A dict with the generated features
        {feature_name: feature_value}
    :rtype: dict
    """
    ...
```

Snippet B.5: Documentación de la función principal

B.1.2. Agrupaciones de características

Por debajo de la función principal, se encuentran una serie de funciones, que tienen la misma estructura que ella. Estas funciones no representan ningún módulo en particular, y simplemente son una herramienta para evitar repetición de código. Estas funciones agrupan la gestión del flujo de transformadas y características que son conceptualmente similares, y requieren el mismo tipo de tratamiento.

Estas son las que realmente controlan el flujo para un subconjunto de transformadas. En primer lugar extraen la configuración relevante, llaman a las transformaciones y a las funciones de extracción de características correspondiente, y por último, unen los resultados. Estas funciones son una version especializada de la función principal, pero lógicamente tienen la misma función.

Por ejemplo, la función *spectrum_feats* que se encarga de gestionar el flujo para las transformadas espectrales, es decir, la *FFT* y la *PSD* es la siguiente:

```
# Extract relevant configuration parameters
```

```
ignore_fft = get_prop_from_config('ignore', 'fft',
    config=config)
ignore_psd = get_prop_from_config('ignore', 'psd',
    config=config)
k_fft = get_prop_from_config('k', 'fft', config=config
    )
k_psd = get_prop_from_config('k', 'psd', config=config
    )

fft_feats = {}
psd_feats = {}
if not ignore_fft:
    if signal is not None:
        sig_fft = transforms.get_abs_fft(signal, pad=
            pad, N=N)
        fft_peak_locs = features.top_k_peaks(sig_fft,
            k_fft)
    else:
        sig_fft = None
        fft_peak_locs = None
    fft_feats = features.prepare_k_peak_features(
            sig_fft, fft_peak_locs, k_fft,
            transform_name='FFT', signal_name=
                signal_name)

if not ignore_psd:
    if signal is not None:
        sig_psd = transforms.get_psd(signal, pad=pad,
            N=N)
        psd_peak_locs = features.top_k_peaks(sig_psd,
            k_psd)
    else:
        sig_psd = None
        psd_peak_locs = None
    psd_feats = features.prepare_k_peak_features(
            sig_psd, psd_peak_locs, k_psd,
            transform_name='PSD', signal_name=
                signal_name)
return {**fft_feats, **psd_feats}
```

Snippet B.6: Implementación de *sprectrum_feats*

B.1.3. Transformaciones

El siguiente tipo de funciones son las transformaciones, que se encuentran en el módulo de *transforms* (ver Subsección 5.2.2). Estas funciones son las encargadas de transformar la señal de entrada en otro tipo de valor, generalmente otra señal, que exponga cierta información sobre las particularidades de una señal de forma que sea fácil de extraer en un paso siguiente. Las transformaciones elegidas finalmente se comentan en el siguiente capítulo (Capítulo 6) con el resto de resultados.

En cuanto a la implementación, ya que la velocidad de desarrollo era un requisito, se ha decidido reutilizar implementaciones de las transformaciones cuando ha sido posible. Otro de los requisitos es la capacidad de realizar modificaciones a futuro. Por eso se ha decidido no incluir directamente en el código principal de la librería esas llamadas a módulos externos, sino que esta dependencia se ha abstraído mediante una función que envuelve la llamada a la dependencia.

Esto permite poder cambiar la implementación concreta de una transformación sin necesidad de cambiar código en ninguna otra parte, más que dentro de la función encargada de abstraerlo.

A continuación se muestra un ejemplo de cómo está implementada la transformación de la *FFT*:

```
def get_fft_values(signal, pad=True, N=None, full=
   False):
    """
    Get the Fast Fourier Transform of the signal.

    pad: pad the signal with 0 to match the length of
       the
    signal with  the next power of 2
    N: if pad=True, this is ignored, else mark a size
       for
    the FFT (len(signal) by default)
    """
    if pad == True:
        N = utils.next_power_2(signal.size)
    elif N is None:
        N = signal.size
```

```
    if np.any(np.iscomplex(signal)):
        # If the signal is complex, use the regular
           FFT
        vals = np.fft.fft(signal, n=N) / N
    else:
        # Real FFT is faster than regular FFT
        vals = np.fft.rfft(signal, n=N) / N
    if full:
        return vals
    else:
        return vals[:N//2]
```

Snippet B.7: Implementación de la *FFT*

Como se puede ver, a excepción de alguna condición para dar control al usuario, la función únicamente llama a la implementación de la *FFT* ofrecida por *numpy.fft*. Un ejemplo aún más claro es el calculo de la *ACF*, que utiliza la implementación del paquete *statsmodels* [35]:

```
def get_acf(signal, nlags=None):
    if nlags is None:
        nlags = signal.size
    # Using statstools
    res = st.acf(signal, nlags=nlags, fft=True)
    return res
```

Snippet B.8: Ejemplo de uso de la función principal

Un caso en el que se demostraron las ventajas de esta técnica se vio en un punto durante el desarrollo. Se observó que el tiempo que tardaban las transformaciones en ejecutarse era inaceptable, especialmente conforme las señales tenían más muestras. Para solucionarlo, se utilizaron técnicas de perfilado, en particular se utilizó el *flamegraph* (ver Figura B.1). Este tipo de figuras visualizar de forma interactiva la distribución del tiempo de ejecución de una función o programa. En el eje X se representa el tiempo que se ha invertido en la ejecución de una determinada función del programa perfilado, y en el eje Y se apilan las subrutinas, es decir, las funciones que se han llamado desde el cuerpo de otra. En el gráfico, se pudo observar que prácticamente todo el tiempo de ejecución se concentraba en calcular la *PACF*.

De esta forma, se buscó una implementación alternativa. Tras una investigación, se encontró una solución, y es que la misma función que se utilizaba para calcular la *PACF* tiene un parámetro que permite elegir entre distintas implementaciones.

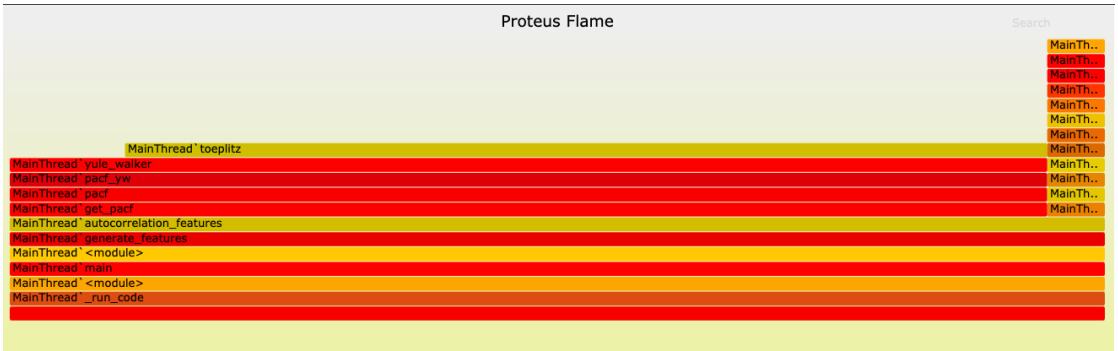

Figura B.1: Flamegraph de ejecución

De esta forma, simplemente añadiendo un parámetro a la función, se consiguió una reducción del tiempo de ejecución cercana a 100 veces.

B.1.4. Extracción de características

Estas funciones se encargan de aplicar algún algoritmo de extracción de características sobre el resultado de alguna transformación (o la señal original). Estos son algoritmos de reducción, que tienen como entrada una lista de valores y devuelven 1 o varios valores, que tienen algún tipo de significado. Estas funciones se encuentran en el módulo de *features* (ver Subsección 5.2.2).

Un ejemplo de estas funciones es la implementación de la función que devuelve los K picos con mayor valor absoluto:

```
def top_k_peaks(values, k, abs_value=False):
    """
    Returns the indices of the top K peaks in the
       signal

    If abs_value=True, try to detect negative peaks
    (valleys) as well
    """
    peaks = detect_peaks(values, abs_value=abs_value)
    top_k_peaks = top_k(values[peaks], k)
    if top_k_peaks.size == 0:
        return top_k_peaks
    else:
        return peaks[top_k_peaks]
```

Snippet B.9: Implementación de *top_k_peaks*

Un ejemplo del uso de esta función se puede ver en el Snippet B.6. Otro ejemplo de extracción de características es la función que calcula estadísticos de una señal o una transformada.

Pero estas funciones no devuelven las características en el formato que espera el usuario, por lo que es necesario coger los valores generados y asignarles unos nombres que permitan interpretarlos después.

Para esto existe una familia de funciones que se cumplen esta función. Por ejemplo, la función encargada de procesar la salida de la mostrada en el bloque anterior (Snippet B.9):

```
def prepare_k_peak_features(transform, peak_locs, k,
    transform_name, signal_name):
    """
    Helper to prepare the K peak features of a signal
    Returns a dict with the form:
        {
            feature_name: feature_value
        }
    Where feature_name has the structure
    `$TRANSFORM_$K_{FREQ,TIME,VALUE}`,
    f.e: { "FFT_2_VALUE": 3.231 }
    """
    res = {}
    for i in range(k):
        if transform is not None and peak_locs is not
           None:
            try:
                loc = peak_locs[i]
                feature_loc_val = loc
                feature_value_val = transform[loc]
            except IndexError as e:
                # Mark peak as missing
                feature_loc_val = None
                feature_value_val = None
        else:
            # Mark peak as missing
            feature_loc_val = None
            feature_value_val = None
```

```
        peak_rank = i + 1
        feature_loc_name =
            f'{signal_name}.{transform_name}_{
               peak_rank}_LOC'
        feature_value_name =
            f'{signal_name}.{transform_name}_{
               peak_rank}_VALUE'
        res[feature_loc_name] = feature_loc_val
        res[feature_value_name] = feature_value_val
    return res
```

Snippet B.10: Implementación de una función de *features*

Esta función devuelve características en el formato que se espera a la salida, como se puede ver en el ejemplo de Snippet 6.1.

B.1.5. Configuración

Para mantener la usabilidad, al mismo tiempo que se da libertad de modificar los parámetros de configuración, se ha utilizado una configuración por defecto.

En la implementación, esta configuración por defecto tiene la misma estructura que la configuración que utiliza el usuario. Cuando comienza la ejecución en la función principal, se une la configuración recibida del usuario con la configuración por defecto.

Para este tipo de casos de uso, *Python* implementa una estructura de datos, llamada *ChainMap*, que es una estructura similar a un diccionario, con la misma interfaz, pero que se construye combinando el diccionario del usuario y el por defecto. Con estos dos diccionarios, cada vez que se haga una consulta sobre el *ChainMap*, se va a buscar la clave en el primer diccionario (el del usuario), y sólo en el caso de que no exista, se realiza la consulta sobre el siguiente en la cadena (el por defecto).

De esta forma, el usuario solo tiene que incluir en su configuración los valores que desea modificar, y el resto se incluyen automáticamente. La configuración por defecto es la siguiente:

```
CONFIG = {
    'k': 3,
    'raw': {
        'ignore': False
    },
    'fft': {
```

```
        'ignore': False,
        'args': (),
        'kwargs': {}
    },
    'psd': {
        'ignore': False,
        'args': (),
        'kwargs': {}
    },
    'acf': {
        'ignore': False,
        'args': (),
        'kwargs': {}
    },
    'cwt': {
        'ignore': False,
        'args': (),
        'kwargs': {}
    },
    'reference': {}
}
```

Snippet B.11: Configuración por defecto de la librería

De esta forma, se puede aprovechar esta técnica para recibir y pasar argumentos de implementaciones específicas, por ejemplo la *FFT* de *numpy.fft* utilizando la técnica de *Python* de ***kwargs*. De esta forma no sería necesario añadir soporte específico para un argumento concreto, y cualquier usuario puede ir a la documentación del proveedor y buscar toda la configuración que está disponible por defecto gracias a esa función.

B.1.6. Testing

Para este proyecto se ha elegido utilizar *pytest*, que es el motor de tests más popular hoy por hoy en el ecosistema de *Python*. *pytest* tiene un diseño modular y extensible a base de plugins que se pueden instalar como un paquete más de *Python* con *pip*.

Este motor requiere una mínima configuración, aunque es un coste que solo hay que pagar una única vez. Una vez configurado, *pytest* es capaz de encontrar automáticamente los archivos que contienen los tests, siempre que se sigan ciertas convenciones. Con seguir estos pasos, ya se pueden ejecutar los tests, aunque para

hacer configuraciones específicas y adaptar el funcionamiento al gusto del desarrollador, se puede añadir en el raíz del proyecto un archivo "pytest.ini". En este caso se ha utilizado para eliminar algunos warnings, reduciendo el ruido en el resultado de los tests, de la siguiente forma.

```
[pytest]
filterwarnings =
    ignore::DeprecationWarning
```

Snippet B.12: *pytest.ini*

Otra de las ventajas de esta librería es el sistema de *fixtures* que permiten definir datos o funciones que se deben preparar al inicio de cada test, o del sistema entero, y que se inyectan automáticamente en la función. De esta forma, se puede definir un dato de prueba, y que se comparte entre distintos tests, entre otras muchas ventajas.

Además, estas fixtures se pueden extraer de los archivos de tests concretos a un archivo separado, y esto permite reutilizar una misma *fixture* en distintos tests sin necesidad de volver a definirlo. *pytest* resuelve esto mediante un archivo especial, llamado *conftest.py* que es importado automáticamente antes de ejecutar otros tests. De esta forma, las *fixtures* ahí definidas están disponibles en cualquier test, sin necesidad de importarlas explícitamente.

En cuanto a la implementación de los tests, *pytest* encuentra automáticamente los tests que se encuentren en archivos que empiezan o terminan por "test". En este proyecto se han agrupado los tests en su propio módulo, y en este módulo se encuentran un archivo separado por cada uno de los módulos que se han definido anteriormente, además de un *conftest.py* que contiene las *fixtures* compartidas.

Por ejemplo, en *conftest.py* se define una fixture, de la siguiente manera:

```
@pytest.fixture(scope='session')
def test_signal():
    signal, _ =
        utils.generate_test_signal(200, 1000, [0, 50,
           100])
    return signal
```

Snippet B.13: Extracto de *conftest.py*

Y de esta forma, el valor de esta señal se encuentra accesible a cualquier test, sólo con declarar un argumento que tiene el mismo nombre que la *fixture*, en este caso, "test_signal".

A continuación se muestra un ejemplo de cómo se están probando algunas

de las funciones complejas como el cálculo de la *FFT*. Este test se encuentra en *test_ transforms.py*, que se encarga de comprobar el correcto funcionamiento de la implementación escogida, utilizando una señal sencilla conocida, como es un seno de una amplitud y frecuencia concretas, de forma que se sabe a priori que valores esperar a la salida.

```
def test_get_abs_fft_sine():
    """
    Test that the absolute value of the FFT of a known
    signal is computed correctly
    """
    A = 2
    N = 128
    Fs = 1024
    Ts = 1 / Fs
    F = 32
    t = np.arange(0, N*Ts, Ts)
    signal = A * np.sin(2 * np.pi * F * t)
    fft = get_abs_fft(signal)
    freqs = np.fft.fftfreq(signal.size, Ts)
    max_idx = np.argmax(fft)
    # The highest value should be at F
    assert freqs[max_idx] == approx(F)
    assert fft[max_idx] == approx(A / 2)
```

Snippet B.14: Ejemplo de test

Cabe destacar el uso de la función de utilidad que ofrece *pytest* llamada *approx*, que permite comparar números en coma flotante con una cierta tolerancia (configurable).

A continuación , se muestra una captura que muestra el feedback que se obtiene de esta herramienta. En la Figura B.2, se muestra un ejemplo de la salida que se obtiene cuando todo funciona correctamente.

Por otro lado, en la Figura B.3 se muestra un ejemplo de la salida cuando alguno de los tests falla. Ahí se puede otro de los puntos fuertes de *pytest*, y son los *diffs* que produce cuando encuentra un error, que ahorran mucho tiempo a la hora de identificar el problema exacto. Esto es especialmente útil cuando se trabaja con estructuras de datos complejas, como los diccionarios que se obtienen a la salida de la librería.

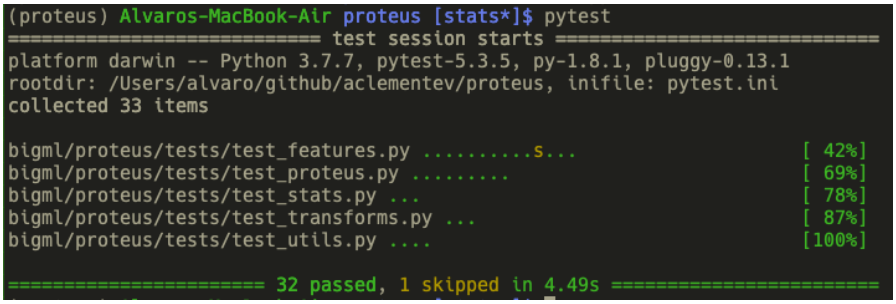

Figura B.2: Ejemplo de ejecución de los tests

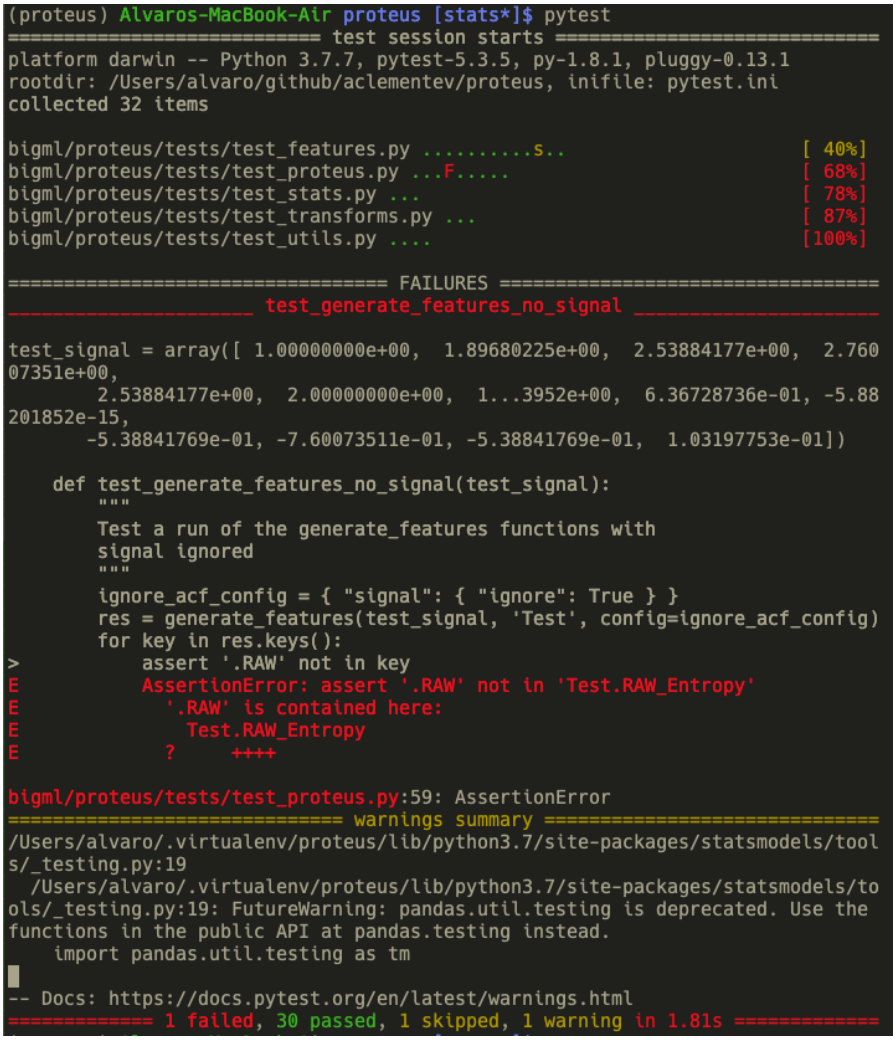

Figura B.3: Ejemplo de salida de los tests en caso de error

B.1.7. Empaquetado

Una parte muy importante de la usabilidad del proyecto es que tiene que ser fácil y simple de integrar en un proyecto. Por una parte esto se consigue gracias a la simple API que se ha diseñado.

Sin embargo, todos los esfuerzos en cuanto a la usabilidad empiezan por el proceso de instalación. Es de extrema importancia que la librería sea fácil de instalar.

Por suerte *Python* ofrece una solución para este problema. Las distribuciones de *Python* vienen con una herramienta a parte incluida, *pip*, que es un gestor de paquetes que permite instalar módulos mediante un único comando e instalarlos de forma que estén accesibles en el *PYTHONPATH*. Cualquier módulo que se encuentre en el *path* de *Python* se puede importar con la sintaxis ya conocida de "import *modulo*".

Pero para que este sistema funcione como debería, es necesario incluir cierta configuración en el proyecto, permitiendo que se puedan empaquetar todos los componentes del mismo en un formato que luego *pip* pueda instalar. Esto se consigue mediante el uso del módulo estándar *setuptools* de *Python*. Incluyendo un archivo *setup.py* en el proyecto, *setuptools* es capaz crear una *wheel*, que es el formato que se distribuye e instala con *pip*.

setup.py contiene una descripción de la estructura del proyecto, información sobre el autor, una descripción, categorías y palabras clave que permiten su búsqueda en repositorios, y enlaces a la documentación.

Sin embargo, la información esencial que incluye en este archivo, son las *dependencias*. En este archivo se incluyen las librerías que este proyecto espera que estén instaladas para poder funcionar, y, adicionalmente, las versiones compatibles.

Con esta información, *pip* puede asegurarse de instalar las librerías de las que depende el proyecto que quieres instalar, consiguiendo así que un simple *pip install* sea suficiente para instalar un proyecto con dependencias complejas.

A continuación se muestra un ejemplo del contenido del *setup.py* de este proyecto:

```
# -*- coding: utf-8 -*-
#!/usr/bin/env python

import os
import re
import sys
from setuptools import setup

# Get the path to this project
project_path = os.path.dirname(__file__)
```

```
# Read the version from bigml.proteus.__version__
   without importing
# the package (and thus attempting to import packages
   it depends on
# that may not be installed yet)
# init_py_path = os.path.join(project_path, 'bigml', '
   proteus', '__init__.py')
# print(open(init_py_path).read())

# version = re.search("__version__ = '([^']+)'",
                      # open(init_py_path).read()).group
                         (1)

# Concatenate files into the long description
file_contents = []
for file_name in ["README.md"]:
    path = os.path.join(os.path.dirname(__file__),
       file_name)
    file_contents.append(open(path).read())
long_description = '\n\n'.join(file_contents)

PYTHON_VERSION = sys.version_info[0:3]
INSTALL_REQUIRES = ['numpy>=1.17,<1.18',
                    'matplotlib>=3.1,<3.2',
                    'scipy>=1.3,<1.4',
                    'scaleogram>=0.9,<0.10',
                    'PyWavelets>=1.0,<1.1',
                    'scikit-image>=0.16,<0.17',
                    'statsmodels>=0.10,<0.11',
                    'dtw-python>=1.0,<1.1',
                    'pytest>=5.3,<5.4']

setup(
    name="bigml-proteus",
    description="Framework for feature extraction from
        signals",
    long_description=long_description,
    long_description_content_type="text/markdown",
    version='0.0.4',
```

```
    author="The BigML Team",
    author_email="bigml@bigml.com",
    url="https://bigml.com",
    download_url="https://github.com/bigmlcom/proteus"
       ,
    license="http://www.apache.org/licenses/LICENSE
       -2.0",
    install_requires = INSTALL_REQUIRES,
    packages = ['bigml.proteus', 'bigml.proteus.tests'
       ],
    classifiers=[
        'Development Status :: 1 - Planning',
        'Intended Audience :: Developers',
        'License :: OSI Approved :: Apache Software
           License',
        'Natural Language :: English',
        'Operating System :: OS Independent',
        'Programming Language :: Python',
        'Programming Language :: Python :: 3.6',
        'Programming Language :: Python :: 3.7',
        'Topic :: Software Development :: Libraries ::
           Python Modules',
    ],
    python_requires='>=3.6'
)
```

Snippet B.15: *setup.py* de la librería

Una vez empaquetado, el archivo *wheel* se puede subir a un repositorio de paquetes público. El que utiliza *pip* por defecto es *PyPI*, el *Python Package Index*, que es público y gratuito.

Con esta configuración, la librería se puede instalar mediante *pip install bigml-proteus*, y se puede importar con *import bigml.proteus*, o directamente los módulos *from bigml.proteus import proteus*.

B.2. Benchmark

Para acelerar el benchmark, se ha paralelizado el proceso de *ETL*, modelado y evaluación, ya que hay que repetirlo para más de 100 datasets, aplicando transformaciones que ya de por sí son exigentes en lo que a computación se refiere. De esta

forma, se pueden estar transformando tantos datasets como cores haya disponibles en la máquina que esté ejecutando el código.

Esto se ha implementado utilizando el módulo nativo de *Python concurrent.futures*, en particular, *ProcessPoolExecutor*. De esta forma, sin hacer muchos cambios en el código que corre en un solo proceso, se puede paralelizar la ejecución de una función en un bucle. La implementación del bucle queda así:

```
# Prepare the args for executor
executor_args = ((path, base_path, transform_fn, api,
   script_id) for path in paths)
# Read the files in the subdirectory in parallel
with ProcessPoolExecutor() as executor:
    for i, (result, name, took) in enumerate(executor.
       map(process_path_wrapped,
executor_args)):
        if result:
            print(f'Processed {name} ({i+1}/{
               total_paths} {((i+1)/total_paths * 100)
               :.2f}%)
Acc: {result["Acc"]} (took {took}s)', flush=True)
            results.append(result)
        else:
            print(f'Processed {name} ({i+1}/{
               total_paths} {((i+1)/total_paths * 100)
               :.2f}%)
with error (took {took}s)', flush=True)
            continue
```

Snippet B.16: Bucle paralelizado

La función *executor.map* es el equivalente a la función *map* de *Python*, pero que manda ejecutar cada paso por el bucle en un proceso diferente, lo que permite el aprovechamiento de todos los núcleos de la máquina.

Por otro lado se han aprovechado las ventajas sobre la gestión de la computación que ofrece BigML dentro de su plataforma. Para ello, se ha creado un script de *WhizzML*. Para importarlo en la plataforma, se puede realizar utilizando la *API* y aportando información adicional, en forma de un archivo de metadatos. Estos archivo, *metadata.json* tiene los siguientes contenidos:

```
{
    "name": "Proteus Benchmark",
    "description": "A small script to benchmark
```

```
        datasets for Proteus",
    "kind": "script",
    "source_code": "dataset-benchmark.whizzml",
    "inputs": [
        {
            "name": "train-src",
            "type": "source-id",
            "description": "Training source id"
        },
        {
            "name": "test-src",
            "type": "source-id",
            "description": "Test source id"
        },
        {
            "name": "target-name",
            "type": "string",
            "default": "label",
            "description": "Target variable name"
        }
    ],
    "outputs": [
        {
            "name": "ensemble",
            "type": "ensemble-id",
            "description": "Generated ensemble"
        },
        {
            "name": "evaluation",
            "type": "evaluation-id",
            "description": "Generated evaluation"
        }
    ]
}
```

Snippet B.17: *metadata.json* del workflow para el benchmark

Apéndice C

Objetivos de Desarrollo Sostenible

A pesar de ser un trabajo de investigación y desarrollo muy técnico, este proyecto se encuentra muy ligado a tres de los objetivos de desarrollo sostenible (ODS) para 2030 de las Naciones Unidas. De esta manera, se deja de manifiesto que desde cualquier ámbito se puede contribuir a una sociedad más justa y sostenible, de la mano del desarrollo tecnológico. A continuación, se describen las metas de los tres objetivos y su estrecha relación con el trabajo realizado y la misión de la tecnología.

El primer objetivo relacionado es el número 9: "Industria, Innovación e Infraestructuras". Este es el ODS más fácil de identificar. Las dos metas en las que se ve reflejado este proyecto son las siguientes:

- 9.b "Apoyar el desarrollo de tecnologías, la investigación y la innovación nacional en los países en desarrollo, incluso garantizando un entorno normativo propicio a la diversificación industrial y la adición de valor a los productos básicos, entre otras cosas"

- 9.5 "Aumentar la investigación científica y mejorar la capacidad tecnológica de los sectores industriales de todos los países, en particular los países en desarrollo, entre otras cosas fomentando la innovación y aumentando considerablemente, de aquí a 2030, el número de personas que trabajan en investigación y desarrollo por millón de habitantes y los gastos de los sectores público y privado en investigación y Desarrollo"

En este proyecto se ha trabajado en investigar las mejores transformaciones y métodos de extracción de características, basándonos en técnicas de procesamiento de señales. Son técnicas, que se basan en nuevos enfoques y perspectivas y permiten entender mejor los datos con los que se trabaja para análisis y modelado. El siguiente objetivo es el número 8 "Trabajo decente y crecimiento económico". Este objetivo busca lograr niveles más elevados de productividad económica a través de la modernización y la innovación con dos metas concretas:

- 8.2 "Lograr niveles más elevados de productividad económica mediante la diversificación, la modernización tecnológica y la innovación, entre otras cosas centrándose en los sectores con gran valor añadido y un uso intensivo de la mano de obra"

- 8.3 "Promover políticas orientadas al desarrollo que apoyen las actividades productivas, la creación de puestos de trabajo decentes, el emprendimiento, la creatividad y la innovación, y fomentar la formalización y el crecimiento de las microempresas y las pequeñas y medianas empresas, incluso mediante el acceso a servicios financieros"

Como se explica a lo largo del trabajo, esta investigación intenta implementar técnicas innovadoras, por ejemplo, la detección de anomalías para detección de fallos, que permiten mejorar el rendimiento en la producción obteniendo mejores resultados y optimizando los recursos económicos. A su vez, estos avances promueven que tanto las instituciones privadas como públicas, desarrollen políticas y medidas que puedan ponerse al servicio de la mejora y el crecimiento.

Por último, tenemos en cuenta el objetivo número 17 "Alianzas para lograr objetivos" y en el apartado relacionado con la tecnología, encontramos tres metas importantes:

- 17.7 "Promover el desarrollo de tecnologías ecológicamente racionales y su transferencia, divulgación y difusión a los países en desarrollo en condiciones favorables, incluso en condiciones concesionarias y preferenciales, según lo convenido de mutuo acuerdo"

- 17.8 "Poner en pleno funcionamiento, a más tardar en 2017, el banco de tecnología y el mecanismo de apoyo a la creación de capacidad en materia de ciencia, tecnología e innovación para los países menos adelantados y aumentar la utilización de tecnologías instrumentales, en particular la tecnología de la información y las comunicaciones"

Con esta librería, se utilizan los conocimientos extraídos en casos reales, como en una fábrica de automóviles de manera que puedan mejorar y optimizar sus procesos. Esto nos da una idea del potencial que tienen los datos y su buen uso para poder ayudar al crecimiento tecnológico y a la mejora de los recursos, acercando este conocimiento a cualquier lugar del mundo. Además, es importante recalcar que uno de los objetivos ha sido ponérselo fácil incluso a aquellas personas que no poseen el conocimiento profundo de lo que pasa por detrás de la tecnología. Se pone al servicio del público una herramienta más que puede crear nuevas oportunidades acercando un tema complejo como es el procesado de señales a diferentes escenarios y sectores.

Bibliografía

[1] Hyung-Chul Lee y Chul-Woo Jung. «Anesthesia research in the artificial intelligence era». En: *Anesthesia and Pain Medicine* 13 (jul. de 2018), págs. 248-255. DOI: 10.17085/apm.2018.13.3.248.

[2] «IEEE Standard for Floating-Point Arithmetic». En: *IEEE Std 754-2008* (2008), págs. 1-70.

[3] Travis E Oliphant. *A guide to NumPy*. Vol. 1. Trelgol Publishing USA, 2006.

[4] Wes McKinney y col. «Data structures for statistical computing in python». En: *Proceedings of the 9th Python in Science Conference*. Vol. 445. Austin, TX. 2010, págs. 51-56.

[5] Fabian Pedregosa y col. «Scikit-learn: Machine learning in Python». En: *Journal of machine learning research* 12.Oct (2011), págs. 2825-2830.

[6] Stefan Van der Walt y col. «scikit-image: image processing in Python». En: *PeerJ* 2 (2014), e453.

[7] Eric Jones, Travis Oliphant, Pearu Peterson y col. *SciPy: Open source scientific tools for Python*. [Online; accessed]. 2001. URL: http://www.scipy.org/.

[8] John D Hunter. «Matplotlib: A 2D graphics environment». En: *Computing in science & engineering* 9.3 (2007), págs. 90-95.

[9] *Simple Plot*. URL: https://matplotlib.org/3.2.1/gallery/lines_bars_and_markers/simple_plot.html.

[10] Toni Giorgino y col. «Computing and visualizing dynamic time warping alignments in R: the dtw package». En: *Journal of statistical Software* 31.7 (2009), págs. 1-24.

[11] *BigML.com*. URL: https://bigml.com/.

[12] *Applying Dimensionality Reduction with PCA to Cancer Data*. Dic. de 2018. URL: https://blog.bigml.com/2018/12/11/applying-dimensionality-reduction-with-pca-to-cancer-data/.

[13] Warren S McCulloch y Walter Pitts. «A logical calculus of the ideas immanent in nervous activity». En: *The bulletin of mathematical biophysics* 5.4 (1943), págs. 115-133.

[14] D Ruhmelhart, G Hinton y R Wiliams. «Learning representations by back-propagation errors». En: *Nature* 323.533-536 (1986), pág. 10.

[15] *Gartner Survey of More Than 3,000 CIOs Reveals That Enterprises Are Entering the Third Era of IT*. URL: `https://www.gartner.com/en/newsroom/press-releases/2018-10-16-gartner-survey-of-more-than-3000-cios-reveals-that-enterprises-are-entering-the-third-era-of-it`.

[16] *2019 Data Science and Machine Learning Market Study Report*. URL: `https://gumroad.com/l/dTfno`.

[17] Kaiming He y col. «Delving deep into rectifiers: Surpassing human-level performance on imagenet classification». En: *Proceedings of the IEEE international conference on computer vision*. 2015, págs. 1026-1034.

[18] Alec Radford y col. «Language models are unsupervised multitask learners». En: *OpenAI Blog* 1.8 (2019), pág. 9.

[19] *State of Data Science and Machine Learning 2019*. URL: `https://www.kaggle.com/kaggle-survey-2019`.

[20] *Recomendación UIT-R v.662-2*. URL: `https://www.itu.int/dms_pubrec/itu-r/rec/v/R-REC-V.662-2-199304-S!!PDF-S.pdf`.

[21] Sanket Doshi. *Extract features of Music*. Abr. de 2019. URL: `https://towardsdatascience.com/extract-features-of-music-75a3f9bc265d`.

[22] Subhashini Narayan y E Sathiyamoorthy. «A novel recommender system based on FFT with machine learning for predicting and identifying heart diseases». En: *Neural Computing and Applications* 31.1 (2019), págs. 93-102.

[23] Steven Davis y Paul Mermelstein. «Comparison of parametric representations for monosyllabic word recognition in continuously spoken sentences». En: *IEEE transactions on acoustics, speech, and signal processing* 28.4 (1980), págs. 357-366.

[24] Ziqin Zhou y Hojjat Adeli. «Time-frequency signal analysis of earthquake records using Mexican hat wavelets». En: *Computer-Aided Civil and Infrastructure Engineering* 18.5 (2003), págs. 379-389.

[25] *What Is Signal Processing Toolbox? - Video*. URL: `https://www.mathworks.com/videos/signal-processing-toolbox-overview-61202.html`.

[26] Maximilian Christ y col. «Time series feature extraction on basis of scalable hypothesis tests (tsfresh–a python package)». En: *Neurocomputing* 307 (2018), págs. 72-77.

[27] Brian McFee y col. «librosa: Audio and music signal analysis in python». En: *Proceedings of the 14th python in science conference*. Vol. 8. 2015.

[28] Youssef Souissi y col. «Novel Applications of Wavelet Transforms based Side-Channel Analysis». En: (abr. de 2020).

[29] Ahmet Taspinar. *A guide for using the Wavelet Transform in Machine Learning*. Abr. de 2019. URL: `http://ataspinar.com/2018/12/21/a-guide-for-using-the-wavelet-transform-in-machine-learning/`.

[30] MYEONG HUN JEONG, Shaowen Wang y Clair Sullivan. «Analysis of dynamic radiation level changes using surface networks». En: feb. de 2016, págs. 199-. ISBN: 9780985244446.

[31] Carl H Lubba y col. «catch22: CAnonical Time-series CHaracteristics». En: *Data Mining and Knowledge Discovery* 33.6 (2019), págs. 1821-1852.

[32] Hoang Anh Dau y col. «The UCR time series archive». En: *IEEE/CAA Journal of Automatica Sinica* 6.6 (2019), págs. 1293-1305.

[33] Christiane Fellbaum. *WordNet: An Electronic Lexical Database*. Bradford Books, 1998.

[34] J. Deng y col. «ImageNet: A Large-Scale Hierarchical Image Database». En: *CVPR09*. 2009.

[35] Skipper Seabold y Josef Perktold. «Statsmodels: Econometric and statistical modeling with python». En: *Proceedings of the 9th Python in Science Conference*. Vol. 57. Scipy. 2010, pág. 61.