COMILLAS
UNIVERSIDAD PONTIFICIA
ICAI

GRADO EN INGENIERÍA INDUSTRIAL

TRABAJO FIN DE GRADO

DESARROLLO DE UN FRAMEWORK DE DETECCIÓN DE DATA DRIFTING

Autor: Pablo Carbonero Álvarez

Director: Alberto Gascón González

Co-Director: Miguel Ángel Velasco Garasa

Madrid

Julio de 2022

PABLO CARBONERO ALVAREZ, declara bajo su responsabilidad, que el Proyecto con título **DESARROLLO DE UN FRAMEWORK DE DETECCIÓN DE DATA DRIFTING** presentado en la ETS de Ingeniería (ICAI) de la Universidad Pontificia Comillas en el curso académico 2022/23 es de su 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.: ..... Fecha: 13 / 07 / 2022

Autoriza la entrega:

Los Directores del Proyecto

Miguel Ángel Velasco Garasa

Fdo.: ..... Fecha: 18 / 07 / 2022

Alberto Gascón González

Fdo.: ..... Fecha: 18 / 07 / 2022

COMILLAS
UNIVERSIDAD PONTIFICIA
ICAI

GRADO EN INGENIERÍA INDUSTRIAL

TRABAJO FIN DE GRADO

DESARROLLO DE UN FRAMEWORK DE DETECCIÓN DE DATA DRIFTING

Autor: Pablo Carbonero Álvarez

Director: Alberto Gascón González

Co-Director: Miguel Ángel Velasco Garasa

Madrid

Julio de 2022

Agradecimientos

Después de un curso entero de trabajo, hoy es el día: escribo este apartado de agradecimientos para finalizar mi trabajo de fin de grado. Ha sido un período de aprendizaje intenso, pero que me ha permitido indagar y entender un poco más el mundo de la inteligencia artificial lo cual era mi objetivo desde el inicio.

Primero de todo, me gustaría agradecer en particular a mi antiguo profesor de estadística II, ahora director de este trabajo de fin de grado, por darme la oportunidad de dar un paso más en mi objetivo de indagar y aprender todo lo que el mundo de los datos ofrece.

Los dos directores detrás de este trabajo de fin de grado han dedicado muchos días a asesorarme y ayudarme en todo lo que he necesitado y por ello estoy muy agradecido.

También me gustaría agradecer a mis padres y mis hermanas por sus sabios consejos y su comprensión. Finalmente agradecer a mis amigos por su apoyo todo este tiempo. También a mis amigos del Erasmus, los cuales han visto nacer desde cero este proyecto y han sido partícipes de todo el proceso. Por último aquellas personas especiales que siempre me han servido de apoyo y son parte fundamental de este proyecto. Gracias.

Pablo Carbonero Álvarez

Universidad Pontificia de Comillas ICAI

DESARROLLO DE UN FRAMEWORK DE DETECCIÓN DE DATA DRIFTING

Autor: Pablo Carbonero Álvarez
Directores: Miguel Ángel Velasco Garasa, Alberto Gascón González
Entidad colaboradora: ICAI – Universidad Pontificia Comillas

RESUMEN DEL PROYECTO

El *data drift* supone una de las principales causas de pérdida de rendimiento silencioso en modelos de machine learning y ocurre cuando los datos de entrada cambian de maneras no vistas en la fase de entrenamiento. En este trabajo se elabora un *framework* para monitorizar la gravedad de este fenómeno, sirviendo al usuario como herramienta para la toma de decisión del reentrenamiento del modelo.

Palabras clave: Inteligencia artificial, machine learning, data drift.

1. Introducción El *machine learning* se trata de una rama dentro del campo de la inteligencia artificial y la ingeniería de software que se centra en el uso de datos y algoritmos para imitar la forma en la que actúan los seres humanos.[1]
Se crean modelos que pueden ser capaces de predecir o clasificar datos en función de lo aprendido. En clasificación, el error suele medirse con el porcentaje de acierto y en regresión con el RMSE (Root Mean Squared Error)

Hoy en día, estos algoritmos conviven con un problema latente, el *data drift*, un fenómeno que puede provocar la perdida de rendimiento silencioso en modelos de *machine learning*. Se trata de un cambio en la distribución de los datos de entrada a un modelo. Para entender por qué esto puede desembocar en una perdida de rendimiento, primero se debe entender cómo se programa o crea este tipo de algoritmos.

Normalmente se tiene como objetivo predecir o clasificar algún tipo de variable y para ello se requiere de la creación de un modelo. Para esto, primeramente se le mostrarán los datos de entrada junto con la salida que cabría esperar. De esta manera y a través de diferentes técnicas dependiendo de la necesidad, el modelo aprende la relación entre los datos de entrada y la salida.

Una vez se aprende esta relación, el modelo permanece estático, esto significa que para unos mismos datos de entrada, la salida será siempre la misma. Sin

embargo, existen ocasiones donde la manera en la que los datos de entrada se comportan, cambia de maneras no observadas en su fase de entrenamiento. Esto hace que el modelo quede anticuado y cometa una cantidad de errores mayores que en el pasado, perdiendo así rendimiento.

2. **Definición del proyecto** Este proyecto tiene como fin elaborar un *framework* que sea capaz de detectar eficazmente el *data drift* y alertar de cuando esto ocurra al usuario del modelo, pudiendo este lanzar un reentrenamiento del mismo.

   A través de este sistema se pretende evitar que el modelo pierda rendimiento en el tiempo. Si no se dispone de un *framework* que monitorice y evalúe los datos de entrada, pero se quiere tener un modelo siempre actualizado, se puede reentrenar este periódicamente. El problema es que esto puede desembocar en reentrenamientos innecesarios, en casos en los que los datos no estén sufriendo de *data drift* o por otro lado, reentrenamientos demasiado tardíos cuando el *data drift* esta muy presente.

3. **Descripción del *framework*** Un modelo de *machine learning* puede contar con cientos de parámetros o variables de entrada. El *framework* desarrollado es capaz de trabajar con variables cuantitativas, categóricas y binarias. Además, es capaz de realizar un análisis del *data drift* en el plano univariante y multivariante, esto es, se analiza el cambio en las distribuciones de los datos por separado, pero también se analiza en su conjunto. Este sistema funciona comparando los datos de entrada al modelo, a través de los diferentes *data-chunks* con los datos con los que el modelo se entrenó originalmente.

   Cabe destacar que se utilizan diferentes tipos de test estadísticos dependiendo del caso de uso que se dé en cada momento [2]. De esta forma, el *framework* logra una detección eficaz para cada caso en concentro. Esto se puede observar en la figura 1.

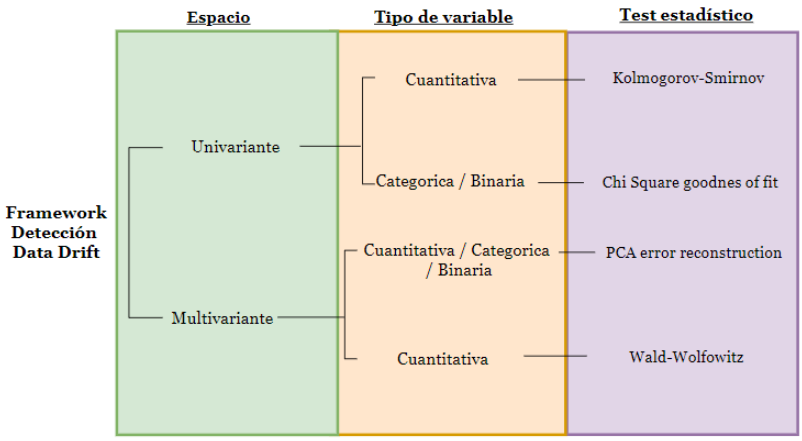

Figura 1: Test estadísticos utilizados por el *framework* en cada caso de uso

4. Resultados Finalmente el *framework* ofrece una evaluación tanto univariante como multivariante, cuantificando el *data drift* que sufre cada variable por separado y en su conjunto. Esto se consigue a través de dos interfaces, la primera, un gráfico automáticamente generado de los valores de los diferentes estadísticos para los distintos paquetes de datos de entrada, un ejemplo de gráfico generado sería: 2. Por otro lado se genera automáticamente un informe de alteras que indica al usuario, cuantas veces los test estadísticos han dictaminado que el paquete de datos recibido en ese momento sufría de *data drift* .

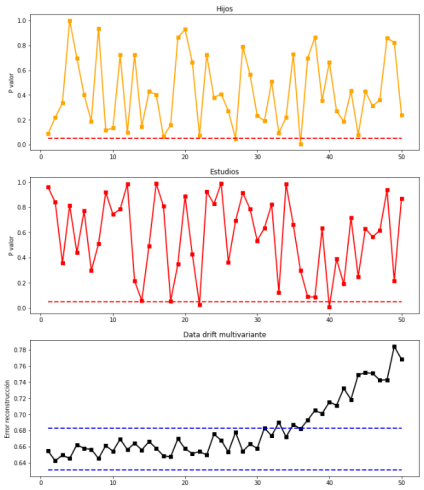

Figura 2: Valor de los test estadísticos a través de los diferentes *datachunks*

REFERENCIAS

[1] Batta Mahesh. Machine learning algorithms-a review. International Journal of Science and Research (IJSR).[Internet], 9:381–386, 2020.

[2] Vance W Berger and YanYan Zhou. Kolmogorov–smirnov test: Overview. Wiley statsref: Statistics reference online, 2014.

DEVELOP OF A FRAMEWORK FOR DATA DRIFT DETECTION
Author: Pablo Carbonero Álvarez
Directors: Miguel Ángel Velasco Garasa, Alberto Gascón González
Collaborating entity: ICAI – Universidad Pontificia Comillas

PROJECT SUMMARY

Data drift is one of the main causes of silent performance loss in machine learning models and occurs when input data changes in ways not seen in the training phases. In this work, a framework is developed to monitor the severity of this phenomenon, serving the user as a tool for decision-making regarding the retraining of the model.

Key words: Artificial Inteliggence, machine learning, data drift.

1. Introduction Machine learning is a branch within the field of artificial intelligence and software engineering that focuses on the use of data and algorithms to mimic the way humans act [1]. Models are created be able to predict or classify data based on what has been learned. In classification, the error is usually measured with the success percentage and in regression with the RMSE (Root Mean Squared Error)

Today, these algorithms coexist with a latent problem, data drift, a phenomenon that can cause silent performance loss in machine learning models. This is a change in the distribution of input data to a model. To understand why this can lead to a loss of performance, the reader must first understand how this type of algorithm are programmed or created.

Usually, the objective is to predict or classify some type of variable and for this, the creation of a model is required. For this, the reader will first be shown the input data along with the output that it would expect. In this way and through different techniques depending on the need, the model learns the relationship between the input and output data.

Once this relationship is learned, the model remains static, this means that for the same input data, the output will always be the same. However, there are times when the way the input data behaves changes in ways not observed in its training phase. This causes the model to become outdated and make a greater number of errors than in the past, thus losing performance.

2. **Definition of the project** The purpose of this project is to develop a framework that is capable of doing an effective detect on data drift and alert the user of the model when this occurs, being able, the user with this info, to launch a retrain of the model.

   This system is intended to prevent the model from losing performance over time. If the user don't have a framework that monitors and evaluates the input data, but he want to have a model that is always up to date, it can retrain it periodically. The problem is that this can lead to unnecessary retraining, in cases where the data is not suffering from data drift or, on the other hand, too late retraining when data drift is very present.

3. **Description of the framework** A machine learning model can have hundreds of parameters or input variables. The developed framework is capable of working with quantitative, categorical and binary variables. In addition, it is capable of performing a data drift analysis at the univariate and multivariate level, that is, the change in the data distributions is analyzed separately, but it is also analyzed as a whole. This system works by comparing the input data to the model, through the different datachunks, with the data with which the model was originally trained.

   It should be noted that different types of statistical tests are used depending on the use case that occurs at each moment [2]. In this way, the framework achieves an efficient detection for each case in focus. This can be seen in figure 3.

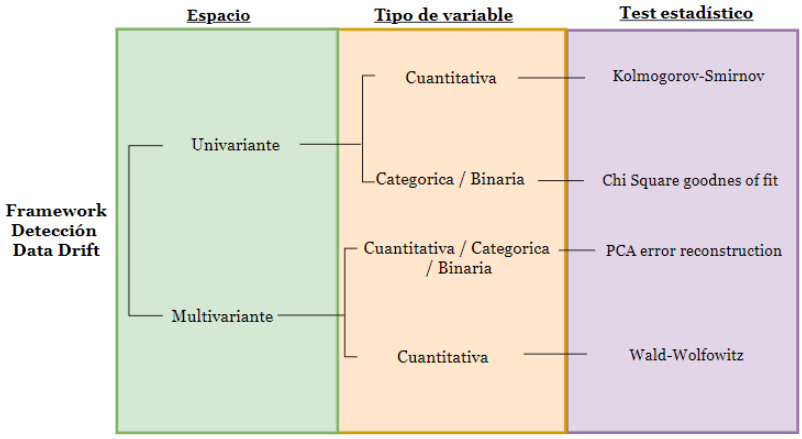

Figura 3: Statistical tests used by the framework in each use case

4. Results Finally, the framework offers both a univariate and multivariate evaluation, quantifying the data drift suffered by each variable separately and as a whole. This is achieved through two interfaces, the first, an automatically generated graph of the values of the different statistics for the different input data packets, an example of a generated graph would be: 4. On the other hand, a report of alterations is automatically generated that indicates to the user how many times the statistical tests have ruled that the data packet received at that moment suffered from data drift.

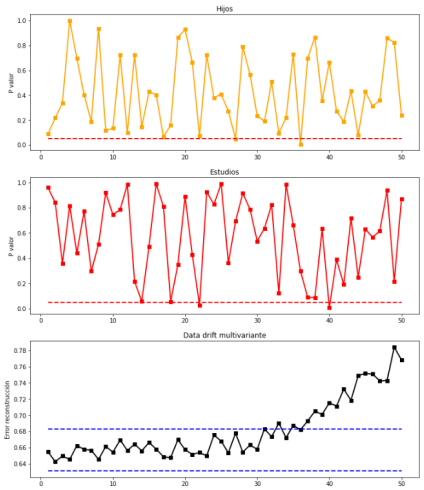

Figura 4: Value of the statistical tests through the different datachunks

REFERENCES

[1] Batta Mahesh. Machine learning algorithms-a review. International Journal of Science and Research (IJSR).[Internet], 9:381–386, 2020.

[2] Vance W Berger and YanYan Zhou. Kolmogorov–smirnov test: Overview. Wiley statsref: Statistics reference online, 2014.

Índice

Índice de figuras

1. INTRODUCCIÓN, MOTIVACIÓN Y OBJETIVOS

1.1. Introducción al machine learning

Reconocimiento del habla, coches que se conducen solos, asistentes que traducen instantáneamente de un idioma a otro o sugerencias de compra personalizadas. Complejas tareas que antes se realizaban con gran dificultad, hoy se pueden implantar de manera más sencilla a través de la utilización de *Machine Learning*, una disciplina de la inteligencia artificial, que permite a los ordenadores aprender por sí mismos y realizar tareas de forma autónoma sin necesidad de ser programados.

Un modelo de *Machine Learning* se diseña con el fin de realizar una tarea, el abanico cubre desde detectar la presencia de un autobús en una imagen, hasta predecir si un cliente está apunto de dejar de usar los servicios de una empresa. La tarea que realizan cada uno de los modelos es muy diferente, sin embargo la manera que tienen de crearse no lo es tanto.

En este panorama existen 3 ramas, el aprendizaje supervisado, no supervisado y el aprendizaje por refuerzo. [1]. En el aprendizaje supervisado el proceso de generación de conocimiento se realiza con un conjunto de datos de entrada de las cuales se conocen la salida. Por otro lado, en el aprendizaje no supervisado se incluyen conjuntos de datos sin etiquetar en los que no se conoce previamente la estructura que estos poseen. En este tipo de aprendizaje se busca obtener información clave o importante sin conocer previamente la referencia de las variables de salida, explorando la estructura de los datos que no están etiquetados. Por último, el aprendizaje reforzado tiene como finalidad construir modelos que aumenten el rendimiento tomando como base el resultado o la recompensa que se genera por cada interacción realizada. Esta recompensa es el producto de una acción correcta o conjunto de datos devueltos que entran en una medida especifica. La imagen 5 muestra gráficamente las distintas ramas anteriormente descritas.
La gran ventaja de la inteligencia artificial es la de poder crear un modelo que aprenda por sí solo. Este se crea a través de una fase entrenamiento.

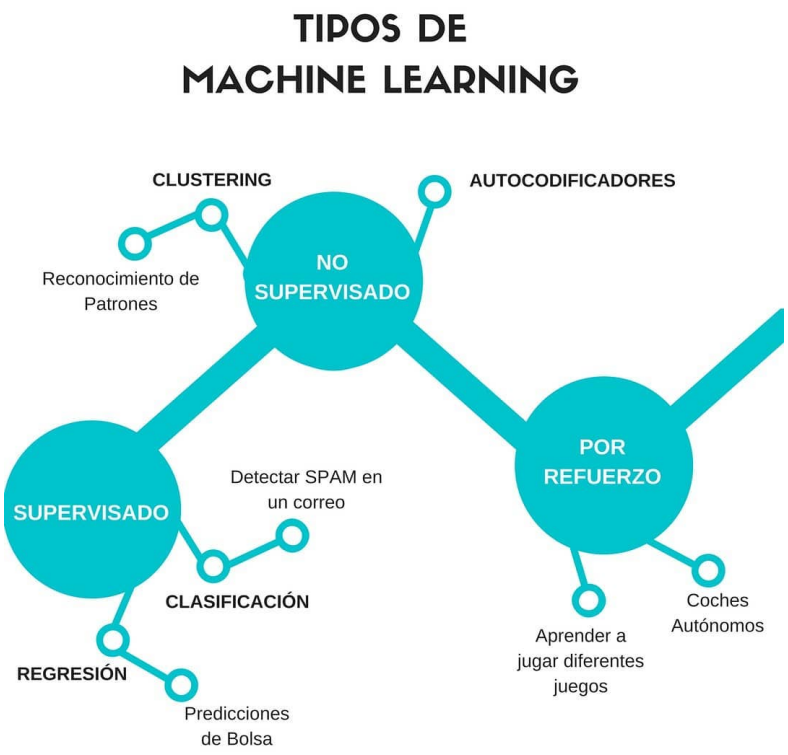

Figura 5: Ramas del *Machine Learning*

El primer paso para la fase de entrenamiento de un modelo es disponer de una serie de datos, las entradas al modelo, de las cuales, por tratarse de aprendizaje supervisado, se conocen sus salidas. A un conjunto de datos se le denomina *dataframe*. Este conjunto debe haber sido procesado, o limpiado, antes de de entrenar al modelo. La limpieza de datos se utiliza para corregir o eliminar registros inexactos, esto significa, identificar y sustituir los datos incompletos, inexactos, corruptos o irrelevantes.

Una gran parte de estos datos, normalmente el 70 %-80 %, serán usados para entrenar el modelo y la otra se utilizará para evaluar su rendimiento una vez se considere creado. Esto se hace proporcionándole unos datos de entrada, nunca antes vistos por el algoritmo y comparando la salida que el modelo estima con la salida real, la cual se conoce en esta fase.

De esta forma se puede saber si el modelo trabaja de la manera que cabía esperar. Por ejemplo, si se extiende uno de los posibles casos anteriormente explicado, el detector de autobuses en una foto, este modelo se entrenaría con un conjunto de abundantes imágenes, de las cuales se dispondría de la información sobre si existe

en la imagen un autobús o no y en la fase de test o evaluación de rendimiento, se comparará la respuesta del modelo con la realidad.

Aunque estos modelos aprenden por sí mismos, las técnicas utilizadas para diseñarlos dependen del tipo de comportamiento que se desea para el modelo, cada uno con hiperparámetros diferentes.

Se debe tener en cuenta que si un modelo se ajusta demasiado para tener el mínimo error en su fase de entrenamiento, es muy probable que ocurra un fenómeno denominado sobreajuste u *overfitting* del modelo, el cual desemboca en un comportamiento pobre del modelo en su fase de test. [2]. Un ejemplo gráfico de este fenómeno en un modelo de clasificación se puede ver en la siguiente imagen 6, también es posible realizar sobreajustes en modelos de regresión, en la figura 7 se puede apreciar la diferencia entre un buen ajuste y un sobreajuste.

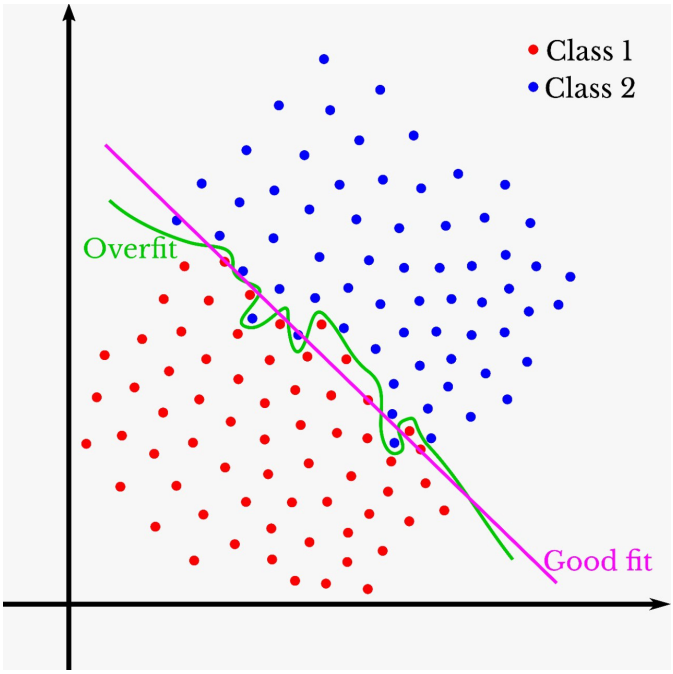

Figura 6: Ejemplo de *overfitting* en modelo de clasificación

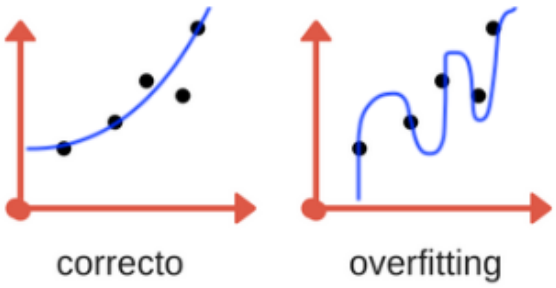

Figura 7: Ejemplo de *overfitting* en modelo de regresión

A la hora de entrenar un modelo se debe evitar que este aprenda de cerca el patrón de los datos proporcionados por el *dataframe* de entrenamiento e intentar que este entienda la distribución que siguen los datos en general. Dicho de otra manera, se trata de evitar que el modelo memorice los datos de entrenamiento para que sea capaz de generalizar con nuevos datos. Cuando esto se consigue, se deberá obtener un nivel similar de error en la fase de aprendizaje y test. Otro punto muy importante a la hora de crear un modelo, es el de elegir datos que reflejen la realidad en la este va a trabajar.

Una vez el modelo esté entrenado, ya no aprenderá más, y permanecerá por tanto estático, a no ser que se vuelva a entrenar, o que por propia naturaleza esté siempre reentrenandose. Aquí reside la importancia de proporcionar unos datos al modelo que reflejen lo más posible la realidad a la que este se va a enfrentar.

Una vez el modelo se dé por bueno y se ponga en producción, siempre proporcionará la misma respuesta de salida ante el mismo estímulo de entrada. Esto es, si en una imagen hay un autobús, y el usuario muestra la misma imagen al modelo cien veces, las cien recibirá la misma respuesta. Sin embargo, el modelo puede dejar de ser útil con el tiempo y perder rendimiento, por ejemplo si nuevos diseños de autobuses comienzan a cambiar de color, tamaño o forma. Este tipo de situaciones se comentarán en el próximo capítulo.

Existen diferentes formas de supervisar el correcto funcionamiento de un modelo de *Machine learning*, pero como se ha comentado anteriormente, de la misma manera que hay modelos que realizan tareas muy diferentes, la forma en la que su rendimiento es evaluado también lo es.

En algunos modelos, esta tarea resulta trivial, supóngase el predictor de tiem-

po del que dispone 'Google Maps'. Este estima cuanto va a tardar un individuo en llegar de un sitio a otro. En este caso el modelo puede auto-evaluarse y saber si sus medidas son acertadas o no simplemente comparando el tiempo que el modelo estimó con el tiempo que realmente le llevó a la persona llegar a su destino.

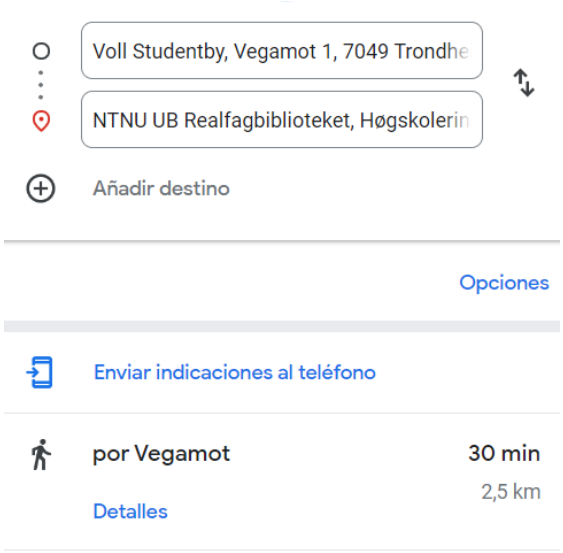

Figura 8: Predictor de tiempo Google Maps

Sin embargo, no todos disponen de una forma tan sencilla de obtener un *feedback* del rendimiento. En el caso de un modelo que predice si una persona va a pagar o no un préstamo, este tardará considerablemente mas tiempo en averiguar si su predicción era correcta o no, lo cual es un problema ya que quizás para cuando este fallo sea detectado, la compañía ya ha sufrido algunas perdidas.

En el modelo ejemplo primeramente mencionado, el detector de autobuses en una foto, se podría saber si este trabaja correctamente mostrando una serie de imágenes a personas y comparando su respuesta con la del modelo. Esta tarea puede sonar tediosa, pero realmente no se necesita mostrar todas y cada una de las imágenes con las que trabaja el modelo. Además, se puede hacer con alguna técnica de control de *bots* en paginas web, como 'reCaptcha'[3]. La imagen 9 muestra un caso de uso de reCaptcha detectando autobuses en una imagen. Comparando la respuesta del modelo, con la realidad esperada, se evalúa su rendimiento.

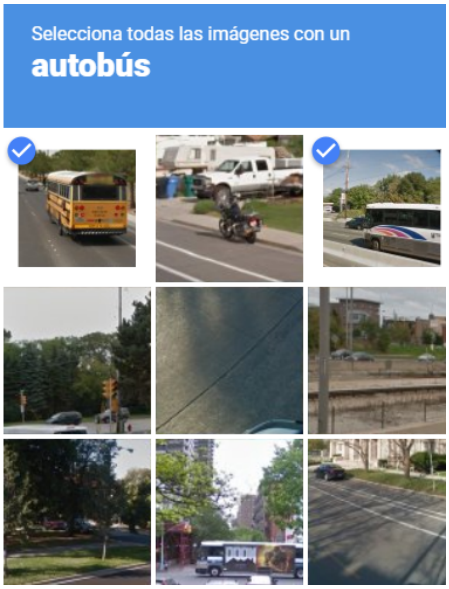

Figura 9: ReCaptcha

Son varios los factores que pueden provocar que un modelo no tenga el rendimiento esperado. Si los datos que se le proporcionó en su etapa de entrenamiento no reflejan la realidad de los datos a los que este se va a enfrentar, entonces no se obtendrá un funcionamiento adecuado. También se puede dar una pérdida de eficacia si el modelo tiene una realimentación que no está correctamente ajustada.

Por ejemplo, si en la construcción de un modelo de predicción de gustos musicales se ofrece a parte de sus usuarios elegir entre 2 canciones A y B y por un sesgo en la selección de esta muestra de usuarios, éstos tienen una educación musical muy superior a población general, justo en ese grupo de personas la canción A es mas escuchada que la B, aunque esto no represente la realidad. El modelo recomendará más la canción A a los siguientes usuarios lo cual posibilitará que esta sea aún mas elegida, cuando en realidad no tiene porque ser la mejor. Este tipo de modelos sufren una perdida de rendimiento por no tener un ajuste efectivo en su realimentación.

1.2. Motivación

Póngase el ejemplo de una compañía dueña de una cadena de supermercados. Esta subcontrata a otra empresa para que le diseñe un modelo de *Machine Learning* que prediga cuanta cantidad de producto debe reponer. La subcontrata toma 6 meses para desarrollar el modelo y, después de este tiempo se pone en funcionamiento.

En un inicio, la cadena de supermercados está muy contenta con el rendimiento del modelo. Sin embargo, un año más tarde esta comienza a tener menos beneficios. El modelo no funciona correctamente y se sobrestima la cantidad de algunos productos a la vez que se subestima el suministro de otros, lo que provoca estanterías vacías en algunos casos. Esto deja a esta empresa con tres opciones: pagar una cantidad de dinero a la empresa que diseñó el modelo para actualizarlo, pagar una cantidad de dinero aun mayor a otra compañía para diseñar otro modelo, o bien contratar un equipo dentro de la empresa para mantener el modelo.

El diseño de un modelo de *Machine learning* no es el fin del proceso. Después de implementarlo se debe evaluar constantemente su rendimiento, así como buscar los fallos que este puede estar teniendo y solucionarlos. Uno de los problemas que pudo llevar a este modelo a perder rendimiento es el *Data Drift* y, debido a que no se llevó a cabo ningún tipo de monitorización del mismo, el rendimiento del modelo cayó.

Este fenómeno, *Data drift* puede darse en los modelos de aprendizaje supervisado y ocurre cuando los datos con los que el modelo trabaja cambian en el tiempo de maneras no vistas en la fase de entrenamiento, lo que produce que las predicciones generadas por el modelo se vuelvan cada vez menos acertadas, perdiendo así rendimiento. Discusiones y artículos sobre este fenómeno han empezado a ser mas comunes en los últimos años con la implantación en muchas áreas de trabajo de modelos de *Machine Learning* en la industria, aunque hay estudios que ya trataban este tema en 1986 [4]

El *data drift* es una de las principales causas que provocan una pérdida de rendimiento silenciosa en un modelo, esto es, cuando aparentemente el modelo no está dañado, este puede no ser tan eficiente debido a la aparición de *data drift*.

Imagínese que se entrena un modelo para clasificar, mediante un árbol de clasificación tres clases de datos.

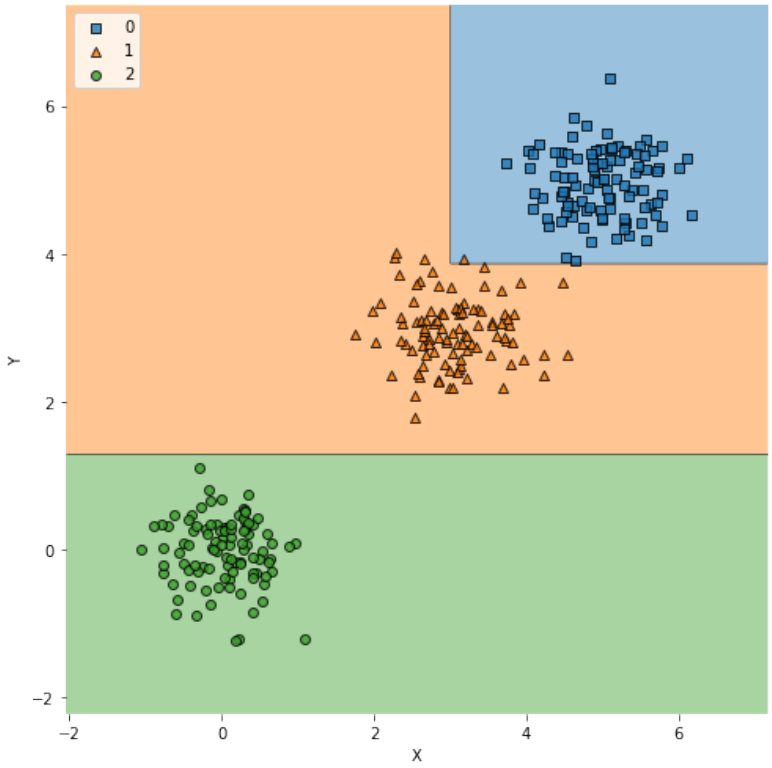

Figura 10: Árbol de clasificación con datos sin drift

Después de un tiempo desde que este fue puesto en funcionamiento, el modelo empieza a cometer una serie de errores mayores que en su etapa anterior de funcionamiento. En este caso, esto es debido a que este modelo está sufriendo *data drift*. La distribución en los datos ha cambiado, pero el modelo sigue estático, por lo que sufrirá una pérdida de rendimiento si no se hace nada al respecto, como reentrenar el modelo.

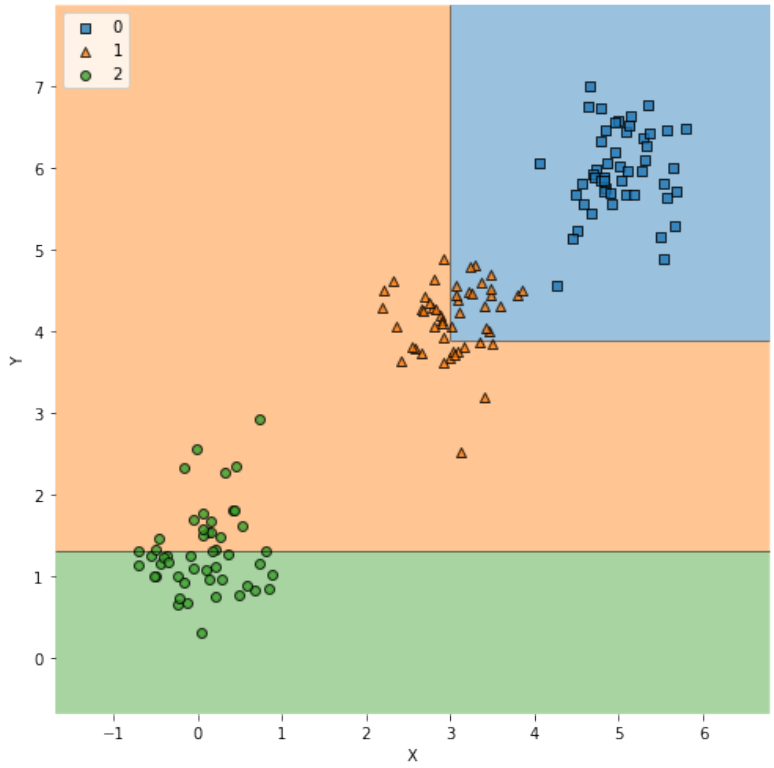

Figura 11: Árbol de clasificación con datos con drift

Como se puede observar en la figura 11, el modelo claramente necesitaría de un reentrenamiento para poder clasificar correctamente estos datos.

Este es el motivo que mueve la creación de este *framework*, analizar y evaluar los datos de entrada tratando de minimizar la perdida de rendimiento que puede llegar a tener un modelo. Además, analizando las tecnologías usadas actualmente para la detección de *data drift*, no encontramos ninguna que cumpla con todos los requisitos a la vez. Esto se explica con detalle en el capitulo 3, donde se revisan las tecnologías utilizadas.

1.3. Objetivo

El objetivo de este proyecto de fin de grado es desarrollar un *framework* que analice los datos de entrada a un modelo e indique al usuario del mismo si es recomendable tomar alguna medida al respecto, como por ejemplo, la de reentrenar. Este sistema desarrollado facilitará la gestión del *data drift* y por tanto se utilizará como herramienta de apoyo en la toma de decisiones por parte del usuario.

Para el correcto desempeño de esta función deben cumplirse varios requisitos.

- Se deben de poder analizar los datos de entrada tanto de manera univariante como multivariante.
- Se debe de poder analizar el *data drift* en variables categóricas, cuantitativas y binarias.
- El tamaño del *dataframe* analizado debe poder ser variado a gusto del usuario.
- Debe contar con una lógica que aplique unos test estadísticos u otros dependiendo del *dataframe* analizado
- Debe ser gratuito de software abierto

Surgen algunas dudas ante esta situación, debido a que esta variación se puede producir de manera momentánea, por lo que más tarde los datos volverán a sus distribuciones originales. Por otro lado, se considera que las distribuciones pueden tener ruido y no ser siempre exactamente iguales, por lo que se debe trabajar con algún grado de confianza. No cualquier mínimo cambio en la distribución de los datos será tachado de *data drift*. Todo esto se tiene en consideración a la hora de crear el *framework*.

2. INTRODUCCIÓN AL DATA DRIFT

Este término se usa conjuntamente, y por tanto suele confundir, con otros como *Concept drift*, *Covariative shift* e incluso *Label shift*. Para explicar las diferencias entre todos estos, primeramente se definen las siguientes notaciones matemáticas. Se denomina X al conjunto de datos de entrada al modelo, por consiguiente Y a los datos de salida. Los distribuciones X, Y pueden provenir del *dataframe* de entrenamiento, usado para entrenar el modelo, o bien del *dataframe* de test, que contiene datos nunca antes vistos por este, pero cuya salida si es conocida por el usuario. $P(X|Y)$ se refiere a la probabilidad condicional de obtener una salida Y conociendo la entrada X.

2.1. Tipos de Drift

Covariate shift

Un modelo sufre *Covariate shift* cuando los datos que fueron utilizados para entrenar al mismo no representan la totalidad de la información con la que dicho modelo podría enfrentarse una vez entrenado $P_{entrenamiento}(X) \neq P_{test}(X)$. Sin embargo ocurre que $P_{entrenamiento}(Y|X) = P_{test}(Y|X)$ en un rango amplio o popular de trabajo.

Por ejemplo, se diseña un modelo para detectar el cáncer de pecho y este es entrenado con los datos de una clínica médica. Debido a que es mucho más probable que mujeres de más de 40 años se presenten a estas pruebas, el modelo será entrenado con un *dataframe* donde abundan los casos en los que la variable edad es mayor que 40 años. El modelo, una vez en funcionamiento, trabajará con buen rendimiento si los datos de entrada son cercanos o mayores a 40 años, pero tendrá un error muy grande cuando los datos difieran de estos. En definitiva, la causa principal del *Covariate shift* reside en la incorrecta selección de una muestra representativa de la población a la hora de entrenar el modelo.

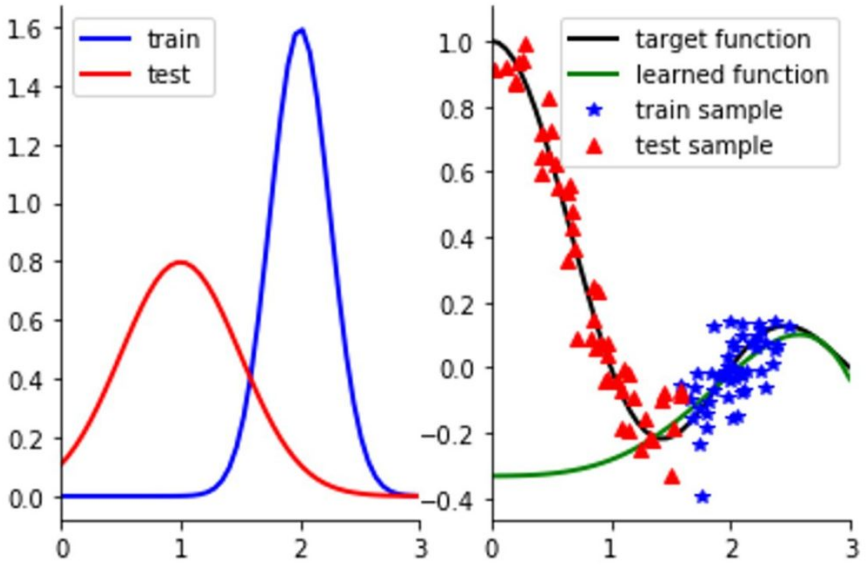

Figura 12: *Covariate shift*

En la imagen 12 se puede observar como en azul se tienen los datos contenidos en el dataframe utilizado para el entrenamiento, y en rojo los datos del test. En este caso se observa como, los datos del test pertenecen a la misma población que los de entrenamiento, sin embargo, como el modelo se entrenó con una muestra no representativa de la población, el modelo tiene un rendimiento paupérrimo.

Label shift

Por otro lado, el fenómeno *Label shift* ocurre cuando la distribución de las predicciones cambian para una salida dada $P_{entrenamiento}(Y) \neq P_{test}(Y)$ pero $P_{entrenamiento}(Y|X) = P_{test}(Y|X)$. Volviendo al ejemplo de antes, como en el *dataframe* con el que se entrenó el modelo, la cantidad de mujeres mayores de 40 años es frecuente, el porcentaje de positivos en cáncer es mayor que en el *dataframe* que engloba toda la población de mujeres de todas las edades. Sin embargo, una mujer mayor de 40 años, tiene la misma probabilidad de tener cáncer de pecho en los dos *dataframes* $P_{entrenamiento}(Y|X) = P_{test}(Y|X)$. Por otro lado, la probabilidad de obtener un positivo en la variable binaria cáncer es mayor en el *dataframe* de test que en el que engloba toda la población. Un *Label shift* suele venir de la mano de un *Covariate shift*, pero esto no es siempre un requisito indispensable para que ocurra.

Concept drift

Por último, un modelo trabaja bajo la presencia de *Concept drift* cuando $P_{entrenamiento}(Y|X) \neq P_{test}(Y|X)$, sin embargo $P_{entrenamiento}(X) = P_{test}(X)$. Es decir, los datos de entrada se comportan de la misma manera, siguiendo la misma tendencia. Sin embargo, las predicciones del modelo empeoran con el paso del tiempo.

Pongamos de ejemplo que se elabora un modelo que estima la temperatura a la que se encuentra el interior del cilindro de un motor. Este modelo cuenta con numerosos datos de entrada y su salida es la variable temperatura del interior del cilindro.
En este caso, el modelo funciona sin problema y durante su primera etapa las predicciones sugeridas por el modelo son correctas.

Sin embargo, y de manera gradual , el cilindro comienza a contar con pequeñas muescas en su interior además de tener los conductos de admisión y escape cada vez más obstruidos. Aunque la variable a predecir sea la misma y los datos de entrada sigan la misma tendencia que los usados para entrenar el modelo, el error en la estimación de la temperatura se hace cada vez mas grande. Esto se debe a que el modelo se entrenó sobre un sistema que ha cambiado. Es en este tipo de situaciones, cuando ocurre un *concept drift*.

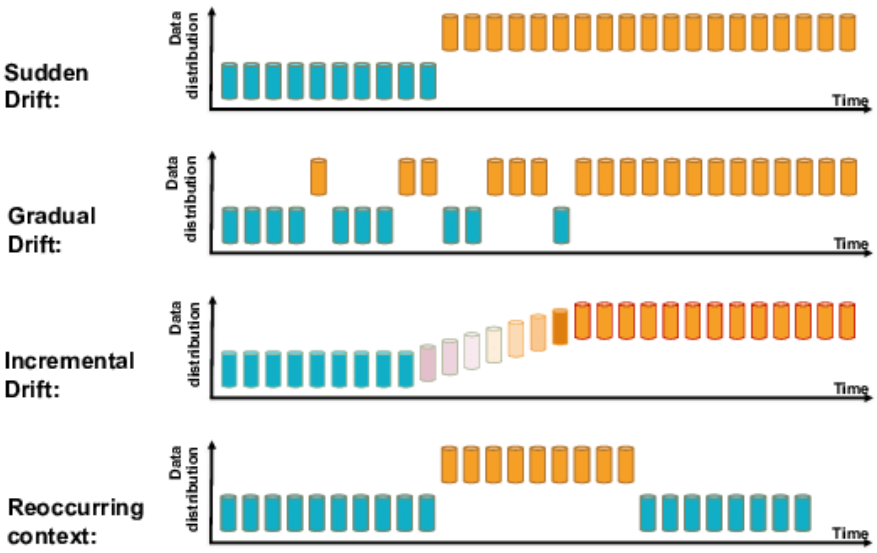

Figura 13: Tipos de Concept drift

El *concept drift* no tiene porqué presentarse únicamente de manera gradual en

todos los casos, las diferentes formas en las que puede aparecer el *concept drift* se plasman en la figura 13.

Data drift

El *data drift* ocurre cuando los datos de entrada a un modelo cambian de maneras no observadas en su fase de entrenamiento. Este cambio puede ser gradual, repentino o estacionario, $P_{entrenamiento}(Y|X) = P_{test}(Y|X)$ sin embargo $P_{entrenamiento}(X) \neq P_{test}(X)$.

Sobre este fenómeno en particular se centrará este proyecto de fin de grado. Un modelo responde generando la misma salida ante la misma entrada. El problema reside en que la manera en la que se comportan los datos de entrada cambia de formas no vistas en su fase de entrenamiento, esto desemboca en una predicción errónea de la variable de salida.

Se vuelve al modelo ejemplo, presentado en el capítulo 1, de detección de autobuses en una imagen, para ilustrar una posible existencia de *data drift*. Este podría haberse entrenado con imágenes de los autobuses de línea de la ciudad de Madrid. Con el paso del tiempo, los autobuses antiguos han ido siendo reemplazados por autobuses eléctricos de estética completamente diferente a los anteriores. El modelo, al no haber sido entrenado con las imágenes de estos nuevos autobuses, y por tanto no tener información sobre estos, cometerá una cantidad de errores mucho mayor que al principio de su etapa de puesta en funcionamiento, de esta forma, se ha perdido rendimiento. Aquí se considera que la distribución de los datos de entrada cambian debido a que las muestras de datos ya no provienen de la misma población que se usó para entrenar el modelo en primera instancia.

Este sería un claro caso de *data drift* progresivo, en el cual los datos de entrada varían lentamente con el tiempo y el modelo empeora sus predicciones progresivamente. Otro ejemplo de *data drift* fue brindado por el Covid-19, cuando la manera en la que se comportaban los datos de entrada de muchos modelos (como predictores de cantidad de gente en establecimiento, de compra en 'Amazon' etc), variaron de forma radical debido a este rápido cambio. Este sería un buen ejemplo de *Data drift* repentino, donde los datos de entrada cambian la manera en la que se comportan en un corto periodo de tiempo.

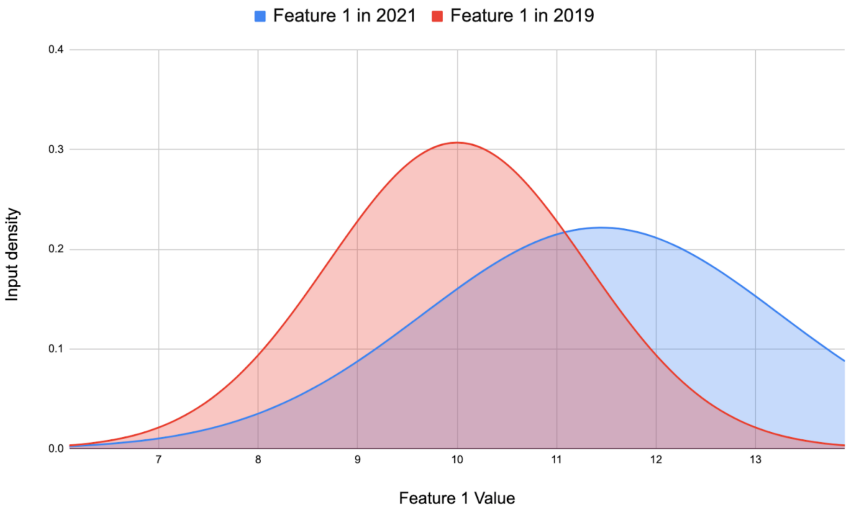

Figura 14: Data drift

En la figura 14 se muestra la distribución que sigue una variable de entrada a un modelo. Como se puede observar, en 2019, este parámetro seguía una distribución distinta a la observada en 2021, sin embargo se trata de la misma variable. Si se aprendió la manera en la que se comportaban los datos en 2019 pero se sigue utilizando este modelo en 2021, entonces no se tendrá la misma eficacia en 2021 a no ser que se ordene un reentrenamiento del mismo.

2.2. Concept drift y Data drift

Resulta importante dedicar un apartado extra a la diferenciación de estos dos tipos de drift, ya que en la actualidad, aun siendo conceptos distintos, se suelen confundir.

El *Concept drift* ocurre cuando el espacio físico, sobre el que se entrena un modelo, ya no representa la realidad, ha cambiado. Por otro lado el *Data drift* ocurre cuando los datos de entrada, que recibe el modelo, ya no se comportan de la misma manera que cuando este se entrenó.

Para ver esto con un ejemplo, se usará la figura 15

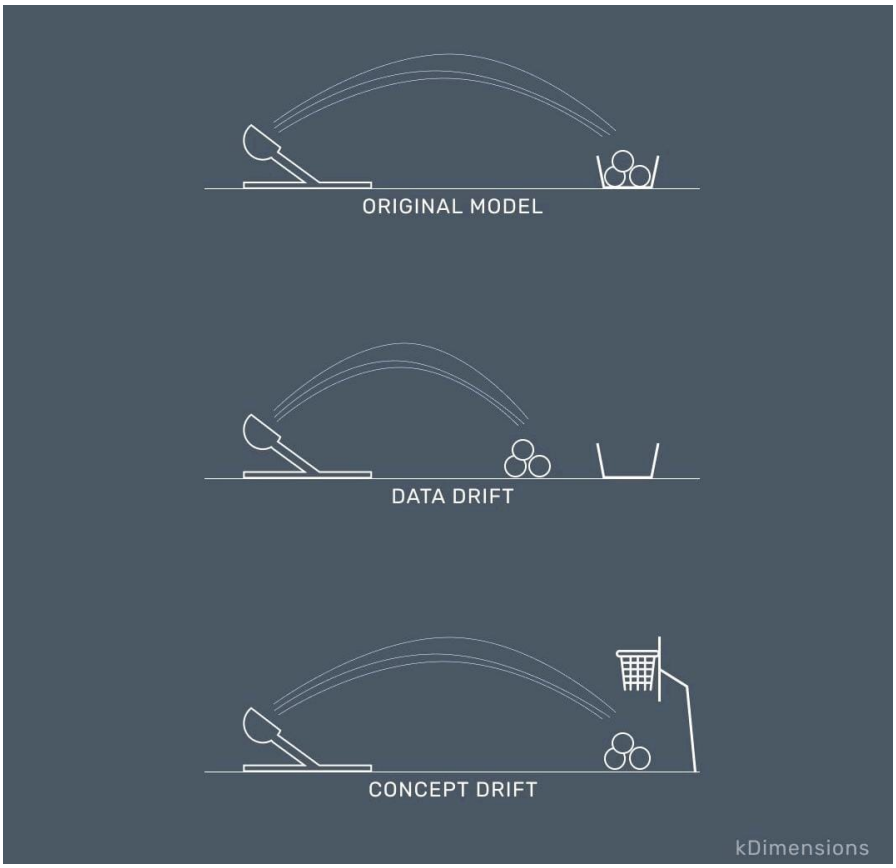

Figura 15: Tipos de drift, imagen original de www.kdimensions.com

Se quiere calcular la probabilidad que tiene una catapulta, de encestar. La cesta se encuentra a una distancia de 5 metros. Para entrenar un modelo que prediga esto, se cuenta con un único parámetro de entrada, distancia, junto con un parámetro de salida que indica si el balón quedó dentro o no.

Después de analizar los datos, se descubre que la distribución de la distancia a la que cae el balón, tiene una media de 5 metros con una desviación típica de 0,5m. El modelo se entrena con estos datos para este quedar estático. Supóngase que se tienen un 70 % de acierto inicial en las predicciones.

Después de un tiempo se descubre que la catapulta solo acierta el 30 % de las veces, y se quiere saber porqué. Podría haber ocurrido, que la manera en la que la catapulta lanza el balón ha cambiado, y de esta manera también lo ha hecho la distribución de las distancias del balón, pudiendo ahora tener una media de 3

metros. En este caso se tendría *data drift*, ya que la manera en la que se comportan los datos de entrada ha cambiado.

Sin embargo, podría haber ocurrido que la cesta hubiese cambiado su posición, encontrándose ésta en una situación más elevada. De esta forma las predicciones serían incorrectas debido a un claro caso de *Concept drift* ya que lo que ha cambiado es el espacio físico sobre el que se realizan las predicciones y no la forma en la que se comportan los datos. Aquí reside la diferencia entre estos dos tipos de *drift*

2.3. Gestión del data drift en la actualidad

La manera en que las compañías interactúan con este problema depende de lo sofisticadas que sean sus estructuras de Inteligencia Artificial. Para algunas empresas que acaban de empezar a implantar sus modelos y están trabajando para que estos tengan un adecuado funcionamiento, el *data drift* no está en su lista de prioridades que atender en el futuro inmediato. Sin embargo, estas mismas empresas podrían potencialmente encontrarse en un futuro con este fenómeno y tendrán que buscar soluciones.

Lo usual es que se trabaje con la premisa de que el *data drift* existe o bien puede llegar a existir en un futuro de manera inevitable y, cuanto más tiempo se tarde en detectar que el modelo está trabajando con unos datos que han sufrido una variación en su distribución, más tiempo estará el modelo operando con un rendimiento inferior al inicial.

Resulta interesante introducir el concepto 'ML Ops' que hace referencia a *Machine learning operations*. Como se menciona en la primera parte de este proyecto, crear y mantener un modelo de *Machine Learning* va mas allá de la simple creación del mismo, por lo que hay que tener algunos aspectos en cuenta que afectan a la creación y manutención del modelo. [5]. La creación de un modelo de aprendizaje automático es mas compleja que la mostrada en la figura 16, pero se hará referencia a estos pasos para explicar las diferentes formas con las que se trabaja para hacer frente al *data drift*.

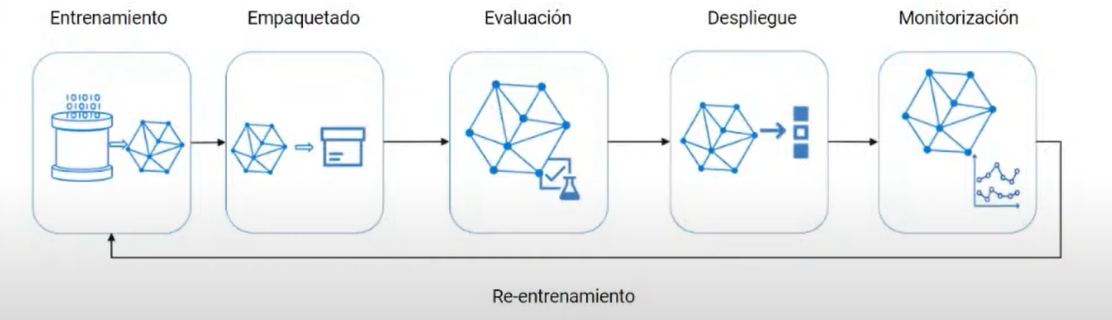

Figura 16: Machine Learning operation

Existen varias formas de hacerle frente a este fenómeno. La primera consiste en reentrenar el modelo sistemáticamente con nuevos datos de manera periódica, independientemente de que se pueda demostrar que los datos de entrada han sufrido alteración. Esta manera de afrontar el problema no requiere de ningún tipo de apoyo externo, como una evaluación de los datos de entrada para saber si se ha sufrido *data drift*. Esto consigue que el modelo esté actualizado constantemente, sin embargo, en muchas de las ocasiones se reentrenará el modelo sin una mejora del rendimiento y sin ser por tanto necesario, haciendo uso de unos recursos que no se han visto reflejados en ninguna mejora.

Por otro lado, existe una manera que busca hacer frente al problema del *data drift* de una forma más enfocada, reentrenando el modelo solo cuando se considere necesario.

En este segundo caso un *framework* monitorizará los datos de entrada al modelo dictaminando si estos sufren o no de *data drift*. En el caso de detectarse, el modelo será reentrenado.
Dentro de esta manera de afrontar el problema, existen diferentes opciones a la hora de elegir como se detecta y con qué datos se reentrena el modelo.

No hay una manera correcta o incorrecta de hacerle frente, sin embargo, dependiendo de la situación del usuario del modelo, una opción es siempre algo mas recomendable que la otra. Si se escoge reentrenar el modelo sistemáticamente, se debe tener en cuenta que se van a consumir unos recursos computacionales mayores que si se usase un *framework* de detección, sin embargo, se ahorra en el uso del recursos computacionales en la detección. Por el contrario el uso del *framework* puede reducir drásticamente el número de reentrenos innecesarios, pero con el coste de tener que analizar los datos. Estas serían las ventajas e inconvenientes de cada solución. Como regla general, el *framework* resultará especialmente útil

en los casos en los que se tengan modelos muy complejos y/o modelos diferentes que utilicen algunos datos de entrada iguales.

En la industria, un método simple que se utiliza para detectar si los datos de entrada están sufriendo algún tipo de cambio es evaluar y comparar sus propiedades estadísticas, así como la media, la mediana o la varianza. En octubre de 2021, 'Tensor Flow' [6] desarrolló un software para poder realizar esta tarea. Sin embargo, la simple evaluación de estas propiedades no es en la mayoría de ocasiones suficiente para dictaminar si los datos de entrada han sufrido *drift*. Si las métricas son diferentes puede saltar una alarma, sin embargo, que las métricas sigan la misma tendencia no significa que no se esté produciendo *data drift*.

Una manera más sofisticada de afrontar este problema consiste en someter a los datos de entrada a uno o varios test estadísticos no paramétricos, debido a que no tiene porqué conocerse de antemano qué tipo de distribuciones siguen los datos, que comparen si las dos muestras de datos provienen de la misma población. Esto se lleva a cabo a través de algún *framework* o *software* que monitorice estas distribuciones. Esto se explicará en profundidad en el capítulo 3.

2.4. Conclusiones

Se tiende a confundir el *data drift* con otros conceptos similares como el *label shift, concept drift o covariate shift*. Aun siendo conceptos diferentes, todos estos tienen en común el ser fenómenos indeseables y por tanto problemas que controlar y en el caso de sufrirlos, solucionar.

Este proyecto se basa en uno de ellos, el *data drift*. Hoy en día este fenómeno no es el más conocido ni estudiado sin embargo de sufrirse puede desembocar en grandes perdidas de rendimiento.

Hay 2 opciones para mitigar los efectos del *data drift*, la primera consiste en un reentrenamiento sistemático del modelo. En este caso se gastarían muchos recursos computacionales y se realizarían entrenos sin mejora alguna, lo que no es óptimo. Sin embargo, esta técnica mitiga los efectos del *data drift* al fin y al cabo, lo cual en algunos casos puede ser suficiente. Existe otra manera de afrontar el problema que es a través de un *framework* que monitorice los datos de entrada. Esta forma de afrontar el problema es mas sofisticada y mejor en el largo plazo, pero requiere de la monitorización constante de los datos de entrada al modelo.

3. ESTADO DEL ARTE DEL DATA DRIFT

3.1. Herramientas utilizadas en la detección de data drift

Resulta intuitivo que un primer acercamiento para detectar el *data drift* sea a través de alguna clase de test estadístico que compare dos muestras de datos y dictamine si estas provienen de la misma población.

Para elegir en cada caso el test estadístico adecuado a la situación, se deben de tener en consideración cuatro aspectos. El primero es que no se puede asumir que las distribuciones procedan de una distribución de probabilidad paramétrica, como podría ser una normal Gaussiana. El segundo aspecto a tener en cuenta es el tipo de variables que componen la distribución, ya que estas pueden ser cuantitativas, binarias o categóricas. Se debe tener en cuenta la dimensión (número de variables) de la muestra, ya que se utilizarán test estadísticos diferentes para comparar muestras univariantes y multivariantes. Por último, se tendrá en consideración el tamaño de la muestra que se va a analizar.

Teniendo esto en cuenta, los test estadísticos que se utilizan hoy en día para la detección de *data drift* se presentan a continuación.

3.1.1. Test estadísticos univariantes

Kolmogorov-Smirnov test

El Kolmogorov-Smirnov test, comúnmente conocido como K-S test, se trata de una prueba estadística que compara la distribución acumulativa de probabilidad de dos muestras.[7] La hipótesis nula de este test dictamina que las dos muestras provienen de la misma población. Si la hipótesis nula se rechaza, entonces se puede concluir que las distribuciones proceden de poblaciones distintas. Es necesario destacar que este test solo puede ser usado para variables que sigan una distribución de valores cuantitativos.

La prueba de Kolmogorov-Smirnov mide la distancia máxima absoluta que existe entre las distribuciones de las dos muestras analizadas, como se puede observar en la figura 17.

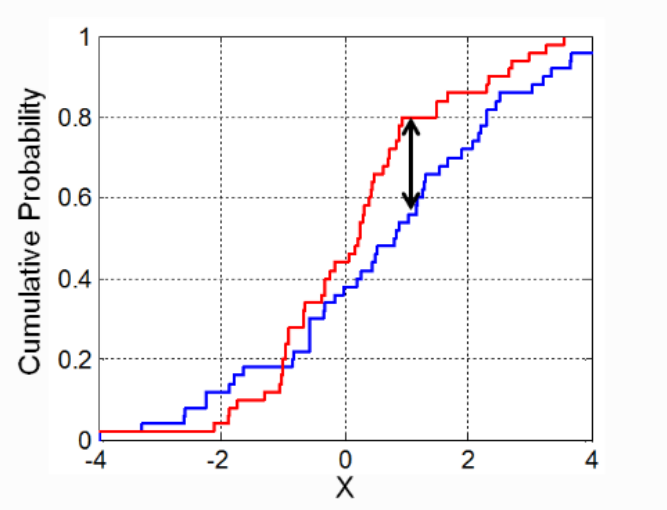

Figura 17: Distancia entre 2 funciones acumulativas de probabilidad

Después, el valor de esta distancia absoluta se compara con una distancia crítica D_{crit} a un nivel determinado de confianza. El valor de este parámetro varía dependiendo de la confianza que se requiera en cada caso. Esta distancia crítica es calculada teniendo en cuenta la distancia máxima que existe entre función de probabilidad acumulativa de la muestra y la función acumulativa de probabilidad teórica de la misma. Este caso se puede observar en la figura 18.

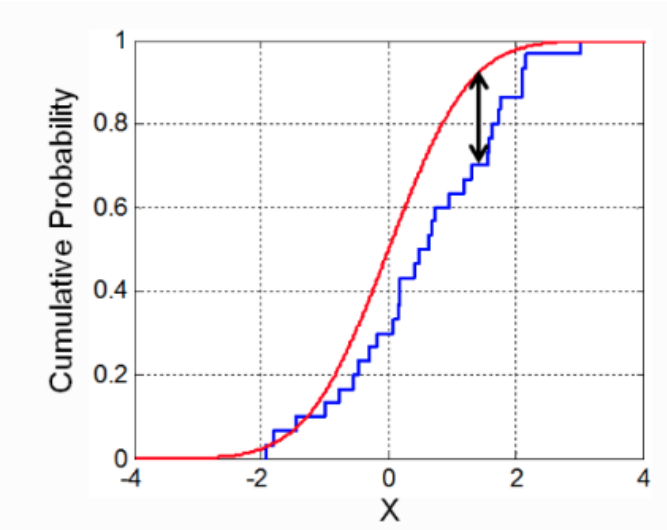

Figura 18: Distancia entre la función acumulada de probabilidad real y su teórica

En la práctica, este valor se obtiene consultando una tabla que proporciona distintos valores de distancias críticas dependiendo del nivel de confianza requerido y tamaño de los datos de la muestra. La cabecera de esta tabla tiene el siguiente aspecto 19.

Test de Kolmogorov-Smirnov sobre Bondad de Ajuste

	Nivel de significación α							
n	**0.20**	**0.10**	**0.05**	**0.02**	**0.01**	**0.005**	**0.002**	**0.001**
1	0.90000	0.95000	0.97500	0.99000	0.99500	0.99750	0.99900	0.99950
2	0.68337	0.77639	0.84189	0.90000	0.92929	0.95000	0.96838	0.97764
3	0.56481	0.63604	0.70760	0.7 8456	0.82900	0.86428	0.90000	0.92065
4	0.49265	0.56522	0.62394	0.68887	0.73424	0.77639	0.82217	0.85047
5	0.44698	0.50945	0.56328	0.62718	0.66853	0.70543	0.75000	0.78137

Figura 19: Cabecera de la tabla del test K-S

Sin embargo, este test cuenta con algunas limitaciones, como el hecho de solo poder ser utilizado para la comparación entre distribuciones continuas.

Population Stability Index

Population stability index (PSI) [8]se trata se un índice cuyo valor cuantifica cuánto ha cambiado una muestra de datos en el tiempo. Este test es utilizado para prevenir el *data drift* en sus etapas tempranas de aparición. Por esta razón se usa para detectar una posible razón de una pérdida de rendimiento del modelo.

Para obtener este índice se procede a implementar los siguientes pasos. Primero se deben ordenar todos los datos de menor a mayor para después calcular sus deciles. Una vez hecho esto, calcular en columnas el porcentaje de datos que hay en cada decil, la diferencia entre estas columnas y por último el logaritmo de esta diferencia. Una aplicación típica se muestra en la figura 20. En este ejemplo se pretende comparar la distribución de las temperaturas diarias en invierno de Nueva York en dos espacios durante dos espacios de tiempo diferentes.

	Temperaturas	deciles	Deciles_sin	Deciles_con	%Sin	%Con	Sin-Con	ln(Sin/Con)	PSI
0	-14.515	1	22.0	2.0	0.004693	0.000427	0.004266	2.398109	0.010231
1	-10.680	2	75.0	32.0	0.015998	0.006824	0.009174	0.851965	0.007816
2	-6.845	3	291.0	155.0	0.062073	0.033056	0.029017	0.630111	0.018284
3	-3.010	4	696.0	437.0	0.148464	0.093197	0.055267	0.465630	0.025734
4	0.825	5	951.0	803.0	0.202858	0.171252	0.031606	0.169373	0.005353
5	4.660	6	1204.0	1129.0	0.256826	0.240776	0.016050	0.064530	0.001036
6	8.495	7	899.0	1118.0	0.191766	0.238430	-0.046664	-0.217800	0.010163
7	12.330	8	417.0	667.0	0.088951	0.142248	-0.053297	-0.469491	0.025023
8	16.165	9	114.0	270.0	0.024317	0.057582	-0.033264	-0.862010	0.028674
9	20.000	10	19.0	70.0	0.004053	0.014929	-0.010876	-1.303843	0.014180

Figura 20: Tabla para calcular el valor del índice PSI

De esta manera las temperaturas entre $-14{,}5$ ℃ y $-10{,}68$ ℃ pertenecerán al primer decil.

Por último, el índice PSI entre dos *dataframes* A y B se calcula a través de la formula siguiente 1.

$$PSI = \sum(\%A - \%B) * \ln(\frac{\%A}{\%B}) \tag{1}$$

Si el valor del índice es menor que 0,1, se puede suponer que las dos muestras provienen de la misma población. Si el valor se encuentra entre 0,1 y 0,2, se considera que sí que se ha sufrido algún cambio, pero no demasiado exagerado. Valores mayores a 0,2 en el índice PSI indican que puede existir un cambio significativo entre las dos poblaciones.

Chi Square goodnes of fit test

Chi Square goodnes of fit [9] se trata de un test utilizado cuando se requiere estimar si una muestra procede de una determinada población. Este acepta valores categóricos y por esta razón es una buena opción para la detección de *data drift* en variables que pertenezcan a esta categoría.

La hipótesis nula de este test dictamina que las dos muestras a comparar siguen la misma distribución, por lo tanto el rechazo de esta hipótesis, a un nivel de confianza determinado, haría saltar las alarmas por la posible existencia de *data drift*.

Una muestra está compuesta por los datos de entrada al modelo y la otra en este caso consta de los datos con los que se entrenó originalmente el modelo. El test cuenta de varios pasos previos al cálculo de su estadístico.

Primeramente se contabiliza el número de veces que se repite cada variable categórica, tanto en una muestra como en la otra. Después se restan estas dos columnas y se computa un error. Este error es elevado al cuadrado para darle más importancia a los errores más grandes. Finalmente este error al cuadrado es dividido por el número de veces que se espera se repita cada valor y todos estos valores son sumados. El resultado de esta última suma se trata del valor del test estadístico.

Para poder entender estos pasos con un ejemplo real, el lector puede imaginarse que una tienda en el centro de Madrid tiene un modelo de inteligencia artificial que tiene como objetivo predecir las ventas de su tienda dependiendo del día de la semana. Como variable de entrada cuenta con cuantos clientes realmente visitan la tienda. Los datos de la muestra con la que se entrenó y la muestra que se quiere analizar se muestran en la figura 21.

	Dia semana	Clientes Dataset Entrenamiento	Clientes Dataset Analizado	Error	Error al cuadrado	Error al cuadrado / Esperado
0	Lunes	180	170	10	100	0.555556
1	Martes	220	190	30	900	4.090909
2	Miercoles	440	520	-80	6400	14.545455
3	Jueves	390	400	-10	100	0.256410
4	Viernes	550	500	50	2500	4.545455
5	Sabado	780	640	140	19600	25.128205
6	Domingo	380	280	100	10000	26.315789

Figura 21: Tabla para calcular el valor del índice Chi-Sqrt

El estadístico del test es el resultado de la suma de la última columna. Finalmente para poder llegar a una conclusión y determinar de esta manera si existe *data drift* o no, se compara ese valor con un valor crítico obtenido de la distribución Chi cuadrado.

Para esto se decide el nivel de confianza que se desea. Después se accede a una tabla, que ofrece unos valores críticos dependiendo de los grados de libertad. Se podrá de esta manera rechazar la hipótesis nula siempre que el estadístico sea mayor que el valor crítico.

Prueba de hipótesis para proporciones

Para poder analizar el *data drift* en variables binarias, algunas plataformas de detección de *data drift* hacen uso de una prueba de hipótesis que comprueba si las proporciones en las dos muestras de datos son iguales.

La hipótesis nula estipula que la proporción entre el *dataframe* de referencia y el analizado es la misma. Esto es, si en la referencia se obtiene un valor de 1 el 90 % de las veces, en el *dataframe* analizado, se espera una proporción similar para no rechazar la hipótesis nula.

Este test necesita que los dos *dataframes* sean independientes. Además necesita un número mínimo de observaciones, se debe al menos poder contabilizar 10 veces cada valor.

Los pasos previos al cálculo del estadístico son los siguientes; Primero, se establece la hipótesis nula y por consiguiente la hipótesis alternativa. En este caso la hipótesis nula estipula que la diferencia entre las proporciones es cero, por lo tanto, que las proporciones son iguales. Este es un test de hipótesis de 2 colas.

$$\begin{aligned} H_o &: P_1 - P_2 = 0 \\ H_a &: P_1 - P_2 \neq 0 \end{aligned} \tag{2}$$

Después se estima la proporción para los dos *dataframes*.

$$\hat{p} = \frac{p_r n_r + p_r n_a}{n_r + n_a} \tag{3}$$

Donde p_r y p_a son las proporciones y n_r y n_a el número de datos de la referencia y el análisis respectivamente.

Después se calcula el error estándar.

$$SE = \sqrt{p(1-p)(\frac{1}{n1} + \frac{1}{n2})} \tag{4}$$

Con estos datos se puede calcular el estadístico de la hipótesis.

$$z = \frac{p_r - p_a}{SE} \tag{5}$$

El estadístico Z sigue una distribución normal estándar por lo que la decisión de rechazo de la hipótesis nula dependerá del nivel de confianza escogido. Los valores se encuentran recogidos en tablas.

3.1.2. Test estadísticos multivariantes

Generalmente, los modelos de inteligencia artificial, no trabajan con una única variable de entrada, normalmente se trabaja con un espacio multidimensional. Esto implica tener varios tipos de datos pertenecientes a diferentes distribuciones. Cabe destacar que las variables de entrada pueden ser cuantitativas, binarias o categóricas.

La detección del drift univariante busca encontrar cambios en la distribución de cada variable por separado. Sin embargo, existen casos en los que esto no es suficiente para capturar los motivos por los que el modelo puede perder rendimiento.

Los datos de entrada pueden sufrir un cambio en sus correlaciones sin cambiar la manera en la que estos se distribuyen individualmente.

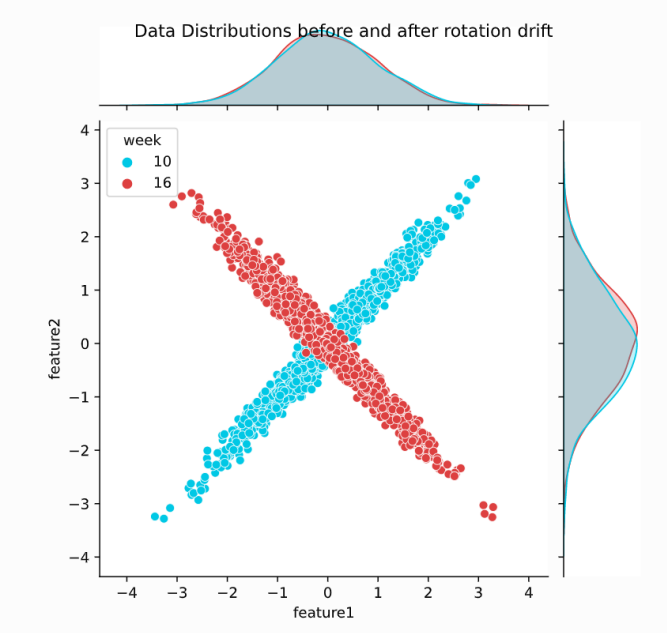

Figura 22: Cambio en la correlación de una variable multidimensional

La figura 22 muestra como la distribución univariante de la variable 1 y de l variable 2 permanecen intactas en el plano unidimensional, sin embargo se ve un claro cambio en el plano multidimensional. Con el color azul se puede observar la distribución original y con el color rojo se representa la distribución resultante después de haber sufrido un cambio en la manera en la que sus datos se correlacionan. Si no se aplica ninguna técnica de detección de *data drift* multivariante, nunca se

podría detectar que estas distribuciones habían sufrido un cambio.

Es por esta razón por la que resulta crucial aplicar métodos de detección de *data drift* multivariantes, ya que existen casos en los que la detección univariante no es suficiente. Tampoco resulta suficiente realizar únicamente un análisis multivariante, esto se verá mas adelante.

PCA error reconstruction

Para identificar el drift que no puede ser detectado en un plano únicamente univariante, se puede utilizar una técnica que se basa en la evaluación del error de la reconstrucción de los datos con PCA.

Principal Component Analysis (PCA) [10] es un método estadístico que permite simplificar la complejidad de espacios multidimensionales a la vez que conserva su información. Un caso de uso podría darse con una muestra de N alumnos, cada uno de ellos con M variables que les definen, de esta manera, se trabaja en un espacio de M dimensiones. Esta técnica permite encontrar un número de factores subyacente (Z<M) que tratarán de explicar lo mismo que las M variables originales. Donde antes se necesitaban M variables para describir a un alumno, ahora puede hacerse solo con Z variables, cada una de estas se denomina componente principal.

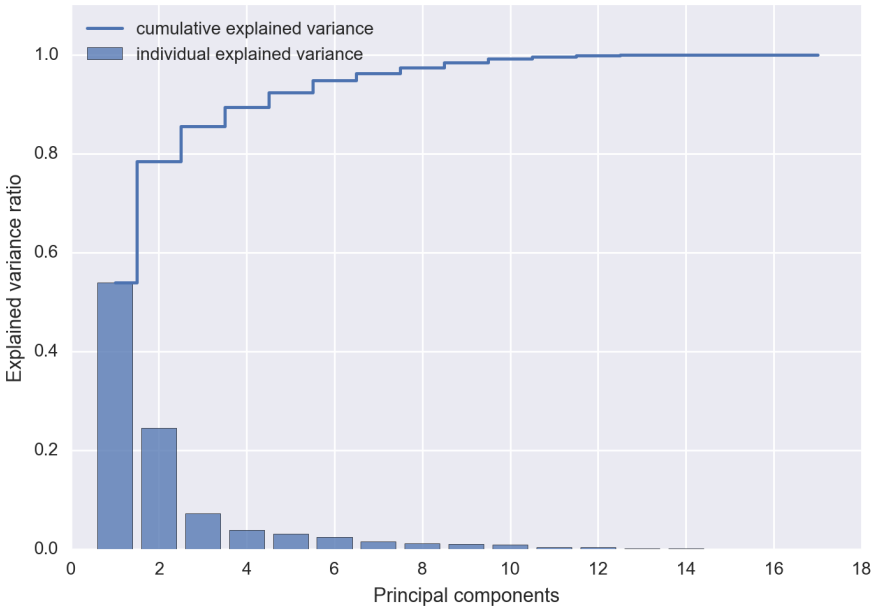

Figura 23: Varianza explicada en función del número de factores subyacentes (Z) utilizados

La figura 23 permite observar como con pocas componentes principales se puede explicar gran parte de la varianza. Es por esto que PCA es utilizado para descargar de información no potencialmente relevante.

El método de PCA permite por lo tanto condensar la información aportada por múltiples variables en solo unas pocas componentes. Esto lo convierte en un método muy útil de aplicar previo al uso de otras técnicas o en este caso, para el aprendizaje de una estructura con la que después se reconstruirán los datos del *dataframe* de análisis, pero esto se explicará mas adelante.

Este método es capaz de detectar complejos cambios en la distribución multivariante de los datos. Es importante destacar que este algoritmo permite la detección multivariante en todo tipo de variables, tanto numéricas, como categóricas. Las variables de tipo binario, son consideradas categóricas con dos clases.

Para poder realizar esto, el primer paso consiste en la preparación de los datos. Esto incluye rellenar los datos que faltan haciendo uso de técnicas de escalado [11]. Que no falte ningún dato es un requisito indispensable.

Para poder hacer uso de variables de tipo categórico se emplea una técnica denominada codificación de la frecuencia [12]. Resulta intuitivo que para poder hacer uso de una variable tipo categórica en un test que solo admite valores numéricos, se le asigne un valor a cada categoría y mas tarde se haga uso de esta nueva distribución para implementar el test. Sin embargo, esta técnica no es la más eficáz ya que cuando las categorías pasan a ser números, unas tienen más peso que otras por el hecho de llevar un valor numérico asignado. Por esta razón se utiliza la codificación de la frecuencia.

A cada categoría se le asigna un valor numérico dependiendo de las veces que esta se repite dentro del *dataframe*. De esta forma se da más importancia a los datos que más veces se repiten y así, si estos datos dejan de presentarse con la misma frecuencia, será mas fácil detectar el *data drift*.

Después de hacer esto, se estandariza la distribución a media 0 y a varianza unidad, esto se hace para asegurar que todos los parámetros contribuyen por igual.

El segundo paso consiste en la reducción dimensional. Los *softwares* que utilizan este método para la detección de *data drift* tienen como parámetro de entrada capturar hasta el 65 % de la varianza del *dataframe* pero sin embargo, este parámetro puede ser cambiado, ya que se puede obtener mas o menos varianza cambiando el número de componentes principales que se utilicen.

El algoritmo de PCA es aplicado al *dataframe* referencia que es el que se considera que no tiene *drift*. Después, la transformación aprendida se le aplica a cada uno de los diferentes *data chunks* de la muestra analizada. Es importante tener en cuenta que PCA captura la estructura interna de los datos e ignora cualquier ruido que el *dataframe* pueda presentar.

El tercer paso consiste en descomprimir cada uno de los *data chunks* del análisis que han sido transformados. Esto se consigue devolviendo los datos al plano multivariante original a través del patrón aprendido de la referencia, pero esta vez a la inversa.

A esta nueva distribución se le aplica la distancia euclídea entre los datos originales y sus reconstruidos. Las resultantes distancias son agregadas para obtener su media. El valor final es el llamado error de reconstrucción.

PCA se encarga de aprender la estructura interna de los datos. Un error signifi-

cativo en la reconstrucción de los datos denota que estos ya no siguen la misma distribución aprendida en la referencia, por lo tanto se estaría lidiando con *data drift*. En cambio, si después de reconstruir los datos el error no es muy grande, se puede considerar que las dos distribuciones provienen de la misma población.

Cuando el algoritmo de PCA es aplicado, se pierde alguna información del *dataframe*. Esto significa que los datos reconstruidos siempre serán algo diferentes a los originales lo cual se refleja en el error de reconstrucción. Se espera de este parámetro ser constante en el tiempo. Si el error aumenta significativamente, se puede concluir que se trata del resultado de reconstruir con unos datos que posiblemente pertenezcan a otra población.

Debido al ruido que existe en los *dataframes* del mundo real, siempre existirá una variabilidad en el error de reconstrucción, aunque todos pertenezcan a la misma distribución que la referencia. Se debe por tanto analizar los *dataframes* de la referencia, cuya distribución se conoce, y establecer unos limites de varianza en el error de reconstrucción.

Es importante mencionar que al aplicar el algoritmo PCA no todas las variables tendrán el mismo peso. Las primeras componentes principales explican significativamente más cantidad de varianza que las últimas escogidas. Es por esto que se debe sacar la siguiente conclusión; Un error grande en la reconstrucción de los datos con PCA es motivo suficiente para pensar que al menos una de las variables de entrada al modelo ha sufrido un *drift*. Sin embargo, que no se considere que exista un error grande en la reconstrucción no significa que todas las variables estén exentas de *data drift*.

Hay variables que contribuyen más o menos a las componentes principales del algoritmo PCA, por lo que el *data drift* en alguna de estas variables será detectado en la distribución multivariante. Por otro lado, una variable que no tenga mucho peso en la componente principal de la estructura PCA aprendida para la reconstrucción, puede sufrir *drift* en el plano univariante sin que este test lo detecte en el plano multivariante. Es por esta misma razón por la que la detección de *drift* univariante y multivariante deben trabajar conjuntamente.

Generalización multivariante del Wald-Wolfowitz runs test

Este test estadístico se basa en una generalización multivariante de un test diseñado para distribuciones univariantes llamado *Wald-Wolfowitz runs test* [13]. Para poder explicar esta generación, primero se deben presentar las bases del test estadístico univariante.

El *run test* propuesto por Wald-Wolfowitz tiene como objetivo comprobar si dos muestras univariantes pertenecen a la misma distribución. La hipótesis nula dictamina que las dos muestras provienen de la misma población. Para verificar esto se siguen una serie de pasos.

Primeramente se ordenan todos los valores de ambos *dataframes*, de esta manera se genera una lista que contienen todos los valores de las dos muestras ordenadas de menor a mayor. Después, se sustituye el valor numérico por uno categórico dependiendo del *dataframe* al que pertenezcan. Se asigna en la lista la categoría X si un valor pertenece a un *dataframe* o Y si pertenece al otro. De esta forma, la lista que antes estaba formada formada por valores numéricos, pasa a estar formada por valores categóricos.

Después se procede a contar el número de veces que X o Y aparecen de manera consecutiva en la lista anteriormente generada. Cada vez que se termine una secuencia, se considerará un *run*. De este modo si los valores están ordenados de la forma $XXXYYXY$ se tendrán 4 *runs*. Como concepto general, cuanto mayor sea el número de *runs*, más similares se considerarán las 2 distribuciones. Sin embargo, la decisión de si rechazar o no la hipótesis nula, no se basará en este número solamente. Esto es porque se debe tener en cuenta el tamaño de las 2 muestras. La decisión se basa en el test estadístico W que se calcula siguiendo la ecuación 6.

$$W = \frac{R - \frac{2mn}{N} - 1}{(\frac{2mn(2mn-N)}{N^2(N-1)})^{\frac{1}{2}}} \tag{6}$$

Donde R es el número de *runs*, m y n son el número de datos de cada muestra y N el número de datos totales.

El estadístico W pertenece a una distribución normal estandarizada, por lo que para rechazar la hipótesis nula, a un nivel determinado de confianza se hará uso de las tablas de la normal estandarizada.

La generalización de este test al plano multidimensional se basa en el uso del *mi-*

nimal spamming trees(MST) [14]. Gracias a este algoritmo se conectan los puntos en el plano multidimensional. De esta forma se consigue obtener un gráfico con una serie de nodos y de bordes. Se denominan bordes a los puntos en el espacio y nodos a las conexiones entre estos.

Para poder llevar a cabo la búsqueda del estadístico que indique si los dos *dataframes* son similares o no, se siguen una serie de pasos. Primero se posicionan todos los puntos en el plano multidimensional. Después, se sustituyen sus valores por el símbolo de asterisco, si se pertenece a un *dataframe* y un cuadrado si se pertenece al otro. En un caso de uso bidimensional podría tener este aspecto 24.

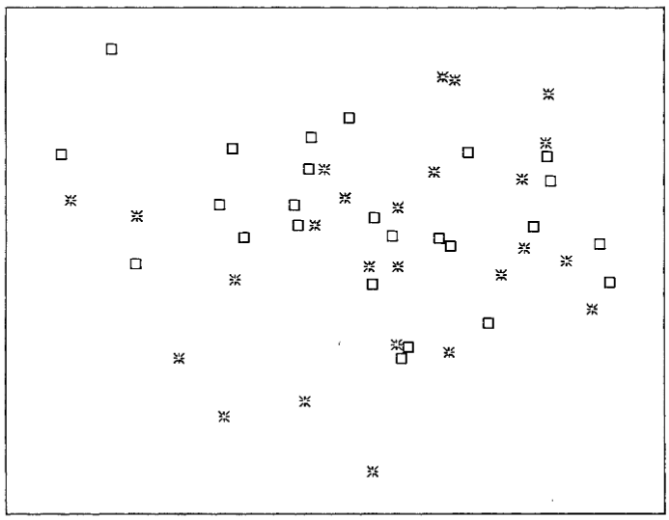

Figura 24: Puntos de distintos *dataframes* representados en plano bidimensional

Después se hace uso del algoritmo MST para conectar todos los puntos como se observa en la figura 25.

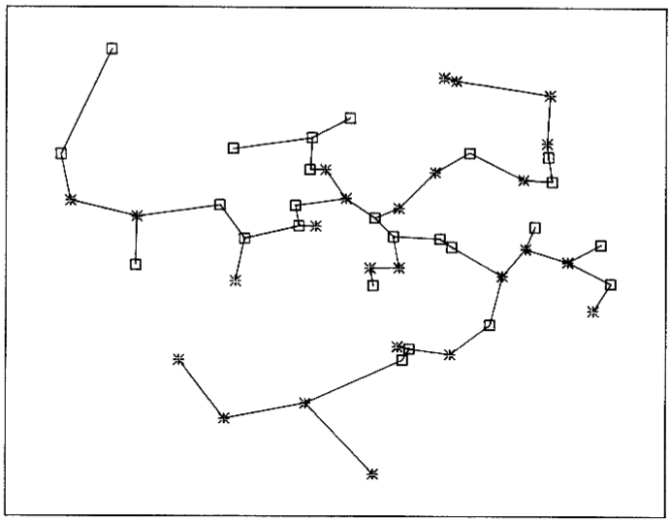

Figura 25: Unión de puntos en el espacio resultante de usar el algoritmo MST

Por último, se eliminan los nodos que conectan bordes de *dataframes* distintos.26

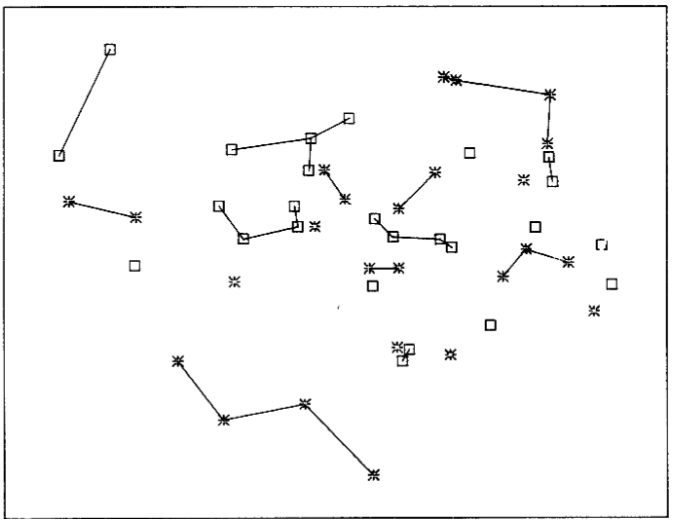

Figura 26: Subconjuntos en el plano bidimensional

Esto deja un número de subconjuntos del mismo *dataframe* conectados por nodos. Lo que en el plano univariante era llamado *run*, ahora es llamado subconjunto. Cuanto mayor sea este número, más similares se espera que sean las 2 distribuciones.

Finalmente, y similar al caso univariante, se obtiene un test estadístico W. R en

este caso es el número de subconjuntos. E[R] es el valor esperado de R. De la misma manera que en el caso univariante m y n representan el tamaño de los 2 *dataframes* y N el número total de puntos en el plano. W sigue una distribución normal estandarizada.

$$W = \frac{R - E[R]}{\sqrt{\frac{2mn(2mn-N)}{N^2(N-1)}}} \tag{7}$$

$$E[R] = \frac{(2mn)}{N} + 1 \tag{8}$$

3.2. Frameworks de detección de Data Drift

El *data drift* se muestra como un problema relativamente reciente y por tanto no hay una manera definida de hacerle frente. Debido a que lo que se busca es evitar la pérdida de rendimiento del modelo, y el *data drift* es solo uno de los motivos que puede causar esto, encontramos la detección de *data drift* en algunos casos dentro de *softwares* dedicados a la detección de la pérdida de rendimiento.

A continuación se presentan tres *softwares* que implementan algún tipo de monitorización para la detección de variaciones en los datos de entrada.

3.2.1. NannyML

NannyML se trata una librería de código abierto que tiene como objetivo estimar el rendimiento de un modelo una vez este ha sido puesto en funcionamiento [15]. La eficacia del modelo es estimada sin la necesidad de proporcionar al software de la información necesaria para comparar las predicciones realizadas por el modelo con la información real. Aquí reside la gran ventaja que pretende aportar esta librería.

Para estimar el rendimiento del modelo cuenta con un detector de *data drift* en el cual se apoya. Esto es debido a que se espera que la aparición de un cambio en la distribución de los datos desemboque en una perdida de rendimiento.

En su apartado de detector de *data drift*, destaca por poder ser implementado de manera conjunta al modelo. Esto permite al usuario analizar los datos de entrada a tiempo real. De esta forma, poder detectar el *data drift* en sus etapas más tempranas.

La forma en la que el detector de *data drift* funciona, se resume en comparar cada *data chunk* de datos entrante al modelo con la distribución que se utilizó para entrenar al modelo en primera instancia. Los datos de entrada inspeccionados son una parte de lo que se denomina *dataframe* de análisis. Cada *data chunk* de datos de entrada se compara con el *dataframe* de referencia. Se evalúa entonces como de parecidas son las distribuciones. Esto se realiza a través de varios test estadísticos.

Para la detección en el plano unidimensional, esta librería hace uso de los test estadísticos de K-S, para variables de tipo cuantitativas y de el test Chi cuadrado para variables categóricas. Si en una de las columnas se dispone de una variable de tipo binaria, el software la tomará como una variable categórica de dos clases y por tanto evaluará el *data drift* en esta distribución haciendo uso del test Chi

cuadrado. Para la detección de *data drift* multivariante, la librería utiliza la técnica de reconstrucción del error por PCA.

NannyML destaca por ser un software gratuito y muy completo con una interfaz de uso sencilla y unas ilustraciones de los valores de los estadísticos para los diferentes *data chunks* que resultan muy intuitivas.

Un ejemplo de esta interfaz la podemos ver en la ilustración 27 en la que se aplica el test estadístico de Chi cuadrado para distintos *data chunks* para evaluar su *data drift*.

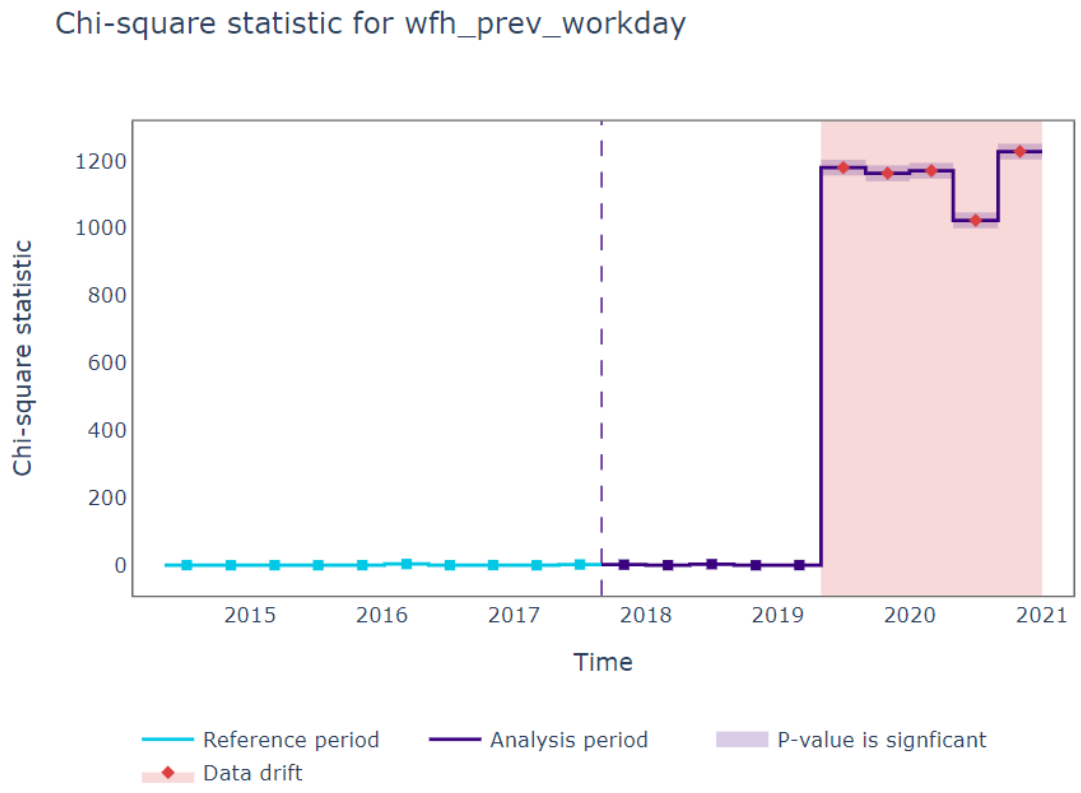

Figura 27: Test chi cuadrado en NannyML

3.2.2. EvidentlyAI

EvidentlyAI es una librería de python de código abierto enfocada al uso por parte de científicos e ingenieros de datos [16]. Este software ayuda a evaluar, testear y monitorizar el rendimiento de un modelo de inteligencia artificial desde su validación hasta su producción. Se puede ver como una capa más en el proceso para obtener un modelo funcional.

Este software, se encarga de realizar tres tareas. La primera se basa en la monitorización y evaluación de los datos de entrada al modelo. Esto es, evaluación de *data drift*. La segunda tarea se resume en la monitorización de las predicciones. De esta manera se busca encontrar fallos en el modelo descubiertos por un cambio en la distribución en las predicciones del modelo. Por último, y de manera muy similar a NannyML, este software busca analizar el rendimiento del modelo a través de diferentes técnicas. Cabe destacar, que el rendimiento solo puede ser analizado para modelos de clasificación.

EvidentlyAI analiza el *data drift* únicamente de manera univariante, sin embargo su *framework* consta de plenitud de test estadísticos. Primeramente realiza una diferenciación entre los *dataframes* que cuentan con mas de 1000 observaciones y los que no. Después se utilizarán test estadísticos diferentes para variables categóricas y numéricas en cada caso.

Para los *dataframes* que cuenten con menos de 1000 observaciones, se utilizará el K-S test para variables cuantitativas y el Chi cuadrado para las categóricas. Cuando el número de observaciones supere los 1000 datos entonces se utilizará el PSI para variables cuantitativas y un test estadístico llamado Jensen Shanon divergence para variables categóricas.

Una de las grandes ventajas de este software es que presenta la evaluación de cada variable por separado en un formato .html muy intuitivo y eficaz. 28

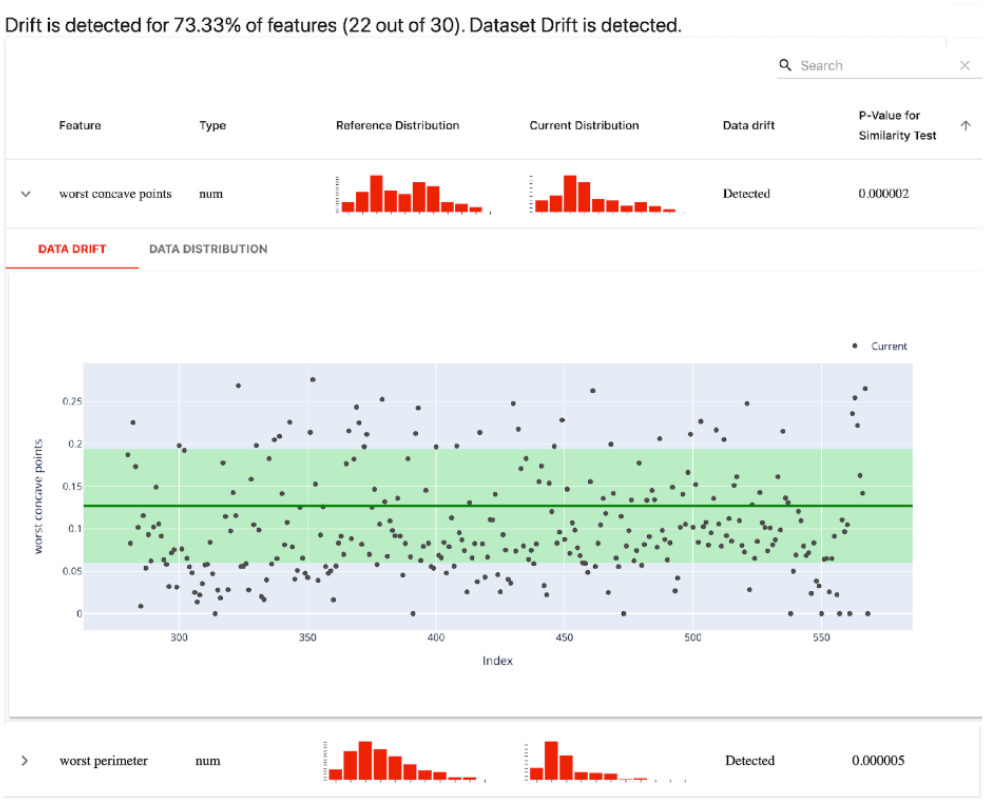

Figura 28: Archivo .html donde Evidently muestras evaluación del *data drift* variable por variable

3.2.3. Azure ML

Azure Machine Learning es un servicio en la nube que permite acelerar y administrar el ciclo de vida de los proyectos de aprendizaje automático [17]. A diferencia de los dos anteriores, Azure ML no se trata de un software de código abierto y por tanto no es gratuito. Los profesionales de aprendizaje automático, científicos de datos e ingenieros pueden usarlo en sus flujos de trabajo diarios: entrenamiento e implementación de modelos y administración de MLOps.

Se puede crear un modelo en Azure Machine Learning o usar un modelo creado a partir de una plataforma de código abierto, como Pytorch, TensorFlow o Scikit-learn. Las herramientas de MLOps pretenden ayudar al usuario a supervisar, volver a entrenar y volver a implementar modelos.

Azure Machine Learning se trata de un software muy completo y dentro de sus posibilidades se encuentra la detección de *data drift*. En este sentido este software

permite analizar las distribuciones naturales de los datos de entrada, monitorizar estos datos en el tiempo y advertir cuando se considere que estos se encuentran sufriendo un *drift*.

Cuenta con una interfaz para presentar el *data drift* muy intuitiva 29.

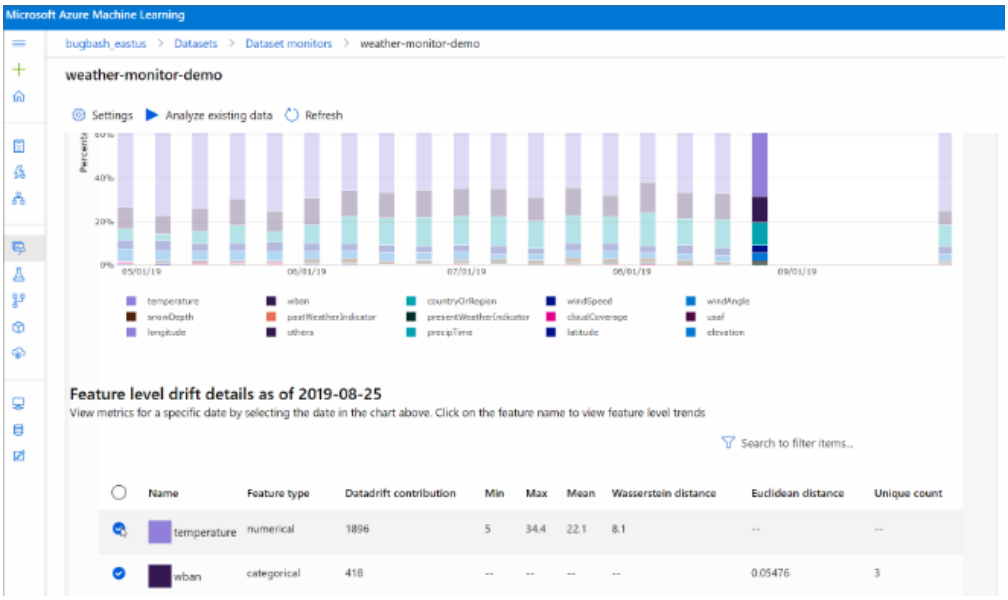

Figura 29: Interfaz Azure ML

Azure ML hace uso del test estadístico *Wasserstein distance* para variables cuantitativas [18]. A parte, también monitoriza y evalúa la media, el valor mínimo y el máximo. Para valores categóricos este software hace uso de la distancia euclídea.

3.3. Conclusiones

Después de analizar todas las tecnologías, se estudia si existe alguna que cumpla con todos los requisitos impuestos en el capítulo 1. Estos requisitos exigen que:

- Se deben de poder analizar los datos de entrada tanto de manera univariante como multivariante.
- Se debe de poder analizar el *data drift* en variables categóricas, numéricas y binarias.
- El tamaño del *dataframe* analizado debe poder ser variado.
- Debe contar con una lógica que aplique unos test estadísticos u otros dependiendo del *dataframe* analizado
- Debe ser gratuito de software abierto

En la siguiente tabla se evalúa cada tecnología de detección teniendo en cuenta el cumplimiento o no de los requisitos impuestos.

Software	Requisito 1	Requisito 2	Requisito 3	Requisito 4	Requisito 5
NannyML	✓	✓	✓	✗	✓
Evidently AI	✗	✗	✓	✓	✓
Azure ML	✗	✗	✓	✗	✗

Como se puede observar, no hay ninguna tecnología que cumpla todos los requisitos. Adicionalmente, el *framework* que en este trabajo se propone enriquece las propuestas anteriores, sin incrementar la complejidad de las mismas. Se cuenta con un número mayor de test y un análisis mas concretamente en el apartado multivariante.

4. PROPUESTA DE UN FRAMEWORK DE DETECCIÓN DE DATA DRIFTING

4.1. Elección de los test estadísticos

El *framework* trabajará analizando muestras de datos compuestas por una o varias dimensiones.

Para la detección del *data drift* en el plano unidimensional, en variables de tipo cuantitativas, se proponen 2 test estadísticos. Se puede elegir entre el test de *Kolmogorov-Smirnov* o *Population Stability Index*.

Justificación del no uso del estadístico PSI

El método PSI fue un estadístico que se diseñó pensando específicamente en la detección de *data drift* por lo que resulta intuitivo usar este, sin embargo este test, de usarse, debería ser con precaución ya que cuenta con algunas limitaciones.

Como se comenta en el capitulo 2, el PSI es un test estadístico que primeramente ordena los datos y los clasifica por deciles. Después con el valor de la cantidad de datos en cada decil, realiza una serie de operaciones matemáticas y es entonces cuando se obtiene un valor.

El PSI presenta problemas a la hora de detectar *data drift* en *dataframes* que presenten un número de observaciones pequeño o en *dataframes* donde la distribución de los datos es muy amplia. No se puede utilizar el estadístico PSI cuando el *dataframe* de referencia no tiene el mismo número de observaciones que el de análisis, por lo que, el PSI de usarse, debería ser en un *dataframe* donde cada *data chunk* sea del mismo tamaño que toda la referencia. Esto mas que un problema resulta una limitación. Sin embargo el caso explicado a continuación si que limita el uso del estadístico a unos pocos casos.

En el caso de trabajar con un *dataframe* que cuente con un número pequeño de observaciones, existe la posibilidad de que no exista ningún valor en un decil determinado, y por lo tanto el valor de esa casilla sea 0. Debido a que dentro de las operaciones matemáticas utilizadas por este test se encuentra el logaritmo natural. La existencia de un 0 en unas de las casillas de recuento de deciles genera automáticamente un problema, ya que el logaritmo natural de 0 es un número no definido. Esto desemboca en un valor de PSI que queda fuera de lo esperable.

```
In [8]: psi = calcular_psi(df_sin,df_con[(df_con["YEAR"]>=1981)&(df_con["YEAR"]<=1983)])
        psi.head(10)

        El valor del psi es= inf
```

Out[8]:

	Temperaturas	deciles	Deciles_sin	Deciles_con	%Sin	%Con	Sin-Con	ln(Sin/Con)	PSI
0	-14.515	1	22.0	0.0	0.004693	0.000000	0.004693	inf	inf
1	-10.680	2	75.0	8.0	0.015998	0.022222	-0.006224	-0.328611	0.002045
2	-6.845	3	291.0	15.0	0.062073	0.041667	0.020407	0.398616	0.008134
3	-3.010	4	696.0	41.0	0.148464	0.113889	0.034575	0.265120	0.009167
4	0.825	5	951.0	53.0	0.202858	0.147222	0.055636	0.320565	0.017835
5	4.660	6	1204.0	83.0	0.256826	0.230556	0.026270	0.107907	0.002835
6	8.495	7	899.0	78.0	0.191766	0.216667	-0.024900	-0.122083	0.003040
7	12.330	8	417.0	54.0	0.088951	0.150000	-0.061049	-0.522555	0.031902
8	16.165	9	114.0	21.0	0.024317	0.058333	-0.034016	-0.874981	0.029763
9	20.000	10	19.0	5.0	0.004053	0.013889	-0.009836	-1.231656	0.012115

Figura 30: Ejemplo de fallo en el caso de uso incorrecto del test PSI

Para ver esto con un ejemplo se propone el caso en el que se requiere comparar 2 *dataframes*. Los 2 representan las temperaturas registradas durante los meses de invierno en la ciudad de Nueva York. El *dataframe* de referencia contiene las temperaturas desde el año 1869 hasta el año 1908. El *dataframe* de análisis contiene las temperaturas registradas desde el año 1909 hasta la actualidad.

Debido al calentamiento global, se sabe que estas dos muestras de datos no siguen exactamente la misma distribución, especialmente los *data chunks* que describen los últimos años. Si el tamaño en el que se divide el *dataframe* de análisis no es lo suficientemente grande, puede ocurrir que algún rango de decil no quede contemplado, desembocando en un 0 en alguna de las casillas.

En la figura 30 se puede observar la tabla PSI resultante de de comparar el *dataframe* de referencia, que contiene un total de 39 años de datos con un *data chunk* del *dataframe* de análisis, de tamaño 3 años. Debido a que durante los últimos años las temperaturas extremadamente frías en invierno son más improbables de aparecer, el primer decil puede no contener ningún valor y por tanto dar error.

Si se analiza el valor del estadístico PSI y del estadístico K-S, a través de los diferentes *data chunks* del *dataframe* de análisis, se obtiene lo siguiente 31

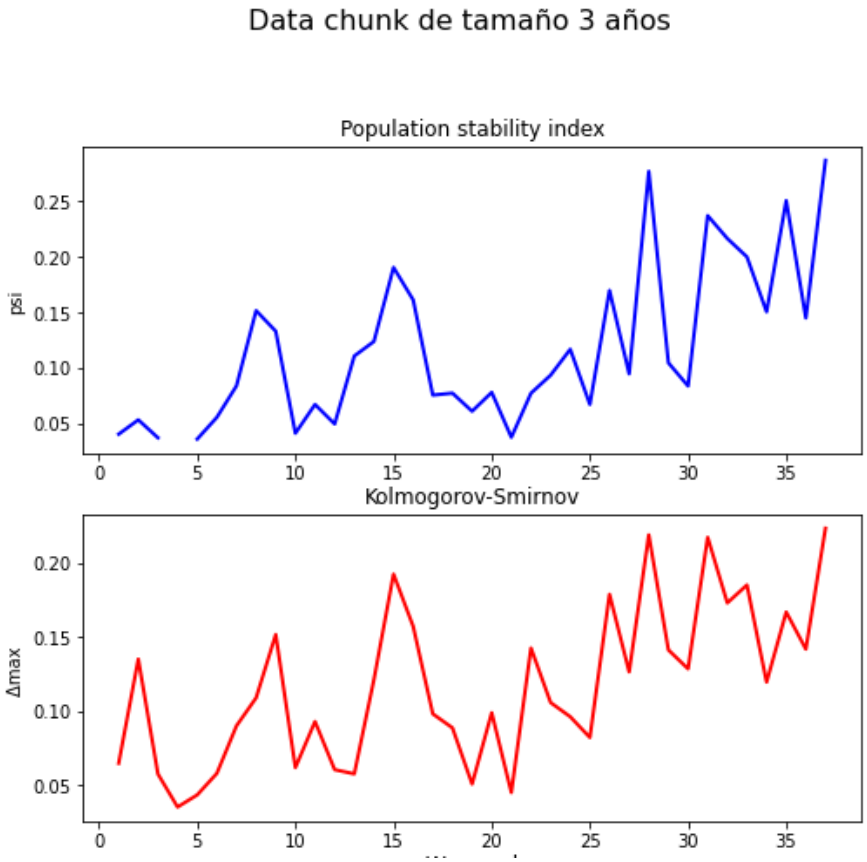

Figura 31: Valor de los estadísticos PSI y K-S para los diferentes *data chunks*

Como se puede observar, el valor de PSI sufre una discontinuidad en el cuarto *data chunk*. Esto es algo que se puede evitar fácilmente ampliando la cantidad de datos con la que cuenta cada uno de los *data chunks*.

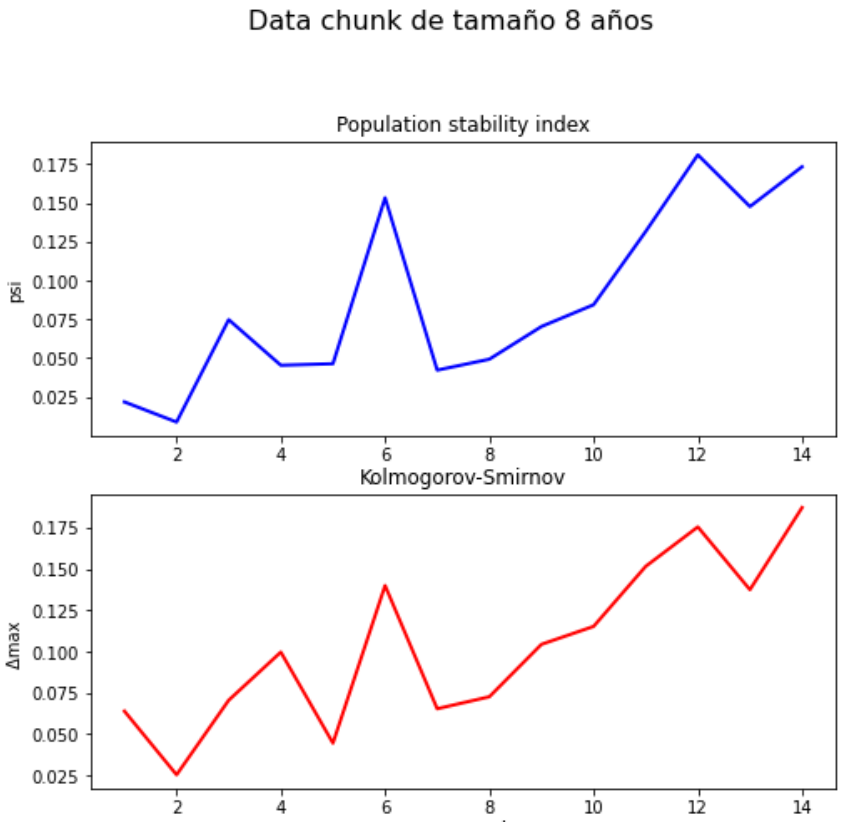

Figura 32: Valor de los estadísticos PSI y K-S para los diferentes *data chunks*

En este caso, 32 la gráfica de los valores PSI no sufre de ninguna discontinuidad. Por lo tanto esta podría ser una posible solución, sin embargo no se deberían tener problemas haciendo uso de un tamaño de un *data chunk* pequeño, siempre y cuando cada *data chunk* contenga un número adecuado de observaciones. Esto es, en este conjunto de datos, un año puede tener tan solo registrada una temperatura al día mientras que en otro conjunto de datos en un día se registra la temperatura 100 veces. Esta es la razón por la que los datos deben separarse basándose en el número de observaciones. En la figura se observa también como el número de *data chunks* analizados ha disminuido. Esto se debe a que la misma muestra de datos se divide en partes de mas grandes o mas pequeñas dependiendo del caso.

Computacionalmente hablando, el PSI es más rápido y consume menos recursos, sin embargo se debe apoyar en otro test estadístico para los casos en los que no se reúnan las condiciones suficientes para poder hacer uso del PSI. Es por esta razón por la que para detectar el*data drift* en el plano univariante cuantitativo se opta por no usar el PSI. El test de K-S es mas lento pero asegura que siempre devolverá

un valor razonable.

Cuando se deba realizar una detección en variables de tipo categórico se hará uso del test Chi cuadrado. Este mismo también se utilizará cuando la variable sea binaria. La comparación de hipótesis en relación a las proporciones podría ser incluida en el *framework* para su implementación en variables binarias, sin embargo el uso del test chi cuadrado es valído en este tipo de variables ya que al fin y al cabo una variable binaria es una variable categórica de tan solo dos categorías. De esta forma, sin perder eficacia en la detección, se consigue reducir la complejidad del *framework*

De forma paralela al análisis unidimensional, se realiza una detección a la distribución completa de todos los datos de entrada al modelo. Esta distribución de datos multidimiensional puede contener variables categóricas y cuantitativas en el mismo *dataframe*.

Se hará uso del *Multivariate generalizations of the Wald-Wolfowitz runs test* para aquellas distribuciones multivariantes de cualquier dimensión que solo contengan variables numericas en su interior.

Para aquellos casos en los que los datos de entrada al modelo cuenten con variables de tipo categórico junto con variables de tipo cuantitativo, el test estadístico utilizado en este caso será la reconstrucción del error por PCA.

4.2. Diseño

Los datos manejados necesitan ser manipulados en algún entorno de software. En el caso de este proyecto de fin de grado se ha optado por elaborar un entorno de detección de *data drift* en python.

Las librerías principales de python que son necesarias para la creación del *framework* así como para poder realizar el manejo y análisis sobre los datos son: numpy, pandas, matplotlib, NannyML, scipy y sklearn.

El *framework* utilizará 3 inputs por parte del usuario. Por un lado se debe cargar el archivo con los datos de referencia y otro con los datos de análisis. Los datos de referencia son considerados los datos sin *data drift*, que son los que se utilizaron para entrenar el modelo, o parte de este conjunto. Por otro lado, el conjunto de datos de análisis es dividido en diferentes *data chunks* los cuales se van comparando uno a uno con el conjunto de referencia.

Dependiendo de los tipos de datos que utilice el modelo asi como el tipo de dimensión en la que se esté realizando el análisis, se utilizará un test estadístico u otro. Finalmente la lógica del *framework* queda representada en la imagen 33.

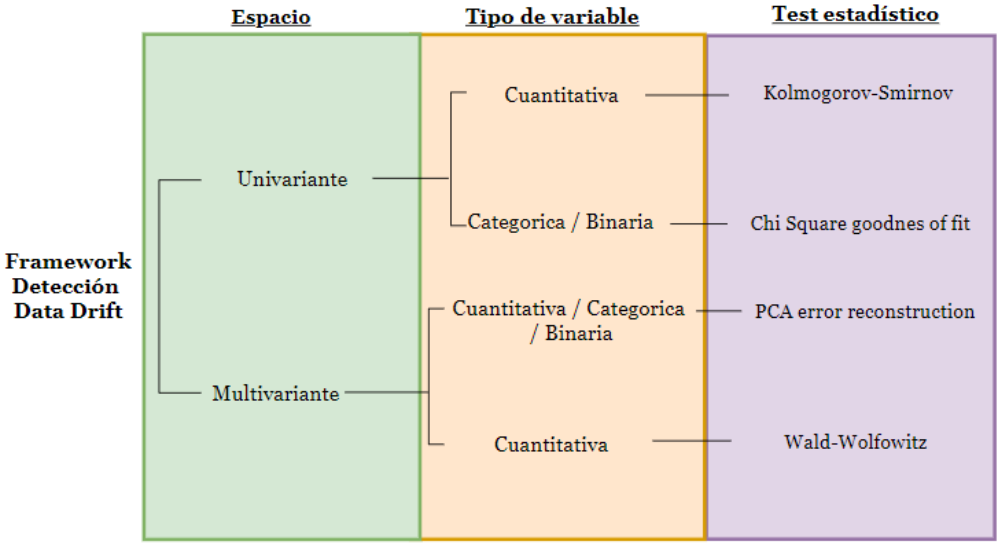

Figura 33: Diagrama de lógica del *framework*

Este proyecto se encuentra accesible para cualquier persona de forma gratuita en el siguiente repositorio:
https://github.com/CarboPablo/Framework-Data-Drift-Detection

4.3. Implementación

A continuación se muestra la utilización de este *framework* en un posible caso de uso real.

Los datos son creados de manera sintética. Resulta esencial utilizar datos creados de manera artificial para la verificación de este *framework*, ya que de esta manera es posible introducir un cambio en la distribución de los datos y analizar si el *framework* lo detecta. De esta manera se puede verificar que trabaja o no correctamente.

Es importante demostrar también que es eficaz en todos los posibles casos de uso, esto es, para los distintos tipos de dimensiones y variables en el *dataframe*.
Sobre el mismo conjunto de datos se mostrará el funcionamiento de este *framework* para diferentes casos de uso. El ejemplo consta de una serie de datos manejados por un banco.

Se cuenta con información sobre cada cliente en diferentes ámbitos. Se supone que todos las personas que están registrados en el *dataframe* han solicitado un préstamo al mismo, y por lo tanto el banco conoce de cada individuo información relativa a la cantidad de dinero que se pretende prestar por parte del banco e información sobre el patrimonio de la persona que lo solicita. Además de estos datos, y para poder demostrar el correcto uso del *framework* en variables tipo categóricas o binarias se dispone de información relativa al nivel académico de cada persona como variable tipo categórica. Finalmente se cuenta con una columna extra en la que se indica si el cliente tiene o no hijos, como variable binaria.

4.3.1. Caso de uso 1

Este apartado tiene como fin cumplir dos objetivos. El primero es demostrar de manera muy intuitiva que el *data drift* provoca una pérdida de rendimiento en un modelo de inteligencia artificial. De esta manera, se pretende justificar su implementación de forma paralela en modelos ya puestos en funcionamiento. El segundo, es demostrar la eficacia del *framework* en casos en los que no se dispone de variables de tipo categóricas ni binarias.

Para poder demostrar que un modelo pierde rendimiento, primeramente se debe entrenar un modelo de inteligencia artificial. En este caso de uso, el banco dispone

de información relativa al patrimonio de la persona que pide el prestamos así como la cantidad de dinero que esta solicita.

El banco busca entrenar un modelo de clasificación que separe a los clientes en 3 clases. La primera clase consta de las personas que pagan el préstamo y lo hacen dentro del plazo solicitado, sin retraso. Estas personas se caracterizan por tener un patrimonio elevado y un préstamo pequeño, se trata de la clase 0. La segunda clase, la clase 1, la conforman las personas que pagan el préstamo pero fuera de plazo. Se caracterizan por tener un préstamo elevado pero un gran patrimonio. La última clase, la 2, está formada por las personas que nunca llegan a devolver el dinero del préstamo al banco. Se caracterizan por deber una cantidad de dinero grande y no disponer de gran patrimonio.

Como se comenta anteriormente, estos datos son creados de manera sintética. Cada clase de personas sigue una distribución normal de 2 dimensiones. La primera dimensión corresponde con el dinero prestado y la segunda con el patrimonio.

Se entrena por tanto, un modelo de clasificación [19]. En este caso se utiliza este algoritmo que divide el espacio en franjas que delimitan la clase a la que cada punto pertenece. Como cualquier modelo de inteligencia artificial, el proceso de creación de un modelo consta una fase de entrenamiento y otra de verificación.

Para la fase de entrenamiento se utiliza un *dataframe* con información relativa a 15000 clientes. La manera en la que el árbol de clasificación aprende a separar el espacio bidimensional en franjas se observa en la figura 34.

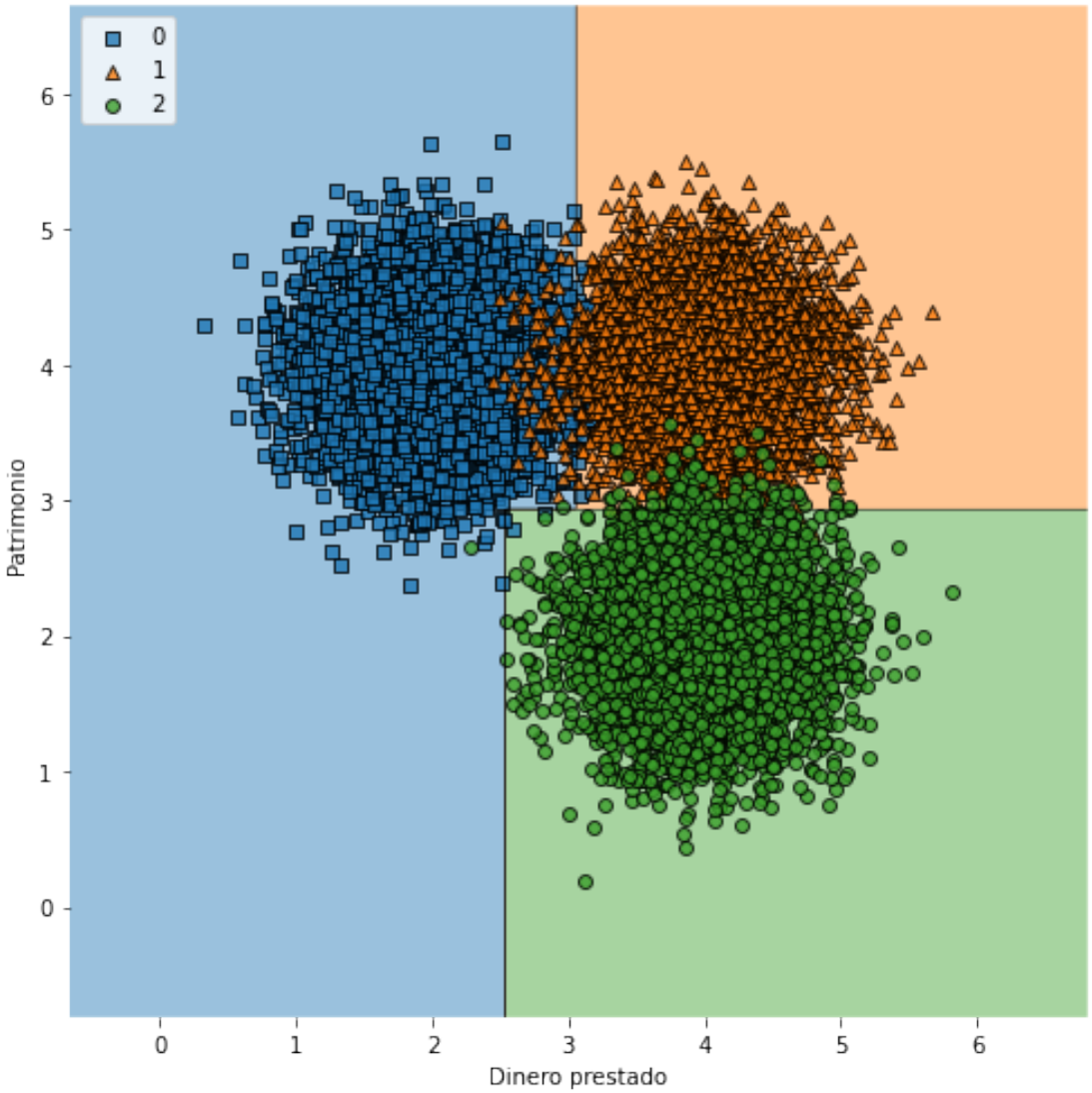

Figura 34: Caso 1: Árbol de clasificación datos entrenamiento

Una vez el algoritmo ha aprendido las franjas que delimitan el espacio, se testea el rendimiento de este modelo con un *dataframe* de tan solo 1500 datos y se observa que tan bien clasifica información nunca antes vista. Los datos nunca vistos por el modelo se pueden ver en la figura 35

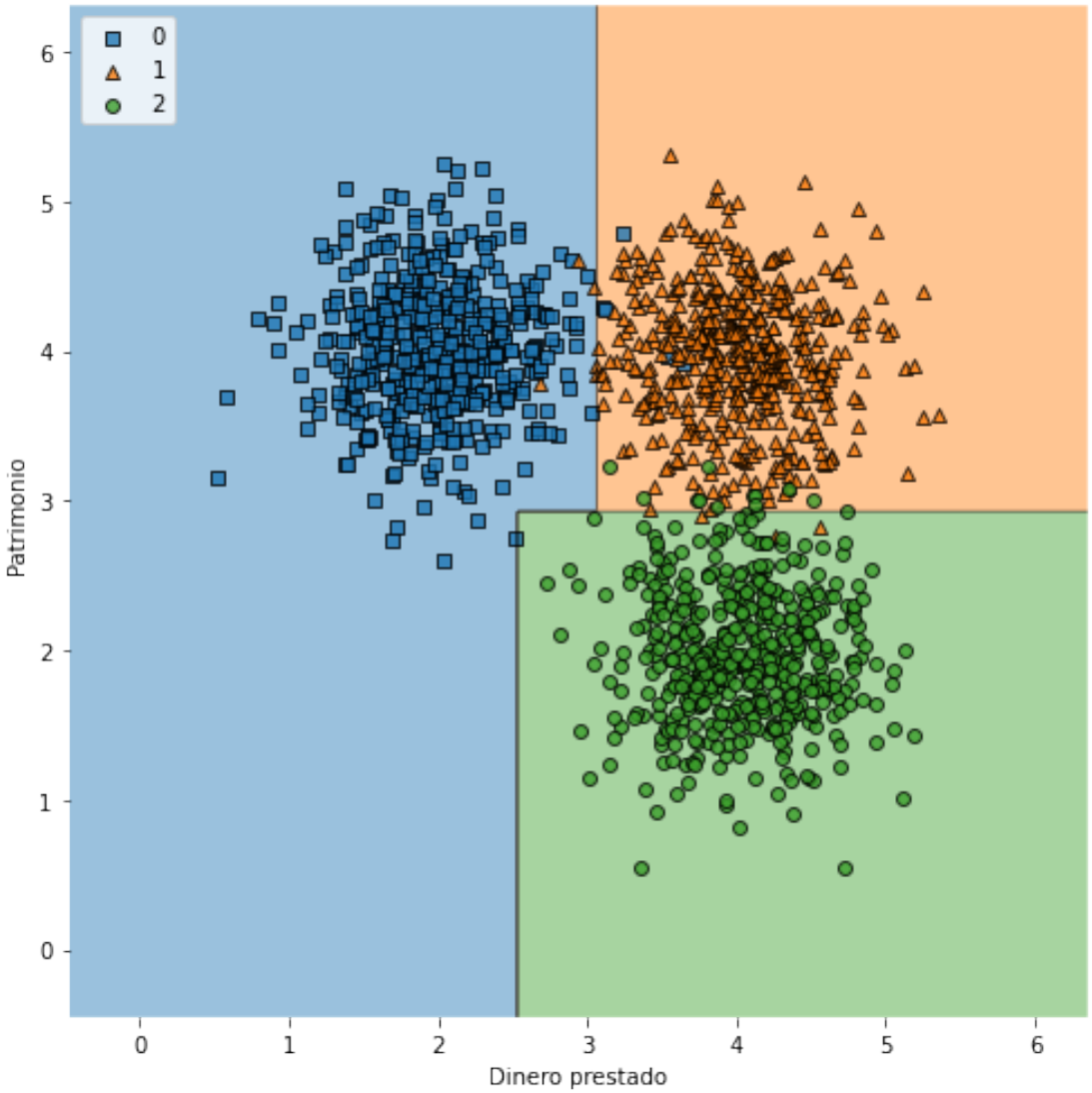

Figura 35: Caso 1: Árbol de clasificación datos nunca vistos

Se calcula el rendimiento inicial de este modelo como el porcentaje de aciertos del mismo, la matriz de confusión del árbol de clasificación contiene esta información. El rendimiento es la relación entre los datos correctamente clasificados y los datos totales.

	Clase 0	Clase 1	Clase 2
Clase 0	495	5	0
Clase 1	4	493	3
Clase 2	0	11	489

Cuadro 1: Matriz de confusión

En la traza de la matriz se encuentran los valores que el modelo ha clasificado de manera correcta. La suma de todos los datos de la matriz coincide con el número de datos totales contenidos en el *dataframe* de verificación. Es por esto que el rendimiento se calcula como la traza de la matriz entre la suma de esta. En este caso el rendimiento inicial del modelo es del 98.5 %.

Imagínese que ahora el banco implementa este modelo para clasificar los datos de los nuevos clientes. El modelo recibe *data chunks* de datos cada cierto tiempo y estos datos son a su vez clasificados.

Se simula ahora, un *data drift* en la distribución de los nuevos datos de entrada. Esto se consigue desplazando el valor de las medias de las 3 distribuciones generadas. Donde antes la media estaba situada en la posición (2,4), después de un tiempo estará en (2.2,4.2) y por tanto se consigue un desplazamiento de la distribución de los datos.

Después de un tiempo, desde que el modelo fue puesto en marcha, se observa como los datos ya no están siendo clasificados de la manera mas óptima posible 36. Con el paso del tiempo las medias de las distribuciones han cambiado y el modelo está perdiendo rendimiento.

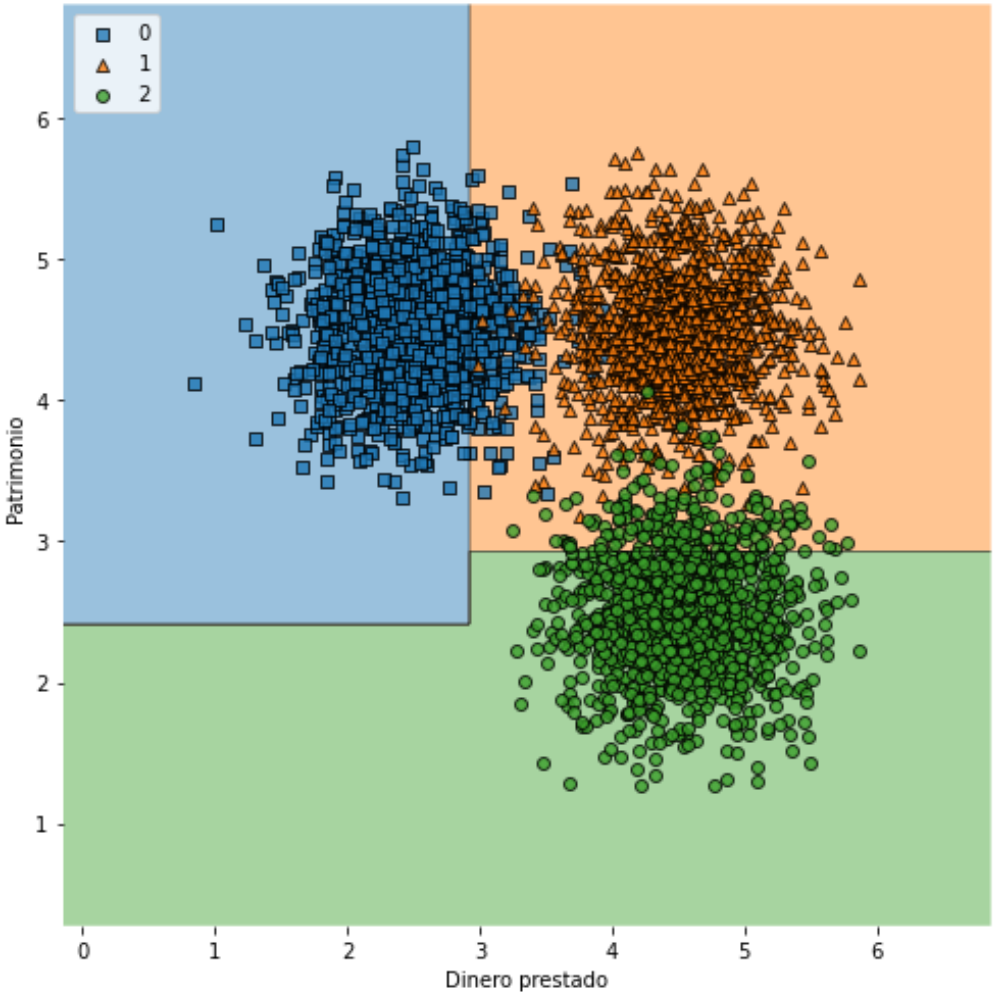

Figura 36: Caso 1: Datos de entrada con drift

La figura 36 representa el *data chunk* número 100 que ha entrado al modelo. Como se puede observar, en este caso el rendimiento será mucho menor que cuando se entrenó el modelo debido a que la distribución de entrada ya no se corresponde con la que el modelo estaba acostumbrada a predecir.

Como el *data drift* está simulado como un desplazamiento en las medias de las distribuciones, se entiende que el rendimiento va disminuyendo cada vez más. La gráfica 37 muestra el valor del rendimiento para cada *data chunk*.

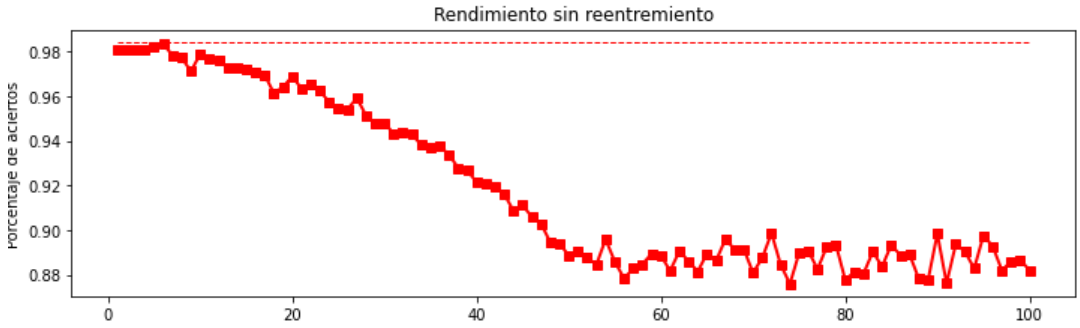

Figura 37: Caso 1: Rendimiento del modelo para los distintos *data chunks*

La línea discontinua representa el valor del rendimiento inicial. La razón por la que el rendimiento cae para después estabilizarse en un valor más o menos concreto reside en cómo se introduce el *datadrift*. En este caso solo existe *data drift* creciente en los primeros 50 *data chunks*, después el desplazamiento en la media cesa, sin embargo, no se llega a recuperar la distribución inicial para la cual el modelo fue entrenada.

Se confirma a través de este ejemplo que la existencia del *data drift* puede ser la causa de una pérdida de rendimiento en un modelo.

El valor de la variación de la media para cada *data chunk* se puede observar en la siguiente grafica 38. Se llega a un máximo de 0.2 de variación total. Se quiere ahora ver, a través de los test estadísticos, como de precisos son estos test a la hora de detectar hasta el mas mínimo cambio.

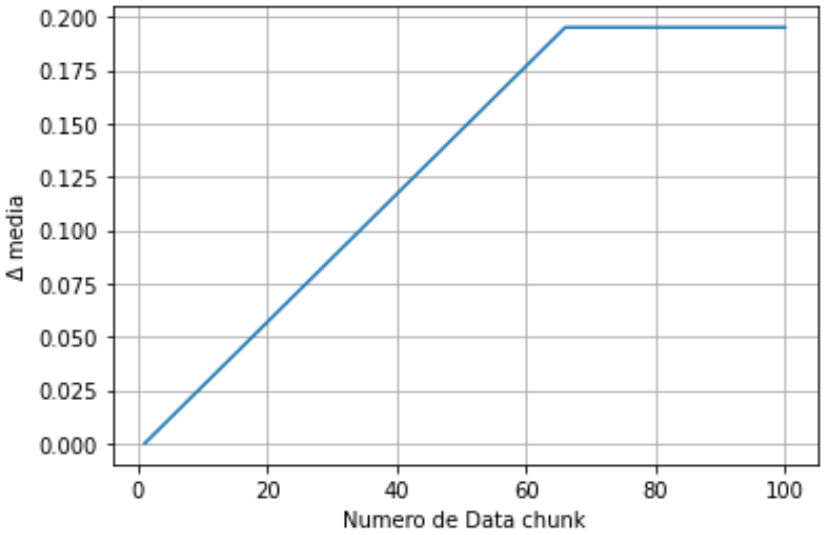

Figura 38: Valor del desplazamiento de la media de las distribuciones

Se pretende ahora, haciendo uso del *framework* diseñado, analizar y evaluar el *drift* que puede sufrir cada paquete de datos de entrada al modelo. En este ejemplo, se debe tener en cuenta que todas las variables de entrada al modelo son de tipo cuantitativas. Por tanto, haciendo uso del razonamiento que sigue el *framework* visto en la figura 33 se hará uso del estadístico K-S para la detección de *data drift* unidimensional y del estadístico de *Wald Wolfowitz* para el análisis multidimensional.

De esta forma, de la misma manera que se analizó el rendimiento para cada paquete de datos, se analiza el valor de los estadísticos 39

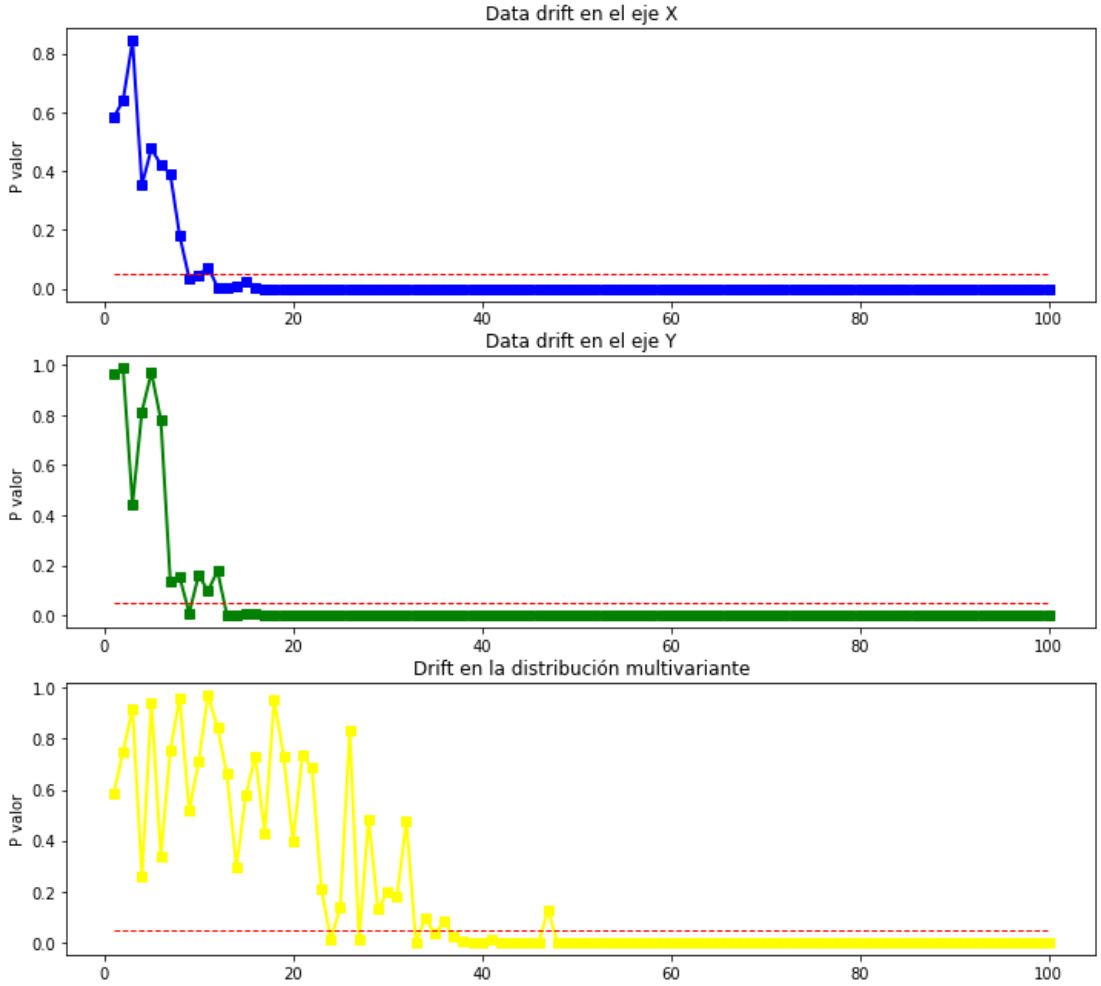

Figura 39: Caso 1: Evaluación *data drift*

En azul y en verde se muestran las gráficas que representan el p valor del test K-S para los distintos *data chunks* del conjunto de datos de análisis. Por otro lado, en amarillo se presentan estos valores para la distribución multivariante. Cuando este valor cae por debajo de 0.05 en este caso, no se puede confirmar que las 2 muestras procedan de la misma población y por tanto se considera, a ojos del usuario del *framework* que los datos de entrada están sufriendo un *data drift*.

Se pueden comparar las gráficas 39 y 38 para observar que antes del *data chunk* número 20, los test estadísticos ya han detectado la presencia de un *drift*. En este momento, las medias de las distribuciones tan solo han sufrido un 0.05 de variación en su media, sin embargo ya se ha dictaminado con un nivel de significación del 5 % que esas dos muestras no provienen de la misma población.

Se puede demostrar a través de este caso que un *datadrift* puede provocar una pérdida de rendimiento en modelos ya puestos en funcionamiento. La solución para este problema reside en el re-entrenamiento. Esto es puesto a prueba.

El objetivo final de este proyecto de fin de grado es construir un *framework* que pueda servir de apoyo a la hora de tomar la decisión de reentrenar un modelo. Por esta razón se procede a reentrenar en base a la información que se obtiene. La manera de realizar el reentrenamiento es decisión del usuario y no hay una sola manera de hacerlo. Para demostrar que un reentrenamiento puede reducir drásticamente este problema, en este caso se opta por reentrenar el modelo cada vez que, en la distribución multivariante, el p valor sea menor que 0.05 3 veces. Se evita reentrenar cada vez que baje de 0.05 para evitar el abusivo reentrenamiento.

Si se hace esto, se puede observar como, con un reentrenamiento sencillo, se puede conseguir disminuir la pérdida de rendimiento 40.

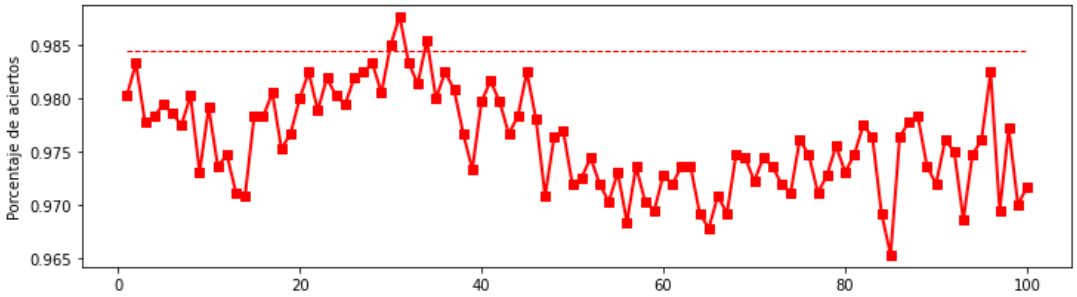

Figura 40: Rendimiento del modelo con reentrenamiento

Se justifica de esta forma que el *framework* puede servir de apoyo para la toma de decisión de reentreno. Además se demuestra que la existencia del *data drift* puede causar una pérdida de rendimiento en el mismo.

4.3.2. Caso de uso 2

En este caso, se pretende demostrar la eficacia del *framework* cuando no solo se tienen datos cuantitativos. Para conseguir esto se amplia el *dataframe* de datos en una entrada de tipo categórica y binaria. El modelo cuenta en este caso con información sobre el dinero que se le presta al cliente, el patrimonio del que este dispone, la información sobre su nivel de estudios y si tienen hijos o no. La cabecera del *dataframe* tiene el siguiente aspecto: 41.

Dinero prestado	Patrimonio inicial	Hijos	Nivel de estudios
1.661426	2.362934	No	Master
1.833376	2.483333	No	Doctorado
1.414204	2.175130	Si	Master
2.198163	2.530174	No	Grado medio
2.928536	2.538519	No	Universidad

Figura 41: Caso 2: Datos de entrada

Se evalúa primeramente de manera univariante cada categoría para después evaluar todos los datos de manera multivariante. Como el objetivo es explorar las posibilidades del *framework* al completo, en el plano unidimensional, el análisis del *data drift* para variables categóricas se realizará haciendo uso del test Chi cuadrado.

La distribución completa multidimensional de los datos incluye variables categóricas junto con numéricas. Es por esta razón que la evaluación multidimensional de los datos se hará a través de la reconstrucción del error por PCA.

Primeramente se evalúa si el *framework* funciona en casos en los que no hay *drift*. Para ello se utiliza la misma población para generar los datos de referencia y los datos analizados. En la figura 42 se puede apreciar cómo, aunque es muy poco habitual, en ocasiones los test estadísticos indican la existencia de data drift en alguno de los *data chunks* analizados. Esto es debido a la gran sensibilidad de los mismos ya que de un *data chunk* a otro pueden percibirse pequeños cambios que los test consideren alarmantes.

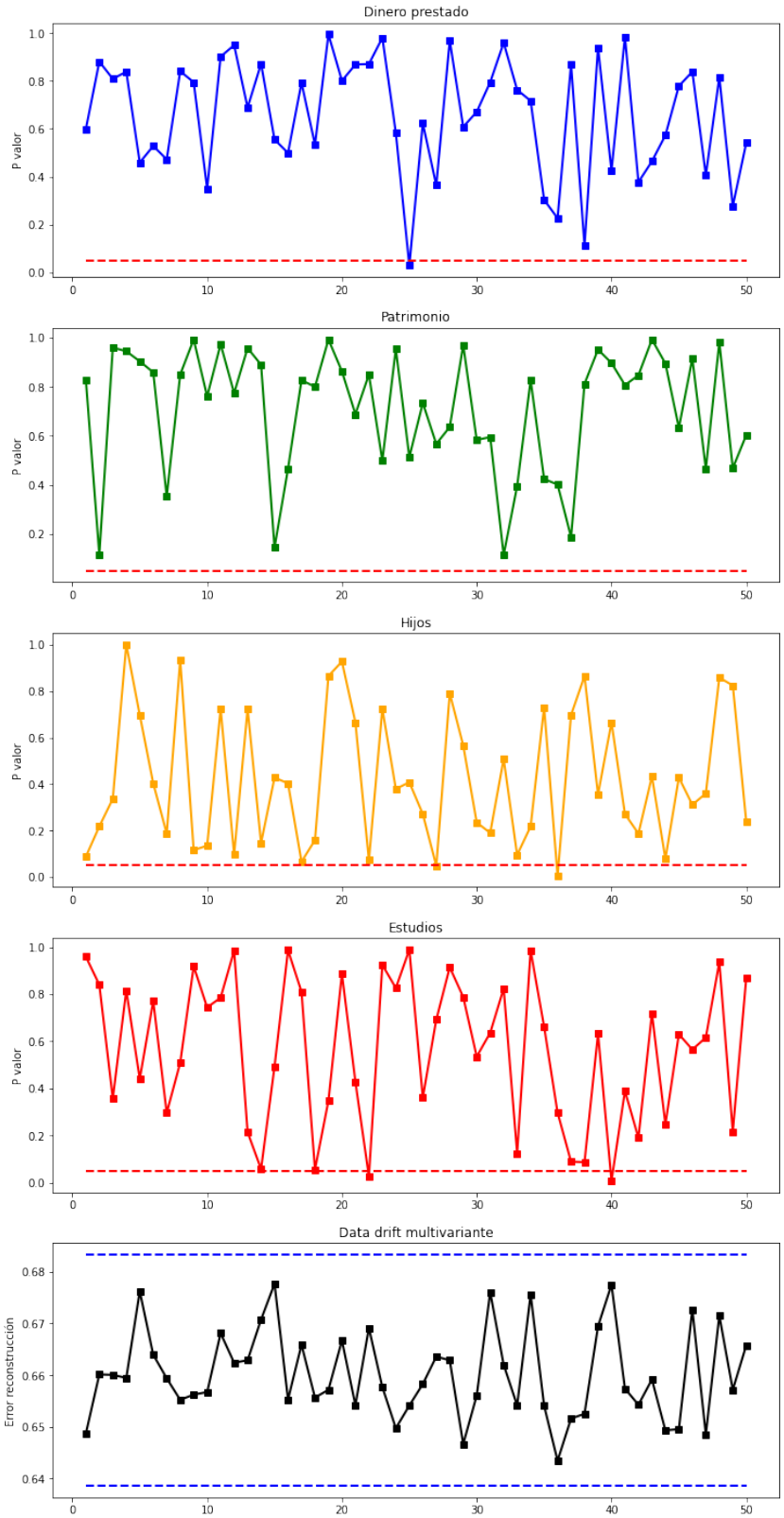

Figura 42: Caso 2: Datos de entrada

En la figura se pueden apreciar unas lineas discontinuas. En el caso de los test estadísticos univariantes, estas lineas representan el valor del p valor 0.05. Si la gráfica cae por debajo de esta linea, se generará una alerta en el *framework*. Por otro lado, en el análisis multivariante no se tiene una línea si no que se tiene una horquilla de valores, entre los cuales se encuentran los errores aceptables. Un error por encima o por debajo de esa horquilla significará la creación automática de una alarma en el *framework* de detección de *data drift* multivariante.

El *framework* creado cuenta con la creación automática de un informe de alertas. Se considera alerta cada vez que el valor de uno de los test se considera fuera de lo esperado.

En este caso, el informe generado es el mostrado en figura 43. A partir de este análisis previo, podemos imaginar que la detección puntual de alarmas por nuestro framework no debería implicar necesariamente un reentranimiento de los modelos, sino simplmente una inspección o una alerta para el equipo de análisis. Se verá posteriormente que es sencillo identificar una necesidad de reentrenamiento, o en definitiva un verdadero textitdata drift cuando las alarmas ocurren de manera constante en el tiempo.

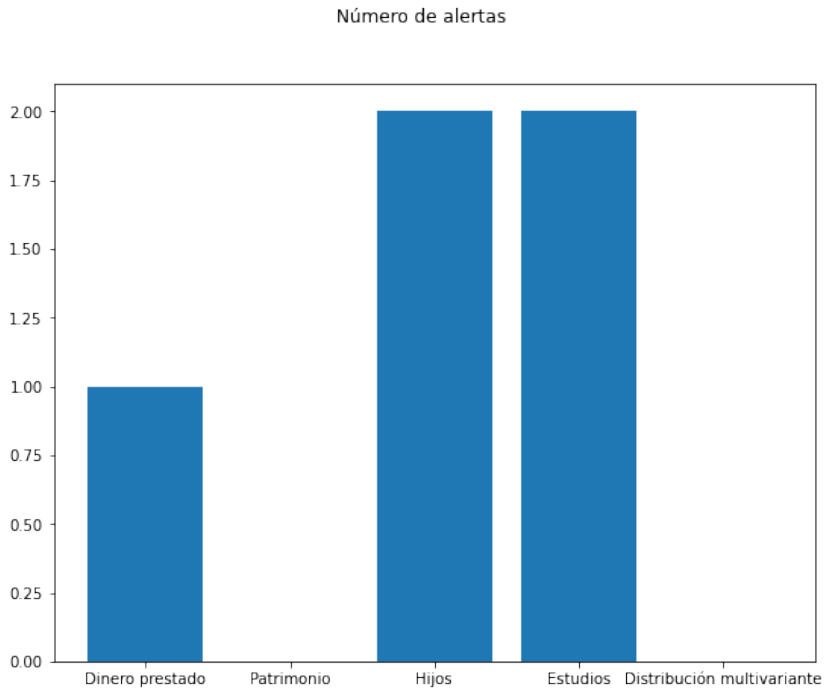

Figura 43: Caso 2: Informe de alertas

Teóricamente, la variación en cualquiera de las distribuciones univariantes se debería ver reflejada en un cambio en la distribución multivariante. Resulta intuitivo por tanto basar la decisión de re-entrenar el modelo, unicamente en las alertas generadas por la distribución multivariante, sin embargo eso sería un error.

El informe de alertas informa del estado del *data drift* tanto univariante como multivariante y esto es así por un motivo. Como regla general, si se detecta un data drift en la distribución multidimensional, se puede confiar en que al menos una de las distribuciones de los datos de entrada ha sufrido *drif*. Sin embargo, que no se genere ningún aviso en el informe de alertas en la distribución multivariante no significa que una de las variables haya podido sufrir drift.

Primeramente se procede a analizar un caso en el que, una variación en la distribución de una de las variables se detecta tanto univariantemente como multivariantemente.

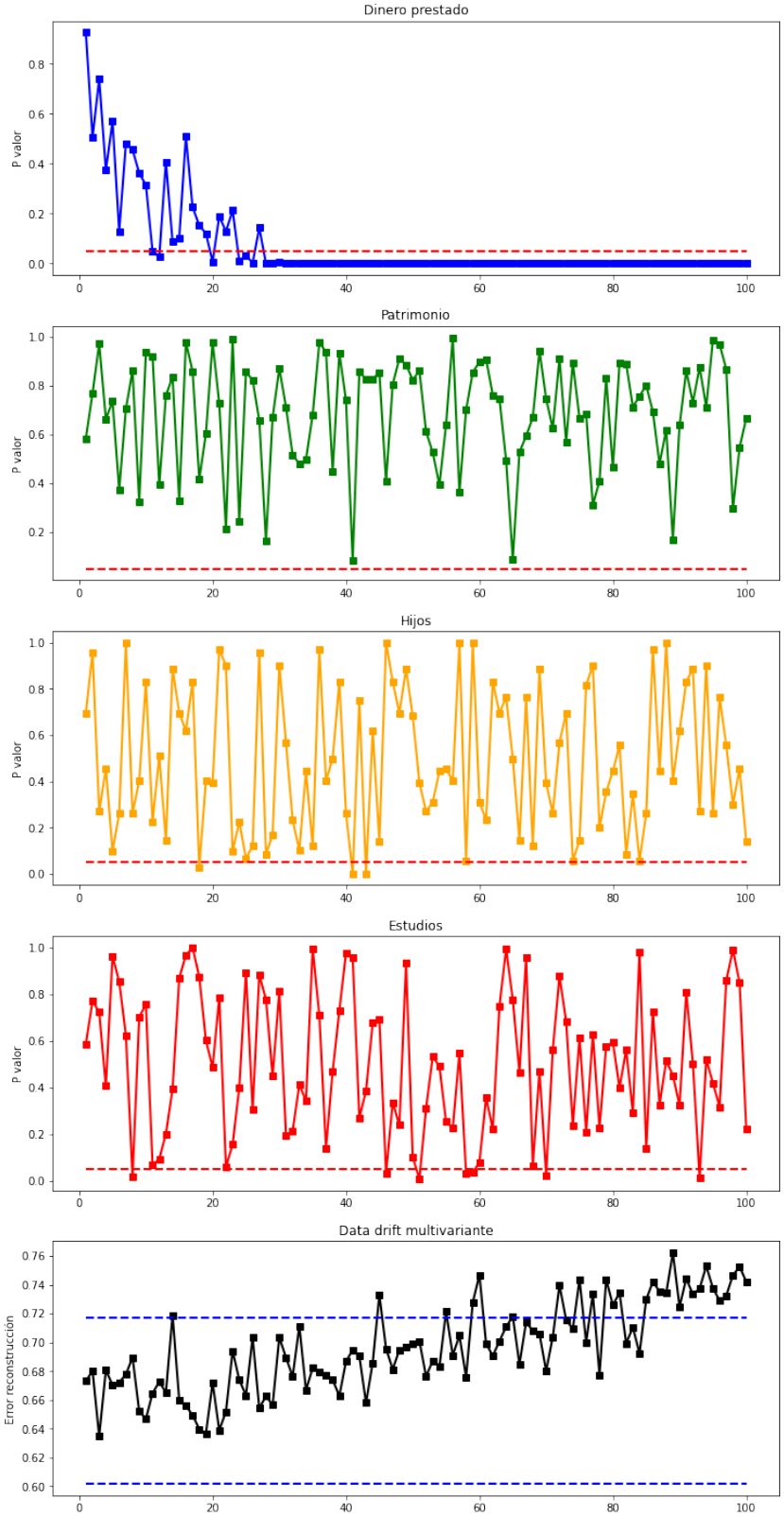

Figura 44: Caso 2: *Data drift* en una única variable de entrada

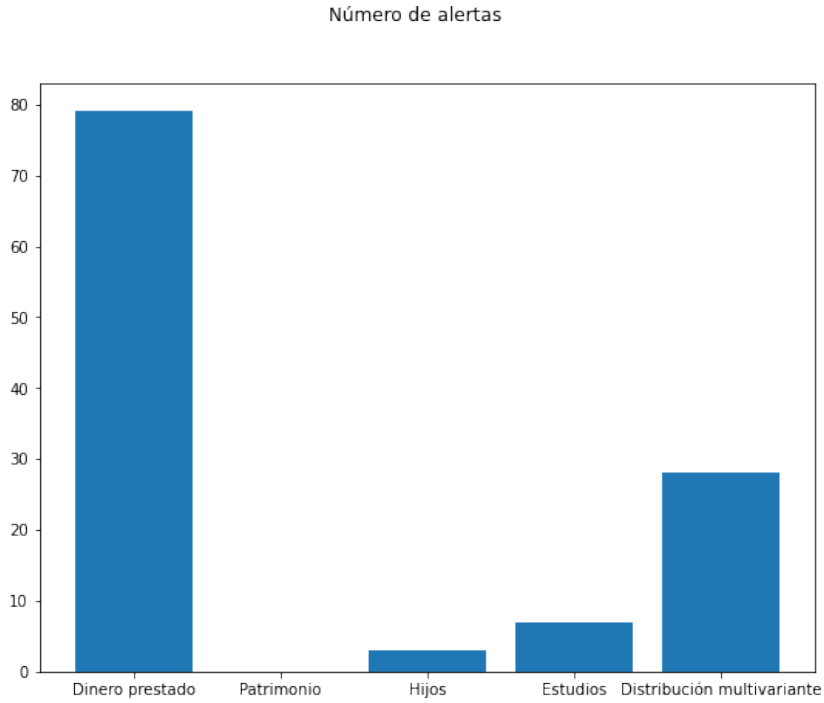

Figura 45: Caso 2: Informe de alertas, drift en una única variable de entrada

Como se puede observar tanto en la figura 44 como en el informe de alertas 45 la variación en la distribución se detecta en las 2 distribuciones. Cabe destacar que hay un retardo desde que se detecta el drift en la variable hasta que se denoda un cambio alarmante en la multivariante, sin embargo la explicación de esto reside en la manera en la que el *drift* está simulado. El *drift* ocurre de menos a más y en este caso exactamente cambia muy lentamente de un *data chunk* al siguiente.

Sin embargo, esto no es una regla general. Como ya se ha comentado, existe la posibilidad de que se produzca un *drift* en la distribución de alguna de las variables, y sin embargo, esto no se vea reflejado en la detección de *data drift* univariante. Esto ocurre porque, como se comenta en el capitulo 3, la detección del *data drift* univariante se basa en la reconstrucción del error por PCA. Utilizar el algoritmo PCA implica que hay variables que tienen más peso que otras, y por tanto, cuando existe una variación en alguna de estas variables, estas generarán más impacto a la hora de reconstruir los datos, que las variables con menos peso.

Para mostrar este fenómeno con un ejemplo, se propone introducir un *data drift* en la distribución que sigue la variable Hijos. En el *dataframe* de referencia, el 50 % de las personas tienen hijos y el otro 50 % no. En los datos de entrada al modelo,

se simula un *data drift* un tanto extremo, para observar este fenómeno. En este caso, ninguna de las personas que aparecen en el *dataframe* de análisis cuentan con hijos, por lo tanto se puede esperar que los test estadísticos, al menos de manera univariante, detecten un cambio.

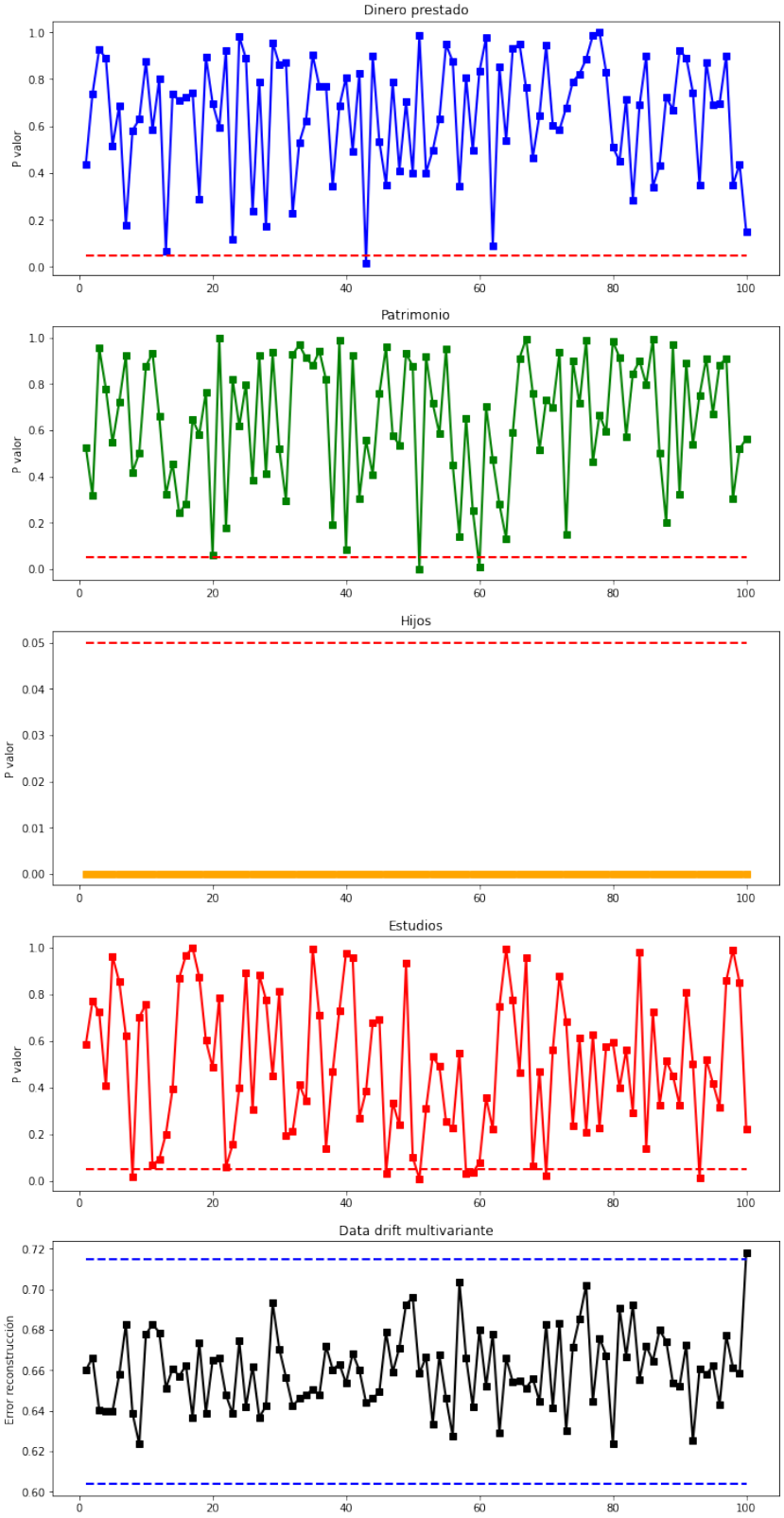

Figura 46: Caso 2: *Data drift* en la variable Hijos con *dataframe* completo

Como se puede observar en el gráfico 46, el cambio en la distribución en la variable hijos es detectado de manera perfecta por el test estadístico del Chi cuadrado, sin embargo, en la detección de *drift* multivariante no parecen saltar las alarmas. Es por esta razón por la que se debe realizar una detección del *drift* tanto univariante como multivariante.

Sin embargo, este fenómeno no solo le ocurre a la variable Hijos. Se puede observar que si se introduce un *drift* en la variable Estudios, se obtiene un resultado similar 47

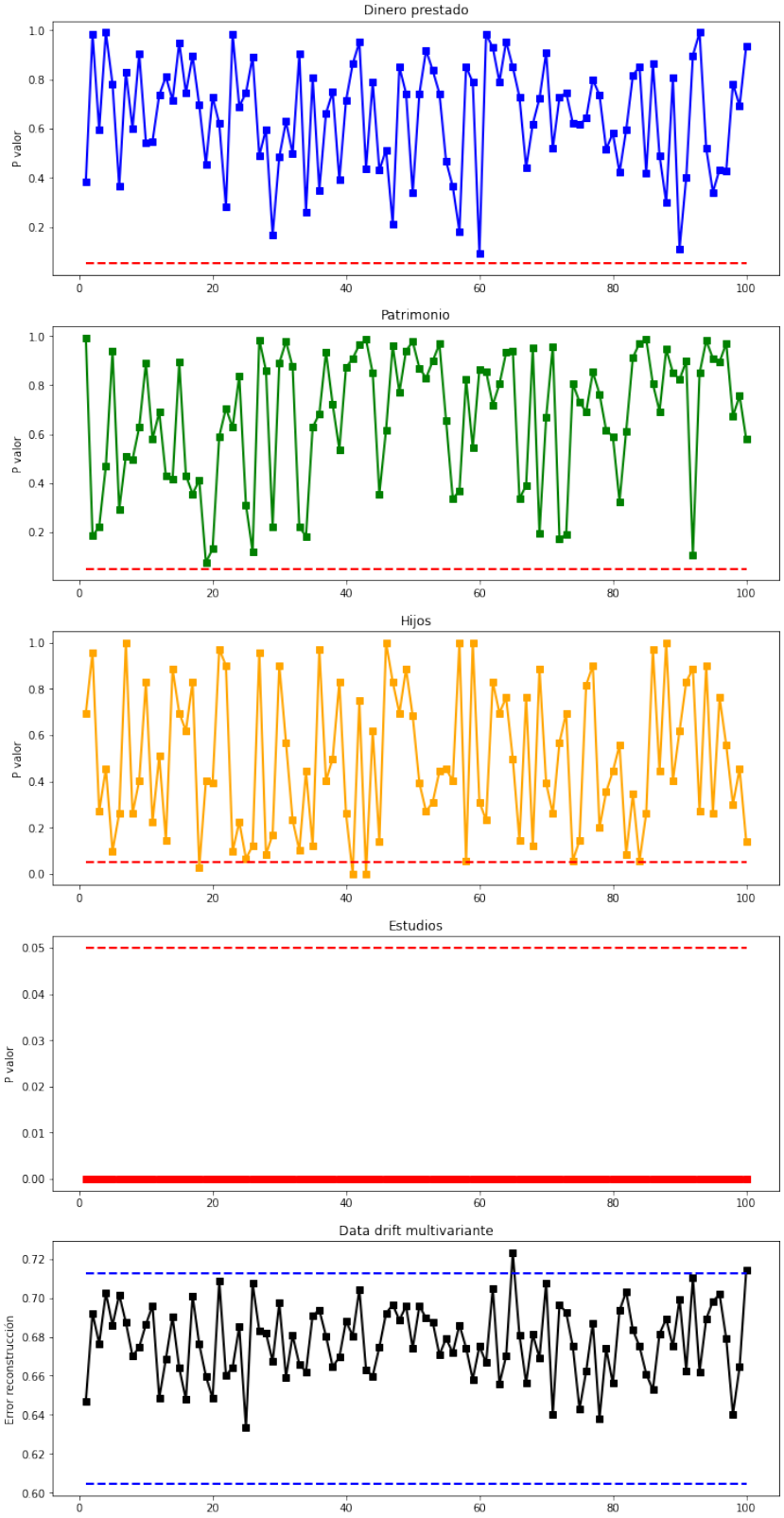

Figura 47: Caso 2: *Data drift* en variable estudios con *dataframe* completo

Que en este caso de uso, las datos de entrada relativos al patrimonio y el dinero que el banco presta, tengan más peso en el algoritmo PCA, no significa que este método no funcione para detectar *drift* en variables categóricas.

Si eliminamos estas dos variables de entrada con más peso de nuestro *dataframe*, se puede observar como, ahora, la variable categórica estudios tiene más peso en la reconstrucción del error por PCA, y si se introduce un cambio en la distribución de esta variable, este cambio será detectado tanto de manera univariante como multivariante.

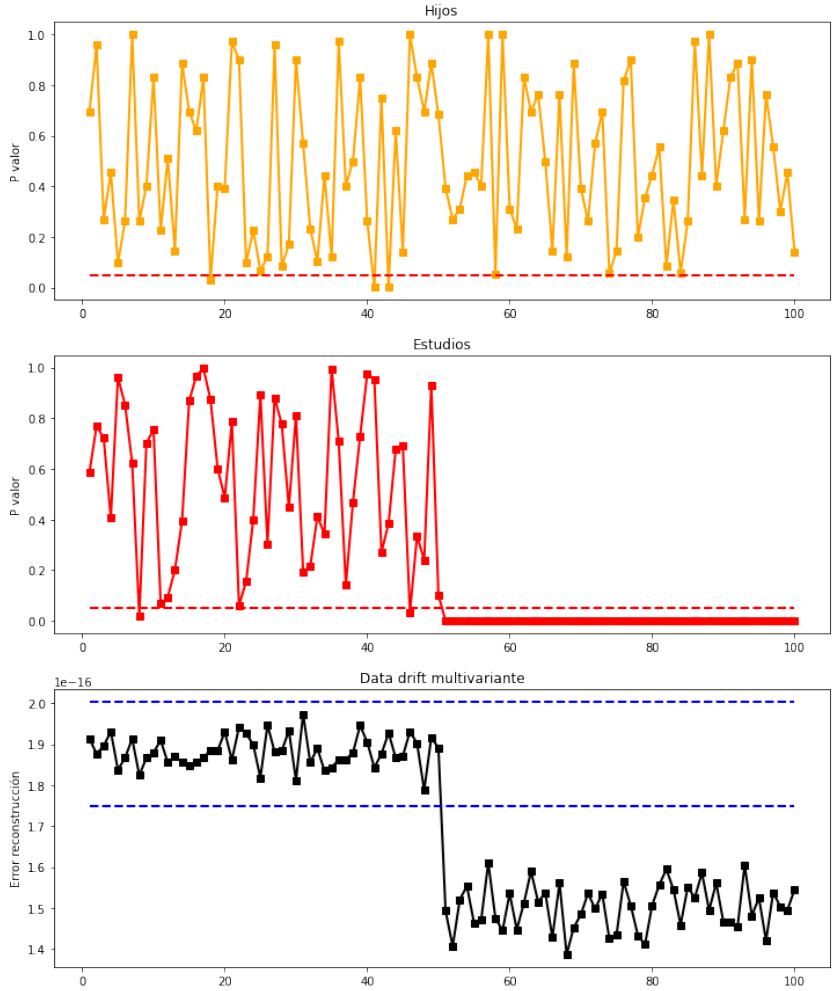

Figura 48: Caso 2: *Data drift* en variable estudios, con *dataframe* simplificado

5. CONCLUSIONES Y TRABAJOS FUTUROS

Este proyecto se ha centrado en un fenómeno denominado *data drift* que resulta un problema para los modelos de inteligencia artificial. Se ha puesto el foco en detectar este problema a través del análisis de datos sintéticos viendo como podría afectar a un modelo real. Para hacer esto se ha pintado la evaluación de los datos a través de los diferentes paquetes de entrada.

Se concluye después de hacer uso del *framework* como herramienta de apoyo para el reentreno del modelo, que este mismo sirve como freno de perdida de rendimiento y funciona relativamente bien. En la memoria queda demostrado como ocurre esto, con un ejemplo aplicado a un modelo de clasificación.

Los objetivos cubiertos han sido, desarrollar un software que analice los datos de entrada tanto de manera univariante como multivariante. El *framework* puede trabajar en entornos que contengan tanto variables cuantitativas como categóricas y binarias. Este sistema cuenta con que el tamaño del *dataframe* puede ser modificado, y de esta forma poder analizar el *data drift* con más o menos cadencia. Se ha conseguido también desarrollar un *framework* que cuente con una lógica que permita utilizar unos test estadísticos u otros dependiendo de la situación. Por último y puesto que todo el proyecto está colgado en GitHub [20] todo el código desarrollado puede ser utilizado por cualquier usuario de manera completamente gratuita.

El valor de este proyecto reside en la especialización de la tarea. No existe una tecnología hoy en día que se centre únicamente en la monitorización y detección de este fenómeno. Tampoco hay ninguna tecnología que cumpla con todos los requisitos impuestos por este proyecto para conseguir un buen detector. Es por esto, que este *framework* desarrollado aporta un valor a la comunidad.

Como propuesta de mejora futura, se trabajaría en mejorar la simplicidad a la hora de tomar la decisión de reentreno a un único parámetro. De esta forma el usuario no necesitaría de ver gráficas ni histogramas, toda la información estaría basada en un index en el que este se apoyaría. Por otro lado, del *framework* se podría mejorar el código en python, reduciendo el número de funciones que ser cargan cada vez a el uso de una única librería. De esta forma el usuario podría reducir el tiempo de carga de cada análisis.

6. ANEXOS

6.1. Enlace a repositorio GitHub

Acceso al repositorio de GitHUb donde se encuentra disponible el *framework* desarrollado:
https://github.com/CarboPablo/Framework-Data-Drift-Detection

6.2. Alineación del proyecto con los objetivos de desarrollo sostenible

Este proyecto de fin de grado se encuentra alineado con alguno de los objetivos de desarrollo sostenible.

Se ha puesto al alcance de cualquiera, de manera *open source*, todo el estudio realizado, desde el estado del arte actual en tecnologías y técnicas utilizadas, hasta el framework de trabajo completo de detección de data drift. Por este motivo, el objetivo de desarrollo que busca una educación de calidad se alinea con este proyecto de fin de grado.

Por otro lado, uno de los beneficios del framework a desarrolla es el de parar la caída de rendimiento de un modelo de *machine learning* y por tanto dar pie a ser capaces de conseguir mejores resultados desembocando en conseguir trabajo creciente y crecimiento económico. Al mejorar la manera en la que los modelos de *Machine Learning* operan, y como estos están implicados en amplios campos de la economía y el trabajo, mejorar este aspecto es directamente proporcional a mejorar y hacer crecer la economía y por tanto el trabajo.

Una de las grandes ventajas del framework que se ha diseñado es la de reducir el número de veces que se re-entrena un modelo. Este proceso requiere de unos recursos que son potencialmente reducibles. Por lo tanto este proyecto fin de grado, al reducir la frecuencia con la que se hace uso de estos recursos, se alinea con el objetivo de conseguir ciudades y comunidades sostenibles. La reducción del uso de las herramientas necesarias para el re-entreno de un modelo también tiene un efecto positivo al medio ambiente, ya que se reducen las emisiones que colateralmente estos medios implican.

Referencias

[1] Batta Mahesh. Machine learning algorithms-a review. *International Journal of Science and Research (IJSR).[Internet]*, 9:381–386, 2020.

[2] Tom Dietterich. Overfitting and undercomputing in machine learning. *ACM computing surveys (CSUR)*, 27(3):326–327, 1995.

[3] Google. reCAPTCHA sistema control de bots. Último acceso 10/4/2022. `https://www.google.com/recaptcha`, 2016.

[4] Jeffrey C Schlimmer and Richard H Granger. Incremental learning from noisy data. *Machine learning*, 1(3):317–354, 1986.

[5] David Sculley, Gary Holt, Daniel Golovin, Eugene Davydov, Todd Phillips, Dietmar Ebner, Vinay Chaudhary, Michael Young, Jean-Francois Crespo, and Dan Dennison. Hidden technical debt in machine learning systems. *Advances in neural information processing systems*, 28, 2015.

[6] Tensor Flow plataforma de extremo a extremo para aprendizaje automático. Último acceso 10/4/2022. `https://www.tensorflow.org/tfx/data_validation/get_started#checking_data_skew_and_drift`, 2017.

[7] Vance W Berger and YanYan Zhou. Kolmogorov–smirnov test: Overview. *Wiley statsref: Statistics reference online*, 2014.

[8] Bilal Yurdakul. *Statistical properties of population stability index*. Western Michigan University, 2018.

[9] MG Habib and DR Thomas. Chi-square goodness-if-fit tests. *The Annals of Statistics*, pages 759–765, 1986.

[10] Rasmus Bro and Age K Smilde. Principal component analysis. *Analytical methods*, 6(9):2812–2831, 2014.

[11] Julián Luengo, Salvador García, and Francisco Herrera. On the choice of the best imputation methods for missing values considering three groups of classification methods. *Knowledge and information systems*, 32(1):77–108, 2012.

[12] Raji Nair, Etienne Perret, and Smail Tedjini. Temporal multi-frequency encoding technique for chipless rfid applications. In *2012 IEEE/MTT-S International Microwave Symposium Digest*, pages 1–3. IEEE, 2012.

[13] Jerome H Friedman and Lawrence C Rafsky. Multivariate generalizations of the wald-wolfowitz and smirnov two-sample tests. *The Annals of Statistics*, pages 697–717, 1979.

[14] Uthpala Ekanayake. The minimal spanning tree algorithm. 2014.

[15] NannyML librería de código abierto que tiene como objetivo estimar el rendimiento de un modelo una vez este ha sido puesto en funcionamiento. `https://www.nannyml.com/`, 2020.

[16] Evidently ayuda a analizar y monitorizar los datos de entrada a un modelo de machine learning durante todo el proceso de vida de este. encaja como un bloque mas en la cadena de mlops. Último acceso 10/4/2022. `https://github.com/evidentlyai/evidently`, 2019.

[17] Azure ML servicio en la nube que permite acelerar y administrar el ciclo de vida de los proyectos de aprendizaje automáticos. *Framework* para detectar *Data drift*. `https://docs.microsoft.com/en-us/azure/machine-learning/how-to-monitor-datasets?tabs=pythona`, 2018.

[18] SS Vallender. Calculation of the wasserstein distance between probability distributions on the line. *Theory of Probability & Its Applications*, 18(4):784–786, 1974.

[19] Roger J Lewis. An introduction to classification and regression tree (cart) analysis. In *Annual meeting of the society for academic emergency medicine in San Francisco, California*, volume 14. Citeseer, 2000.

[20] GitHub github es un portal creado para alojar el código de las aplicaciones de cualquier desarrollador. Último acceso 10/4/2022. `https://github.com/`, 2008.