Universidad Zaragoza
1542

Trabajo Fin de Máster

Mecanismos para la detección y adaptación del Concept Drift

Autor

Alejandro Muro Belloso

Director

Susana Ferreiro Del Río

Ponente

Jorge Delgado Gracia

FACULTAD DE CIENCIAS
2022 - 2023

Tekniker
MEMBER OF BASQUE RESEARCH & TECHNOLOGY ALLIANCE

RESUMEN

Los algoritmos de Aprendizaje Automático/Machine Learning tradicionales procesan la información recibida asumiendo una distribución estacionaria subyacente. Por ejemplo, los modelos predictivos se entrenan con conjuntos históricos de datos en forma de (input, output) o (variables predictivas, variables objetivo), de forma que puedan ser usados para obtener predicciones sobre nuevos datos. Sin embargo, resulta habitual que estos nuevos datos lleguen en forma de flujos/*streams*, produciendo gran cantidad de información a analizar cuyo contenido es susceptible de evolucionar en el tiempo. Esto conlleva un cambio entre la distribución de los datos inicial (con los que entrenamos los modelos pertinentes) y la distribución de las nuevas instancias de datos que se reciben a lo largo del tiempo, fenómeno que se conoce como *concept drift*. Dado que puede afectar al rendimiento de los modelos, es de vital importancia la detección de *concept drift* y posterior adaptación para mantener la precisión requerida.

ABSTRACT

Traditional Machine Learning models assume that data is drawn from a stationary distribution. For instance, predictive models are trained using historical data given as a set of pairs (input, output) so they can be afterwards applied for predicting the output for new unseen input data. However, very often data comes in the form of streams, resulting in large volumes of data, whose content is changing and evolving over time. This results in a change between the distributions of training data seen so-far and the distribution of newly coming data, which is known as concept drift. Because it can affect our model's predictive performance it is of utmost importance to detect and adapt to concept drifts in order to maintain the accuracy and reliability of our predictions.

Índice

Capítulo 1

Introducción

Desde hace unos años y de forma cada vez más frecuente, encontramos que gran parte de la información disponible llega a nosotros en forma de flujos/*streams*[1], debido fundamentalmente a la prevalencia de la telefonía móvil y el *Internet of Things (IoT)*[2].
La técnicas de Aprendizaje Automático o Machine Learning tradicionales asumen que los procesos que generan tales datos son estacionarios, siguiendo por tanto una distribución de probabilidad fija aunque desconocida, y se crean así algoritmos y modelos predictivos con esa base. No obstante, en la mayoría de escenarios con que uno se encuentra en el mundo real tal asunción resulta incorrecta. Efectivamente, el proceso generativo es intrínsecamente no estacionario y se producen en cambios en la relación existente entre la variable que el modelo busca predecir y el resto de datos que usa para ello. Este problema se conoce como *concept drift* y puede conllevar una degradación significativa en la capacidad predictiva.

En este trabajo centramos la atención en hacer una caracterización matemática tanto del *concept* como del *concept drift*, tal y como se definen en el estado del arte referente al tema. Continuaremos el estudio analizando los métodos y algoritmos más utilizados, acompañados de su aplicación sobre ciertos conjuntos de datos de naturaleza sintética.
Los contenidos desarrollados se estructuran como sigue. En el Capítulo 2 se introduce el Aprendizaje Automático o Machine Learning y se establecen las principales definiciones que permitirán avanzar en la lectura. El Capítulo 3 introduce formalmente el *concept drift*, con su tipología y la metodología comunmente seguida para su detección y adaptación. Un ejemplo práctico del tratamiento con *concept drift* se trata para el caso de regresión en el Capítulo 4 y para clasificación en el Capítulo 5, expandiendo así todo lo expuesto en capítulos anteriores. Finalmente en el Capítulo 6 se incluyen las conclusiones obtenidas y se establecen las directivas a seguir de cara a continuar el trabajo en un futuro.

[1]Toda secuencia continua y ordenada (a través de un índice temporal o *timestamp*) de datos se conoce como flujo de datos/*data stream*. Resulta imposible controlar el orden de llegada de las diferentes instancias y dada su extensión puede ser imposible tratarlos en su totalidad; son grandes cantidades de datos que llegan con elevada frecuencia.

[2]El *Internet of Things (IoT)*[1] describe objetos físicos (o conjuntos de ellos) con diferentes sensores, capacidad de procesamiento, software y otras tecnologías que les permitan conectarse e intercambiar datos e información con otros dispositivos y sistemas a través de internet u otras redes de comunicaciones.

Capítulo 2

Machine Learning: Definiciones Previas

Podemos definir el Aprendizaje Automático o Machine Learning[2] (ML) como el subcampo de las ciencias de computación y una rama de la Inteligencia Artificial, Artificial Intelligence (AI), cuyo objetivo es desarrollar técnicas y algoritmos capaces de "aprender". Es decir, técnicas y algoritmos con la capacidad de aprovechar la información proporcionada por los datos tratados con el objetivo de mejorar el rendimiento de ciertas tareas, como puede ser la obtención de predicciones precisas.
Puesto que el éxito obtenido con los diferentes algoritmos de aprendizaje depende en gran medida de los datos empleados, el machine learning muestra una profunda relación con el campo de la analítica de datos y la estadística matemática. Es más, las técnicas empleadas se basan en el tratamiento de datos combinando conceptos fundamentales de computación con estadística, probabilidad y optimización.
Antes de proseguir resulta conveniente establecer una serie de definiciones previas:

- Instancias/ observaciones. Son cada uno los elementos individuales del conjunto de datos usados en las diferentes labores de aprendizaje, validación de modelos... Considerando los datos en forma de tabla, cada fila corresponde a una instancia individual.

- Features/ variables predictoras/ explicativas/ independientes o covariables. El conjunto de atributos o características asociados a cada instancia; en el caso tabular estarían representadas por columnas.
  Matemáticamente las representamos con el vector $\mathbf{x} \in \mathcal{X}$, siendo $\mathcal{X}$ el conjunto de todos sus posibles valores.

- Etiquetas/ labels/ variables objetivo. Son valores/clases asociados a cada instancia y, dado un modelo predictivo, es aquello que buscamos obtener/predecir. En representación tabular pueden verse como una columna (o columnas) de especial importancia; aunque en ocasiones se consideran por separado.

Generalmente trataremos con un único valor por instancia y matemáticamente quedan representadas, en una dimensión, por $y \in \mathcal{Y}$, con $\mathcal{Y}$ el conjunto de sus posibles valores.

- Hiperparámetros. Parámetros libres $\boldsymbol{\theta}$ asociados al algoritmo (en algunos casos no hay) que permiten controlar el proceso de aprendizaje. Por ejemplo, en el caso de una red neuronal, los hiperparámetros definen el número de neuronas y el tamaño de la red.
  Se distinguen del resto de parámetros en que los valores de estos últimos se derivan del ajuste sobre los datos del conjunto de entrenamiento.

- Conjunto de entrenamiento. Conjunto de instancias usado en la fase de entrenamiento del algoritmo para ajustar los diferentes parámetros, como pueden ser los coeficientes de un modelo de regresión lineal[1].

- Conjunto de validación. Este conjunto de instancias se emplea durante la fase de entrenamiento para evaluar el ajuste de los parámetros, así como para elegir los valores óptimos de los hiperparámetros. Usando las etiquetas de cada instancia, se toman aquellos valores que ofrecen los mejores resultados predictivos.

- Conjunto de prueba/ test. Conjunto de instancias utilizado para medir la calidad/rendimiento del algoritmo de aprendizaje una vez definidos los valores de los diferentes parámetros e hiperparámetros. Se comparan así las predicciones sobre cada instancia con su respectiva etiqueta.
  A diferencia del conjunto de validación no esta disponible en la fase de entrenamiento.

- Función de pérdida. Función que mide la diferencia (o pérdida) entre la predicción asociada a cada instancia y su respectiva etiqueta. Sea $\hat{\mathcal{Y}}$ el conjunto de todos los posibles valores de las predicciones, $\hat{y} \in \hat{\mathcal{Y}}$ (normalmente $\hat{\mathcal{Y}} = \mathcal{Y}$), la función de pérdida establece entonces la aplicación

$$\begin{aligned} L : \mathcal{Y} \times \hat{\mathcal{Y}} &\longrightarrow \mathbb{R}^+ \\ (y,\, \hat{y}) &\longrightarrow L(y,\, \hat{y}) \end{aligned}$$

  Notar que siempre es mayor o igual que 0 (no consideramos pérdidas negativas).
  Se muestran ejemplos de estas funciones en secciones posteriores.

- Conjunto de hipótesis. Conjunto $\mathcal{H}$ de funciones que relacionan las features $\mathbf{x}$ con las etiquetas predichas, $\hat{y}$. Identificamos así cada hipótesis con una aplicación de $\mathcal{X}$ en $\hat{\mathcal{Y}}$

$$\begin{aligned} h :\, \mathcal{X} &\longrightarrow \hat{\mathcal{Y}} \\ \mathbf{x} &\longrightarrow \hat{y} = h(\mathbf{x}) \end{aligned}$$

[1] Se desarrollará más adelante, en la Sección 4.1

Machine learning trata fundamentalmente la generalización. Así, el problema típico de aprendizaje consiste en elegir una función de las del conjunto de hipótesis (la más adecuada) con el objetivo de etiquetar todas las instancias, incluidas aquellas más recientes y no consideradas en el conjunto de entrenamiento/validación o prueba.
Así mismo, podemos establecer diferentes escenarios en función de cómo se lleve a cabo la labor de aprendizaje. Efectivamente, en función de cómo se presente la información inicialmente disponible, así como el orden y la forma en que vayan llegando nuevos datos distinguimos principalmente los siguientes casos.

2.1. Aprendizaje Supervisado

En este escenario, el algoritmo de aprendizaje recibe un conjunto inicial de datos que incluye los valores reales de la variable que buscamos predecir (las etiquetas reales). Este conjunto se divide en los diferentes subconjuntos de entrenamiento, validación y prueba[2] y con ello se determinan los valores de los diferentes parámetros e hiperparámetros. Obtenemos así un modelo que permite obtener predicciones sobre el conjunto de prueba, ofreciendo información sobre el rendimiento de nuestro algoritmo mediante una función de pérdida.
Una vez entrenado, validado y probado/testeado, el algoritmo se emplea para obtener predicciones a partir de los nuevos datos que van llegando. Dentro del escenario supervisado se distinguen principalmente los casos de regresión y clasificación.

2.1.1. Regresión

En problemas de regresión, la etiqueta de cada instancia toma valores reales en un continuo. En estos casos suele considerarse una función de pérdida cuadrática, que para un instancia i toma la expresión

$$L(y_i, \hat{y}_i) = (\hat{y}_i - y_i)^2 . \tag{2.1}$$

Como ejemplo podemos encontrar la predicción de variables meteorológicas, así como la producción de energía solar o eólica.

2.1.2. Clasificación

En este caso, la etiqueta y asociada a cada instancia es una variable categórica y por consiguiente toma valores discretos, correspondientes a cada una de las diferentes clases. En este tipo de problemas resulta muy común hacer uso de la función de perdida 0-1, que para

[2]El tamaño de cada conjunto depende de diversas consideraciones, como puede ser el número de hiperparámetros del modelo. No obstante, como norma general, se suele definir un tamaño para el conjunto de entrenamiento sensiblemente superior al del resto.

una instancia i queda definida como sigue

$$L(y_i,\, \hat{y}_i) = \begin{cases} 0 & \text{si } y_i = \hat{y}_i\,, \\ 1 & \text{si } y_i \neq \hat{y}_i\,, \end{cases} \tag{2.2}$$

es decir, la pérdida es nula si la predicción de clase es correcta y vale 1 si es incorrecta. Notar que en el caso de tratar múltiples clases, la pérdida es la misma indiferentemente de qué clase sea predicha, con tal de que no sea la correcta.
Como ejemplo de este tipo de problemas puede considerarse la clasificación de correo como *spam* o *no spam*. En este caso la clasificación es binaria (dos clases).

2.2. Aprendizaje No Supervisado

Al contrario que los problemas de aprendizaje supervisado, donde los datos que se usan en el entrenamiento del algoritmo están previamente etiquetados, en aprendizaje no supervisado el conjunto inicial que recibe el algoritmo no incluye las etiquetas reales asociadas a cada instancia. Por tanto, el objetivo de estas técnicas de aprendizaje se centra en encontrar patrones o asociaciones entre los diferentes datos (o variables).
Entre los problemas así tratados encontramos el clustering[3], que consiste en particionar las instancias en diferentes subconjuntos homogéneos (clusters). Se comienza "a ciegas" y dado el conjunto de datos, se trata de encontrar tanto el número de clases en las que se podrían agrupar las instancias (no siempre están predefinidas) como el número de ellas que pertenecen a cada clase. Dada la falta de etiquetas puede resultar complicado medir el rendimiento del algoritmo debiendo recurrir a argumentos heurísticos (subjetivos) para validarlo. Como ejemplo podemos considerar el caso del análisis de redes sociales, donde se intenta agrupar a gran cantidad de gente en diferentes comunidades más reducidas (los clústeres) según su comportamiento.
Así mismo, se utilizan técnicas de aprendizaje no supervisado para reducir la dimensionalidad de un problema agrupando variables.

A medio camino entre el aprendizaje supervisado y el no supervisado encontramos el caso del aprendizaje semi-supervisado, mezclando instancias con y sin etiquetas. El objetivo en este caso es hacer predicciones para toda instancia. Por otro lado, también podemos distinguir el caso en que el conjunto de entrenamiento/validación presenta etiquetas mientras que en el conjunto de prueba están ausentes. El objetivo se centra en predecir únicamente sobre este último y se conoce como *transductive inference*.

2.3. Aprendizaje Online

En contraste con los escenarios de aprendizaje definidos con anterioridad, en el caso del aprendizaje online[4, 5] se suceden numerosas etapas (se pueden asociar con instantes temporales) en las que las fases de entrenamiento y prueba/testeo se encuentran interconectadas. De esta forma, en cada etapa el algoritmo recibe una instancia (o más de una) sin su etiqueta correspondiente, se realiza una predicción y al recibir el valor real se puede medir la pérdida cometida. El objetivo en esta situación consiste en minimizar la pérdida acumulada a lo largo de las diferentes etapas.
Surge como una solución para tratar con grandes conjuntos de datos, como puede ser el caso de los flujos de datos/*data streams*, en los que nuevas instancias van llegando a velocidades tan elevadas que pueden complicar el análisis. Es este el escenario en que centraremos nuestra atención en capítulos posteriores. Estos métodos cumplimentan a aquellos más tradicionales (offline o *batch*[3] *learning*) tratados anteriormente, en los que el conjunto de datos empleado para entrenar nuestro modelo es inmutable. Es decir, los datos se encuentran disponibles desde el primer momento y el modelo se entrena así una única vez. Por el contrario, en el aprendizaje online, los datos se procesan de forma secuencial, creando así un modelo inicial sin disponer de todo el conjunto de entrenamiento. Con el transcurso del tiempo van llegando nuevas instancias que se emplean para actualizar el modelo.

Otro escenario en el que se mezcla entrenamiento y prueba es el de aprendizaje por refuerzo o *reinforcement learning*. En este caso el algoritmo interacciona de forma activa con el entorno en el que se encuentra desplegado (pudiendo llegar a afectarle) y tras cada acción realizada recibe información de éste en forma de alguna noción de "recompensa". El objetivo es entonces maximizar esta recompensa, acumulada a lo largo de las diferentes acciones.

[3]Entendiendo por *batch* un conjunto de múltiples instancias.

Capítulo 3

Concept Drift: Definición, Detección y Adaptación

Con todo lo expuesto en la sección anterior, hemos establecido las bases para proseguir nuestros desarrollos. Concretamente centramos nuestra atención en el problema del *concept drift*, ligado a la no estacionariedad de los escenarios de aprendizaje *online*.
Comenzamos definiendo qué se entiende por *concept*. Recurrimos así al marco de aprendizaje PAC (*Probably Approximately Correct*)[1] y expresamos *concept* como la aplicación de $\mathcal{X}$ en $\mathcal{Y}$

$$\begin{aligned} c : \mathcal{X} &\longrightarrow \mathcal{Y} \\ \mathbf{x} &\longrightarrow y = c(\mathbf{x}) \end{aligned} \tag{3.1}$$

Es decir, relaciona las etiquetas reales con las features de cada instancia. No obstante, muchos autores adoptan una definición probabilística del término *concept*, asociándolo a la distribución de probabilidad conjunta, $p(\mathbf{x}, y)$, especialmente popular en tratados sobre *concept drift*[6, 7, 8, 9, 10].

Con esta definición podemos tratar el caso más general de aprendizaje supervisado, donde se asume la existencia de una distribución fija y desconocida $\mathcal{D}$, definida sobre $\mathcal{X} \times \mathcal{Y}$. Es decir, $\mathcal{D}$ define la distribución de probabilidad conjunta $p(\mathbf{x}, y)$ y el conjunto de entrenamiento consiste en una muestra $\mathcal{S}$ de variables independientes e idénticamente distribuidas (i.i.d.) según $\mathcal{D}$

$$\mathcal{S} = \{(\mathbf{x}_1, y_1), ..., (\mathbf{x}_m, y_m)\}.$$

Las etiquetas están definidas por un *concept* c tal que $y_i = c(\mathbf{x}_i)$, con $i = 1, ..., m$. Notar así la estrecha realción entre c y $p(\mathbf{x}, y)$ y podemos interpretar *concept* como el conjunto de instancias/datos cuya distribución de probabilidad subyacente es estacionaria.

[1] El aprendizaje PAC (*Probably Approximately Correct*) permite establecer un marco de referencia para el análisis matemático de las técnicas tratadas en Machine Learning.
En este escrito utilizaremos algunos resultados básicos pero si se desea disponer de información más detallada al respecto recomendamos consultar otras fuentes[2]. Este libro de referencia cubre los principales temas y tópicos del machine learning más actual, proporcionando las bases y fundamentos teóricos necesarios para nuestros posteriores desarrollos.

En la labor de aprendizaje se considera el conjunto $\mathcal{H}$ de posibles hipótesis y se usa la muestra $\mathcal{S}$ para elegir aquella hipótesis $h \in \mathcal{H}$ que resulte en un menor error de generalización respecto al *concept* c, $R(h)$, definido como

$$R(h) = \mathop{\mathbb{P}}_{(\mathbf{x},\, y)\sim\mathcal{D}}[h(\mathbf{x}) \neq c(\mathbf{x})] = \mathop{\mathbb{E}}_{(\mathbf{x},\, y)\sim\mathcal{D}}[L(h(\mathbf{x}), c(\mathbf{x}))] = \mathop{\mathbb{E}}_{(\mathbf{x},\, y)\sim\mathcal{D}}[L(\hat{y}, y)]\,, \tag{3.2}$$

siendo $L\,(y,\, \hat{y})$ una función de perdida. El desconocimiento tanto del *concept* c como de la distribución $\mathcal{D}$ nos impide obtener el valor de $R(h)$. Sin embargo, podemos hallar el error empírico sobre $\mathcal{S}$ definido como

$$\hat{R}_{\mathcal{S}}(h) = \frac{1}{m}\sum_{i=1}^{m} L(h(\mathbf{x}_i), c(\mathbf{x}_i)) = \frac{1}{m}\sum_{i=1}^{m} L(\hat{y}_i, y_i)\,. \tag{3.3}$$

El error empírico de h es el promedio de la pérdida cometida al aplicar la hipótesis sobre la muestra $\mathcal{S}$ mientras que el error de generalización es el valor esperado de la pérdida basado en la distribución $\mathcal{D}$. Así, el en el caso de regresión con (2.1), $\hat{R}_{\mathcal{S}}(h)$ representaría el error cuadrático medio o *Mean Squared Error*. En el caso de clasificación, usando (2.2) la expresión anterior daría cuenta del cociente entre clasificaciones erróneas y clasificaciones totales.

Por contraste, en el aprendizaje online se mezclan las fases de entrenamiento y prueba y no se asume ninguna distribución estacionaria $\mathcal{D}$. Así mismo, en vez de disponer de una muestra inicial fija $\mathcal{S}$, consideramos que en cada etapa t el algoritmo recibe datos en forma de tuplas $\mathcal{S}_t = \{(\mathbf{x}_t,\, y_t)\}$. Procediendo así, consideremos T etapas, de forma que en la etapa t-ésima el algoritmo recibe una instancia con sus correspondientes features $\mathbf{x}_t$ y realiza una predicción $\hat{y}_t$. Antes de la etapa $t+1$, es decir, antes de disponer de $\mathbf{x}_{t+1}$, el algoritmo recibe el valor real y_t e incurre una pérdida $L(\hat{y}_t,\, y_t)$. Esta información es usada por el algoritmo en el aprendizaje con el objetivo de minimizar la pérdida acumulada a lo largo de las sucesivas etapas, que escribimos como

$$L_T = \sum_{t=1}^{T} L(\hat{y}_t,\, y_t)\,. \tag{3.4}$$

Como antes, $y_t = c_t(\mathbf{x}_t)$, aunque en este caso denotamos el *concept* en cada etapa con el respectivo subíndice t, indicando que puede variar (en escenarios offline solo se considera un único *concept*). Efectivamente, el flujo de datos/*data stream* toma la forma de una sucesión, $\{\mathcal{S}_t\}$, de longitud potencialmente infinita y no es realista considerar que el *concept* es el mismo en todas las etapas. Del mismo modo, al no hacer asunciones sobre la estacionariedad de la distribución subyacente, asumimos la posibilidad que ésta puede cambiar con el tiempo, especialmente en entornos con alta variabilidad y donde prima la no estacionariedad, dando lugar al problema del *concept drift*.

Notar que siempre disponemos de las valores reales de las etiquetas asociadas a cada instancia, aunque sea con posterioridad a realizar cada predicción. Por tanto, hay algunos

autores[7] que enmarcan el *concept drift* dentro de un escenario de aprendizaje online *supervisado*. Ambos casos se refieren a lo mismo, tratando además problemas tanto de clasificación como de regresión, aunque este último caso no se encuentra tan desarrollado, de modo que escasea la literatura al respecto. Con motivos de remediar esta situación, en posteriores capítulos trataremos de analizar casos de regresión en presencia de *concept drift*.

Usando la definición de *concept* más habitual, como distribución de probabilidad conjunta $p(\mathbf{x},\, y)$, el *concept drift* entre dos etapas o instantes de tiempo t y $t+1$ puede expresarse formalmente

$$\exists\, \mathbf{x} \;:\quad p_t(\mathbf{x}\,, y) \neq p_{t+1}(\mathbf{x}\,, y)\,, \tag{3.5}$$

donde $p_t(\mathbf{x}\,, y)$ y $p_{t+1}(\mathbf{x}\,, y)$ definen las distribuciones conjuntas en sendas etapas/instantes temporales. Entonces, como podemos descomponer $p(\mathbf{x}\,, y) = p(\mathbf{x})\, p(y \,|\, \mathbf{x})$, cambios en los datos analizados pueden caracterizarse como cambios en las componentes de esta relación, estableciendo la siguiente clasificación:

– *Real Concept Drift*. Se refiere a cambios en $p(y \,|\, \mathbf{x})$, que pueden venir acompañados o no de cambios en $p(\mathbf{x})$

$$\begin{gathered} p_t(\mathbf{x}) = p_{t+1}(\mathbf{x}) \;\wedge\; p_t(y \,|\, \mathbf{x}) \neq p_{t+1}(y \,|\, \mathbf{x}) \\ \text{ó} \\ p_t(\mathbf{x}) \neq p_{t+1}(\mathbf{x}) \;\wedge\; p_t(y \,|\, \mathbf{x}) \neq p_{t+1}(y \,|\, \mathbf{x})\,. \end{gathered} \tag{3.6}$$

  Causan un deterioro de la capacidad predictiva del modelo y por tanto requieren de actuación inmediata.

– *Virtual Concept Drift*. Se refiere a cambios en $p(\mathbf{x})$ sin afectar a $p(y \,|\, \mathbf{x})$. Es decir

$$p_t(\mathbf{x}) \neq p_{t+1}(\mathbf{x}) \;\wedge\; p_t(y \,|\, \mathbf{x}) = p_{t+1}(y \,|\, \mathbf{x})\,. \tag{3.7}$$

  Estos cambios no afectan a la capacidad predictiva y por tanto, en la práctica no es estrictamente necesario su tratamiento. Sin embargo, puede resultar instructivo para comprender como se comportan los datos con el paso del tiempo.

Como **ejemplo** para facilitar la compresión podemos remitirnos al caso de una aplicación/plataforma de entretenimiento con la función de recomendar programas y series de televisión, así como películas, a sus usuarios en base las preferencias de cada uno (como puede ser el caso de Netflix). Según esto, para un cierto usuario, los programas de la plataforma se clasificarán en *interesantes* o *no interesantes* (variable objetivo predicha $\hat{y}$) según las características de los mismos (variables predictoras $\mathbf{x}$: el género, la duración...). Así, por cada elección que realiza el usuario (en cada etapa), el algoritmo recibe un *feedback* estableciendo

si efectivamente era de su gusto o no (llegada del valor real y) y el modelo aprende para realizar posteriores sugerencias (predicciones en etapas posteriores).
Asumamos por ejemplo que el algoritmo establece como *no interesantes* aquellos contenidos audiovisuales cuya duración sea inferior a 40 minutos, como es el caso de muchas series. Esto es indicativo de la preferencia del usuario por las series más extensas o películas y podemos plantear los siguientes escenarios:

- Si el usuario últimamente solo consume películas dado que dispone de más tiempo de ocio, la distribución de las características del material consumido experimentará un cambio en la variable que define la duración (cambio en $p(\mathbf{x})$). No obstante, la restricción anterior sigue siendo totalmente válida pues la gran mayoría de películas superan los 40 minutos de duración y por tanto no hay cambio en las predicciones (no cambia $p(y \mid \mathbf{x})$). Estamos ante *virtual concept drift*.

- Si por el contrario el usuario comienza a ver más series cortas por falta de tiempo, entonces tenemos de nuevo cambio en $p(\mathbf{x})$, pero la restricción anterior pierde su validez. Por tanto cambia $p(y \mid \mathbf{x})$ y estamos ante *real concept drift*. Por otro lado, puede ser que un mismo usuario, sin modificar sus preferencias, le permita elegir qué contenido visualizar a un tercero. Estamos por tanto ante una persona con gustos que pueden ser muy diferentes e incluso aunque no varíe $p(\mathbf{x})$, el cambio puede ser notable en $p(y \mid \mathbf{x})$.

Con este ejemplo resulta fácil entender la importancia que tiene la detección del *concept drift* así como su adecuado tratamiento en el mantenimiento de los modelos predictivos. No obstante, encontramos que su utilidad se extiende a ámbitos de diversa índole, como pueden ser los campos de la medicina, la industria o la educación.
En nuestro caso, el entorno industrial resulta unos de los principales focos de atención y por tanto, el tratamiento del *concept drift* tiene gran uso en el **control y monitorización**. Podemos considerar así el tratamiento de las medidas proporcionadas por diferentes sensores, donde la detección del *drift* puede llegar a evitar problemas en gran variedad de procesos industriales e incluso llegar a indicar un deterioro en el propio sensor (con una consecuente pérdida tanto de información como monetaria).

3.1. Tipos de Concept Drift

Una vez comprendido mejor qué se entiende por *concept drift* resulta interesante estudiar la tipología de cambios que engloba. Así, atendiendo a la velocidad con la que ocurre el *concept drift*, comúnmente se establece una división en diferentes tipos[7, 8, 11], como se indica en la Figura 3.1:

(a) *Abrupt Concept Drift/ Concept Drift* Abrupto. En muy pocas instancias el nuevo *concept* reemplaza al ya establecido. La precisión se degrada rápidamente.

(b) *Gradual Concept Drift/ Concept Drift* Gradual. Cuando el nuevo *concept* reemplaza al anterior a lo largo de un periodo ciertamente extenso; progresivamente encontramos más ocurrencias del nuevo *concept* y menos de las del antiguo. Es decir, las instancias se generan a partir de una mezcla de *concepts* hasta que, normalmente, terminan por predominar instancias de un solo *concept*.

(c) *Incremental Concept Drift/ Concept Drift* Incremental. Aparecen *concepts* intermedios en el cambio del *concept* inicial al final. Se caracterizan por ser los *drifts* más lentos.

(d) *Recurring Concept Drift/ Concept Drift* Recurrente. Cuando un *concept* que se dio con anterioridad vuelve a estar presente tras cierto tiempo. No confundir con estacionalidad pues no es periódico; el *drift* es impredecible.

También resulta importante considerar

(e) *Blips*. Son cambios muy repentinos en el *concept* (anomalía). Pueden interpretarse como *outliers* en una distribución estacionaria de datos.

(f) *Noise/* Ruido. Se tratan de desviaciones aleatorias en el *concept* que deben ser debidamente filtradas.

Estos dos últimos casos no suponen *concept drift* y por lo tanto no se consideran dentro de la tipología. No obstante, su correcto tratamiento resulta indispensable para el correcto funcionamiento de nuestros algoritmos.

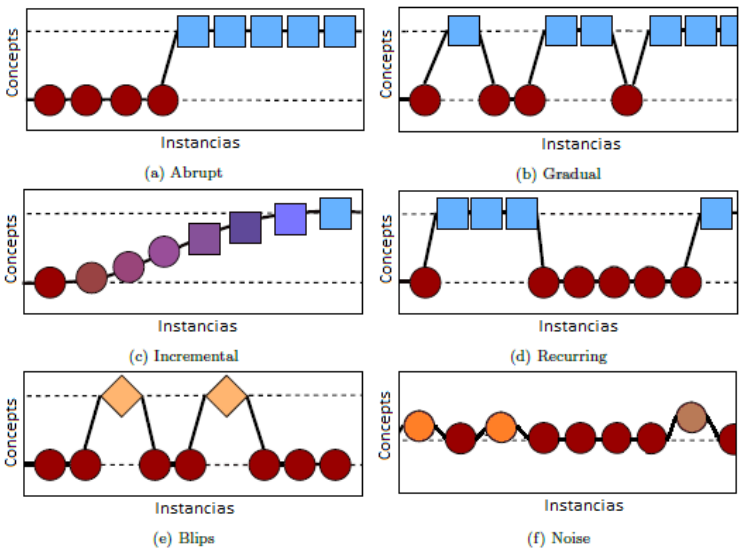


Figura 3.1: Tipos de *concept drift* atendiendo a la velocidad. Imagen adaptada[11].

3.2. Detección de Concept Drift

Hemos visto como la no estacionariedad de la distribución subyacente implica cambios en el *concept* que nuestro algoritmo intenta "aprender" dando lugar al problema del *concept drift*. En esta situación no resulta conveniente hacer uso de las métricas más tradicionales, estableciendo un conjunto de prueba sobre el que evaluar el error empírico (3.3). Puesto que las predicciones se realizan de forma secuencial, de etapa en etapa, únicamente disponemos de la pérdida acumulada al final de cada una (3.4). Este enfoque predictivo secuencial o precuencial (*prequential*)[12] nos lleva a calcular el error del algoritmo a partir de esta suma. Así, dado $\mathcal{S}_t = \{(\mathbf{x}_t,\, y_t)\}$ en una etapa $t \geq 1$ computamos la función de pérdida, $L\,(y_t,\, \hat{y}_t)$, y el error precuencial se define como

$$P_e(t) = \frac{1}{t} \sum_{k=1}^{t} L\,(y_k,\, \hat{y}_k)\ , \tag{3.8}$$

o de forma iterativa, puesto que hay situaciones en las que no es posible disponer de todas las predicciones de etapas anteriores (por ejemplo si hay poca memoria disponible)

$$P_e(t) = \frac{L\,(y_t,\, \hat{y}_t) + (t-1)P_e(t-1)}{t}\,, \qquad \text{con } P_e(t=0) = 0\,. \tag{3.9}$$

De este modo, la evaluación precuencial permite monitorizar cómo evoluciona el aprendizaje del modelo, ofreciendo cierta robustez frente al ruido y potenciales *outliers*.

Sin embargo, el error precuencial, calculado de esta forma puede estar altamente influenciado por las predicciones realizadas en las primeras etapas, comprometiendo nuestros resultados. Esto es especialmente notable en aquellos algoritmos incrementales, donde el modelo inicial se obtiene a partir de una muestra de tamaño reducido y se va actualizando en cada etapa. Es por ello que resulta interesante computar el error cometido implementando algún tipo de mecanismo que permita "olvidar aquella información más antigua". Esto puede conseguirse mediante el uso de ventanas (*windowing*), que permiten hacer particiones del conjunto de datos total en diferentes subconjuntos, cuyo tamaño puede variar. Así, asumiendo una ventana de tamaño w, el error se expresa como

$$P_w(t) = \frac{1}{w} \sum_{k=t-w+1}^{t} L\,(y_k,\, \hat{y}_k)\ . \tag{3.10}$$

En este caso resulta conviente mantener en memoria al menos un número de datos recientes igual al tamaño de la ventana considerada w.

Otra forma consiste en usar factores α que permitan dar menor peso conforme más antigua es la predicción. Se conocen como *fading factors* y el error se expresa

$$P_\alpha(t) = \frac{\displaystyle\sum_{k=1}^{t} \alpha^{t-k}\, L\,(y_k,\, \hat{y}_k)}{\displaystyle\sum_{k=1}^{t} \alpha^{t-k}}\,, \qquad \text{con } 0 \ll \alpha \leq 1\,, \tag{3.11}$$

o de forma iterativa

$$P_\alpha(t) = \frac{S_\alpha(t)}{N_\alpha(t)}\,, \tag{3.12}$$

definiendo

$$S_\alpha(t) = L(y_t,\, \hat{y}_y) + \alpha\, S_\alpha(t-1)\,, \qquad \text{con } S(t=0) = 0\,, \tag{3.13}$$

$$N_\alpha(t) = 1 + \alpha\, N_\alpha(t-1)\,, \qquad \text{con } N(t=0) = 0\,, \tag{3.14}$$

Se ha demostrado además[12], que el error precuencial es pesimista y en problemas de clasificación tiene como límite inferior, el error bayesiano[2].

Una vez definidas las métricas que permiten evaluar nuestros algoritmos en escenarios de aprendizaje online podemos proceder a definir mecanismos que permitan una detección explícita del *concept drift* al monitorizar cambios en este error y por tanto cambios en $p(y \,|\, \mathbf{x})$. No existe una clasificación fija para los diferentes métodos de detección pues varía según el autor y está en continua evolución, aunque hay que notar que en muchos casos vienen a ser prácticamente equivalentes. Nuestro objetivo principal es la implementación de varios de estos métodos y por ello propondremos una división en tres categorías[7, 13].

3.2.1. Análisis secuencial

Se analiza la existencia de patrones en los nuevos conjuntos de datos (aquellos que van llegando) y se generan alarmas de posible *concept drift* cuando el cambio en su distribución excede un límite/umbral/*threshold* previamente especificado.

Como ejemplo tenemos el ***Page-Hinkley Test***[14, 15]. Este test permite analizar los datos (errores) que van llegando de forma secuencial en el tiempo, $\{x_t\}$. Se emplea típicamente en la detección de cambios en la media de una señal gaussiana (sigue una distribución normal) pero ofrece resultados robustos al aplicarse sobre distribuciones no gaussianas y por ello es ciertamente usado para la detección de *concept drift* en flujos de datos. El test considera una variable cumulativa m_T, definida como la diferencia acumulada entre los valores observados y su media hasta esa instancia T, es decir

$$m_T = \sum_{t=1}^{T} \left(x_t - \bar{x}_T - \delta\right)\,, \qquad \text{con} \quad m_0 = 0\,, \tag{3.15}$$

donde $\bar{x}_T = 1/T \sum_{t=1}^{T} x_t$ y δ controla la magnitud de los cambios permitidos.
También guardamos el valor mínimo de esta variable $M_T = \text{mín}\,(m_t\,,\, t = 0\,,\, 1\,, ...,\, T)$, de modo que el test monitoriza la diferencia entre ambas, $m_T - M_T$, y cuando esta supera un cierto umbral/*threshold*, λ, señala un cambio en la distribución y por tanto un cambio en el *concept*. El valor de este parámetro λ vendrá fijado por el usuario, de modo que un valor

[2]Dado un problema de clasificación, se define el clasificador bayesiano como aquella hipótesis h tal que su error de generalización $R(h)$ es mínimo. Éste es el error bayesiano, análogo al error irreducible.

pequeño puede ocasionar falsas alarmas en la detección de *concept drift* mientras que un valor mayor dará lugar a menos falsas alarmas pero puede llegar a retrasar la detección e incluso no detectar los cambios.

3.2.2. Control estadístico

Se basan en el output de los modelos midiendo las fluctuaciones en la tasa de error.

Entre los ejemplos más destacables encontramos el ***Drift Detection Method (DDM)***[6, 7, 16], uno de los primeros algoritmos explícitos desarrollados que permite la detección de *concept drift* en problemas de clasificación.

Supongamos una secuencia de instancias de la forma $(\mathbf{x}_i\,,\,y_i)$, donde para cada valor de i el modelo de decisión predice $\hat{y}_i$, que puede ser True ($\hat{y}_i = y_i$) o False ($\hat{y}_i \neq y_i$). Para el conjunto de instancias el error cometido en la predicción puede entenderse como una variable aleatoria que sigue una distribución binomial y representa el número de errores en una muestra con n datos. Así, por cada instancia i en esta secuencia, la tasa de error es la probabilidad de observar False, p_i, con desviación estándar dada por $s_i = \sqrt{p_i\,(1-p_i)/i}$.

De acuerdo con el marco de aprendizaje PAC (*Probably Approximately Correct*)[2, 17], se asume que si la distribución subyacente de la sucesión de datos estudiada permanece estacionaria, la tasa de error de nuestro algoritmo de clasificación, p_i, disminuirá a medida que aumente el número de instancias[3]. Por lo tanto, un incremento significativo en la tasa de error sugiere un cambio en la distribución de probabilidad de la variable y (cambio de *concept*) y el modelo deja de ser apropiado. Así mismo, con un número de datos lo suficientemente elevado la distribución binomial se aproxima a una distribución normal (o gaussiana) con igual media y varianza. Entonces, el intervalo de confianza $1-\delta/2$ para la variable con un número de datos suficiente (generalmente $n \geq 30$) es aproximadamente $p_i \pm \alpha \cdot s_i$.

Con todo esto en mente, el método de detección del *concept drift* aquí estudiado trabaja con los valores de mínimos, $p_{\text{mín}}$ y $s_{\text{mín}}$, actualizándolos cuando al procesar una nueva instancia, la suma $p_i + s_i$ es inferior al valor $p_{\text{mín}} + s_{\text{mín}}$. Por el contrario, cuando $p_i + s_i$ es superior a $p_{\text{mín}} + s_{\text{mín}}$ distinguimos

- $p_i + s_i \geq p_{\text{mín}} + 2 \cdot s_{\text{mín}}$ establece un nivel de confianza del 95 % ($\delta = 0.95$) para indicar alarma/*warning*, es decir, se alerta de posible *concept drift* en las instancias siguientes. Por ello, desde este instante, se van almacenando las instancias que van llegando en preparación para el posible *concept drift*.

- $p_i + s_i \geq p_{\text{mín}} + 3 \cdot s_{\text{mín}}$ establece un nivel de confianza del 99 % ($\delta = 0.99$) para señalar que ha ocurrido un cambio de *concept* y es necesario reconsiderar el modelo de clasificación. Así mismo, se reinician los valores mínimos $p_{\text{mín}}$ y $s_{\text{mín}}$.

[3] Tendiendo al error bayesiano para una sucesión infinita.

Una vez detectado el *drift*, el nuevo *concept* se declara empezando en la instancia que primero hizo saltar la señal de alarma y se crea un nuevo modelo de decisión usando solo estos datos (desde la alarma hasta que se explicita la detección). Por otro lado, en caso de indicarse el nivel de alerta seguido de un descenso en la tasa de error sin llegar a detectar *drift*, asumiremos que se trata de una falsa alarma sin incurrir en cambio de *concept*.

También resulta interesante considerar el ***Early Drift Detection Method (EDDM)***[16]. Este algoritmo se basa en el anterior *Drift Detection Method* y se desarrolló para mejorar la detección de *concept drift* gradual manteniendo un buen rendimiento en la detección de *concept drift* abrupto. Principalmente consiste en monitorizar la distancia entre errores, representada por el número de etapas entre dos malas clasificaciones consecutivas, en lugar del número de ellas. Como con DDM, sucede que si la distribución de los datos es estacionaria la distancia entre errores aumentará a medida que aumenta el número de muestras analizadas, de modo que una disminución en la distancia implicará un cambio en dicha distribución (cambio de *concept*). Por ello consideraremos la distancia promedio entre errores, p'_i, así como su desviación estándar, s'_i, y guardamos los valores $p'_{\text{máx}}$ y $s'_{\text{máx}}$ que maximizan la suma $p'_i + 2 \cdot s'_i$. El valor $p'_{\text{máx}} + 2 \cdot s'_{\text{máx}}$ se corresponde con el punto en el que la distribución de la distancia entre errores es máxima y es entonces donde la hipótesis del modelo mejor se aproxima al *concept*. Igual que en el caso de DDM, se definen dos niveles:

- $(p'_i + 2 \cdot s'_i) \,/\, (p'_{\text{máx}} + 2 \cdot s'_{\text{máx}}) < \alpha$ define el nivel de alarma/*warning*, indicando que se acerca un posible cambio de *concept*. Por ello, desde este instante, se van almacenando las instancias que van llegando en preparación para el posible *concept drift*.

- $(p'_i + 2 \cdot s'_i) \,/\, (p'_{\text{máx}} + 2 \cdot s'_{\text{máx}}) < \beta$ define el nivel de *drift*, indicando que ha ocurrido *concept drift* y la necesidad de reconsiderar el modelo de clasificación empleado. Así mismo, se reinician los valores máximos $p'_{\text{máx}}$ y $s'_{\text{máx}}$ para posteriores detecciones.

También se define un número mínimo de errores (generalmente en torno a 30 errores) previos a la ocurrencia del *concept drift*. Una vez transcurrido este número mínimo, los diferentes niveles se encargan de la detección, para los que se han determinado (tras cierta experimentación) valores de los parámetros $\alpha = 0.95$ y $\beta = 0.90$.
En el caso de tratarse de una falsa alarma, las instancias almacenadas terminan por ser eliminadas y la detección continúa su curso natural.

3.2.3. Análisis de distribuciones

Generalmente se basan en dividir los datos en dos ventanas, una incluyendo aquellas instancias más antiguas y otra con la información más reciente. Se comparan así las distribuciones en cada una de estas ventanas usando test estadísticos con la hipótesis nula de que ambas son idénticas. Si se detecta cambio se declara la presencia de *concept drift*.

Un buen representante de esta metodología (y probablemente uno de los más utilizados) es el algoritmo ***ADaptive-WINdowing (ADWIN)***[18] que consiste en tomar ventanas móviles (*sliding windows*) cuyo tamaño, en vez de fijarse a priori varía en función de los cambios observados en los datos contenidos en la propia ventana. El algoritmo aumenta de forma automática el tamaño de la ventana (con las instancias más recientes) cuando no se detecta ningún cambio y, en caso de detectarse, disminuye su tamaño permitiendo mantener aquellos datos más relevantes del *concept* actual. La ventana considerada contiene bits o números reales, de modo que puede usarse para monitorizar el error del modelo de predicción. Como entrada al algoritmo tenemos un parámetro definido por el usuario que da cuenta de la confianza $\delta \in [0\,,\,1]$ y una secuencia de valores reales $\{x_t\}$ posiblemente infinita. Cada valor x_t está disponible solamente a partir de la etapa/instante t y sigue una distribución $\mathcal{D}_t$ (de media desconocida μ_t) de forma independiente para cada t. Así mismo asumimos que los valores se encuentran en el intervalo $[0\,,\,1]$, lo cual se cumple fácilmente imponiendo un simple re-escalado.

Sea W la ventana móvil conteniendo valores hasta el más reciente (x_t) y n su tamaño, computamos el valor promedio (observado) de sus elementos, $\hat{\mu}_W$, asociado al valor esperado desconocido μ_t, con $t = W$. La idea principal consiste en que cuando dos subventanas "lo suficientemente grandes" de W muestran promedios "lo suficientemente distintos", podemos concluir que los valores esperados correspondientes son diferentes y por tanto podemos prescindir de la porción de datos más antigua de W. Para ello debemos establecer un valor de corte, ϵ_{cut}, para toda partición de W en W_0 y W_1. Así, sean n_0 y n_1 sus correspondientes tamaños ($n = n_0 + n_1$), $\hat{\mu}_{W_0}$ y $\hat{\mu}_{W_1}$ los promedios observados en cada subventana y μ_{W_0} y μ_{W_1} los valores esperados escribimos

$$m = \frac{1}{1/n_0 + 1/n_1}\,, \qquad \delta' = \frac{\delta}{n}\,, \tag{3.16}$$

$$\epsilon_{\text{cut}} = \sqrt{\frac{1}{2m} \cdot \log\left(\frac{4}{\delta'}\right)}\,. \tag{3.17}$$

El test estadístico propuesto simplemente comprueba si los promedios observados en cada subventana difieren entre sí más de ϵ_{cut}, es decir, si su diferencia excede el umbral ϵ_{cut} propuesto. Procediendo de este modo eliminamos los elementos más antiguos de W mientras se cumpla $|\hat{\mu}_{W_0} - \hat{\mu}_{W_1}| \geq \epsilon_{\text{cut}}$.

En la práctica, la definición (3.17) puede resultar demasiado conservadora[4]. La diferencia $\mu_{W_0} - \mu_{W_1}$ tiende a una distribución normal para grandes ventanas y tomamos

$$\epsilon'_{\text{cut}} = \sqrt{\frac{2}{m} \cdot \sigma^2_W \cdot \log\left(\frac{2}{\delta'}\right)} + \frac{2}{3\,m} \log\left(\frac{2}{\delta'}\right). \tag{3.18}$$

[4]La primera definición de ϵ_{cut} está basada en la desigualdad de Hoeffding[19] y es válida para todas las distribuciones pero tiende a sobre estimar en gran medida la probabilidad de grandes desviaciones en distribuciones con pequeña varianza.

3.3. Adaptación en presencia de Concept Drift

Una vez definidos algunos de los (numerosos) métodos para detectar el *concept drift*, debemos estudiar las diferentes formas en que podemos afrontar la adaptación al nuevo *concept*, tratando así con algoritmos adaptativos.
No obstante, puesto que el método de adaptación depende en gran medida del problema tratado y esto influencia la forma en que se encuentra disponible la información usada en el aprendizaje, resulta indispensable mencionar diferentes situaciones que pueden presentarse. Así, encontramos casos en los que las instancias que llegan en cada etapa no están disponibles en etapas posteriores, normalmente debido a la reducida memoria con la que se trabaja. Por otro lado, pueden darse situaciones en las que la memoria del sistema permita disponer de múltiples instancias a medida que se suceden las etapas. Entonces, puede resultar interesante considerar algún tipo de mecanismo para mantener únicamente aquellas más recientes, pues presentan la información más relevante. Para ello, lo más común resulta tomar ventanas (igual que con el error precuencial) cuyo tamaño puede ser variable o fijo; así mismo, puede considerarse darle menor importancia a aquellas instancias de etapas más distantes.
Una vez conscientes de las limitaciones en el tratamiento de la información, se establecen dos enfoques de adaptación diferente que siguen los algoritmos en escenarios de aprendizaje online con posibilidad de *concept drift*.

3.3.1. Adaptación pasiva/blind

Las estrategias seguidas en este contexto no buscan una detección explícita del *concept drift*, sino que directamente asumen que el *concept* es susceptible de cambiar al sucederse las etapas. Asumen por tanto que la distribución de probabilidad de los datos cambia en el tiempo de forma impredecible y actúan al respecto. Para adecuarse a los nuevos *concepts* se realiza una adaptación continua del modelo empleado con la llegada de nueva información. Dado que esta adaptación se extiende en el tiempo, la metodología resulta de gran utilidad en casos de *concept drift* gradual y *concept drift* incremental, es decir, aquellos cambios más lentos. Distinguimos diferentes casos:

- Modelos individuales. Muy útil en casos con flujos de datos masivos dado su menor coste computacional. Aquí se encuentran enmarcados los algoritmos incrementales, en los que el modelo empleado se actualiza en cada etapa t, de modo que no es necesario mantener instancias en memoria (una forma puede ser modificando los parámetros del modelo). Así mismo, también puede hacerse uso de ventanas (lo que implica almacenar cierta cantidad de información pasada) con las que realizar un re-entrenamiento periódico con las instancias más recientes.

– Conjunto/*ensemble* de modelos. Basados en el empleo de múltiples modelos a la vez ofrecen una combinación (ponderada/*weighted*) de las predicciones como resultado final. Son más costosos computacionalmente pero proporcionan resultados más precisos y con menos varianza y permiten una fácil incorporación de nuevos datos mediante la adición de nuevos modelos. La adaptación tiene lugar modificando los pesos usados en el computo de las predicciones, pudiendo ser nulos para los modelos irrelevantes.
  Es importante mencionar que resultan de gran utilidad en casos de *concept drift* recurrente pues puede suceder que los pesos de modelos individuales que en un *concept* ofrecían buenos resultados (y tras ocurrir el *drift* vean su importancia/peso reducido) vuelvan a adquirir validez al retomarse el antiguo *concept*. Basta entonces con recalcular los pesos, evitando una costosa reconstrucción/re-entrenamiento.

Aunque los algoritmos pasivos no emplean una detección explícita del *concept drift* para adaptarse a un entorno en continuo cambio, es interesante hacer notar que sigue siendo totalmente válido usar los métodos de detección. Aunque no se utilicen en la adaptación, proporcionan información muy valiosa sobre la dinámica del proceso generativo de los datos.

3.3.2. Adaptación activa/informed

Se basan en la detección explícita de *concept drift* a través de diversos mecanismos, algunos de los cuales se tratarion en la sección anterior. La idea principal consiste en mantener un número significativo de instancias recientes en memoria de forma que una vez detectado el cambio de *concept* se inicia una reconstrucción del modelo, usando dichas instancias para su entrenamiento. Esto permite que el modelo se adapte correctamente al nuevo *concept* y resulta especialmente adecuado para tratar *concept drift* abrupto puesto que el cambio en es más fácil de detectar que en el caso de *concept drift* gradual.
Como ejemplo tenemos los casos de DDM y EDDM en los que al indicarse la señal de alarma o *warning*, se empiezan a almacenar en memoria instancias con las que reconstruir el modelo en caso de ocurrir el *concept drift*. Por otro lado ADWIN ofrece una ventana cuyo tamaño se adapta automáticamente, permitiendo disponer en todo momento de aquellas instancias más relevantes. En el caso del Test de Page-Hinkley, solo se indica la ocurrencia del *concept drift*.

En el Apéndice A se detallan los algoritmos anteriormente tratados. Así mismo, hemos optado por realizar una implementación en el lenguaje de programación R (muy usado en el Máster) de los diferentes diferentes métodos y algoritmos tratados en la sección anterior para la detección de *concept drift*. Todo el código pertinente puede encontrarse en el Apéndice C.

Capítulo 4

Casos Prácticos: Regresión

Para facilitar la comprensión de todo lo desarrollado hasta ahora aplicaremos los diferentes métodos adaptativos en casos prácticos, comenzando con regresión. A pesar de que en este caso hay mucha menos literatura referente al *concept drift*, podemos crear conjuntos de datos de forma sintética[20] y asumir un tratamiento online de los mismos, considerando que las instancias van llegando en forma de tuplas asociadas a un flujo/*stream* $\{\mathcal{S}_t\}$. Para ello utilizamos una serie de 300000 valores para 5 variables predictoras continuas, $\mathbf{x} = (x_1, x_2, x_3, x_4, x_5)^T$, independientes y distribuidas todas ellas de forma uniforme en el intervalo $[0\,,\,1]$. Con ellas definiremos una variable objetivo y mediante tres funciones lineales diferentes para establecer así 3 *concepts*:

$$y = c_1(\mathbf{x}) = 10\,x_1 + x_2 + 20\,(x_3 - 0.5) + 10\,x_4 + 5\,x_5 + \epsilon\,, \tag{4.1}$$

$$y = c_2(\mathbf{x}) = 0.5\,x_1 - x_2 + 100\,x_3 + 10\,x_4 - 54\,x_5 + \epsilon\,, \tag{4.2}$$

$$y = c_3(\mathbf{x}) = 3\,x_1 + 22\,x_2 - 8\,(x_3 - 0.5) + 2\,x_4 - 5\,x_5 - \epsilon\,. \tag{4.3}$$

En cada caso se ha añadido ruido aleatorio en forma de una distribución normal de media 0 y varianza 1 ($\epsilon \sim N(0\,,\,1)$). Podemos entonces definir 3 conjuntos de datos diferentes:

- Dataset I. Este conjunto simula un flujo de datos sin presencia de *concept drift*. La función utilizada para crear la variable objetivo en todo instante es (4.1).

- Dataset II. Este conjunto simula la presencia de *concept drift* abrupto. Para ello se introducen 3 puntos en los que cambia el *concept* a partir de un cambio en los parámetros que definen y. Concretamente, el *concept* inicial se corresponde con (4.1) y a partir de la instancia 75000 cambiamos la expresión de y a la función dada por (4.2). A continuación, al rededor de la instancia 150000, volvemos a introducir un cambio en el *concept*, utilizando ahora (4.3). Finalmente implementamos un último *drift*, retomando el *concept* definido por (4.2), sobre la instancia 225000. De esta forma cada cuarto del dataset está definido por un *concept*.

– Dataset III. Este conjunto simula la presencia de *concept drift* gradual. Para ello se introducen 2 puntos en los que comienza el *concept drift* redefiniendo de nuevo los parámetros de que depende y. Inicialmente tenemos una vez más la expresión de y dada por (4.1) y el *concept drift* se inicia por primera vez en la instancia 100000. A partir de entonces van apareciendo gradualmente instancias del nuevo *concept*, definido por (4.2) y cada 500 instancias aumenta la probabilidad de generar instancias de este nuevo *concept* (en nuestro caso optamos por un incremento fijo de la probabilidad en los puntos indicados). Una vez transcurridas 50000 instancias solo se encuentran puntos definidos por (4.2). Finalmente comienza otro *concept drift* (procediendo de igual forma) a partir de la instancia 200000; usamos ahora (4.3) y acaba imponiéndose transcurridas otras 50000 instancias.

Notar como para cada *concept* podemos definir una media y una varianza para la distribución de y, dependiente de $\mathbf{x}$. Efectivamente, sabiendo que para una distribución uniforme en el intervalo [0, 1] la media es 0.5 y su varianza $1/12$[1] y que además ϵ presenta media nula y varianza unidad, en el caso en que y está definida por (4.1) se tiene

$$\begin{aligned} \mathbb{E}[y(\mathbf{x})] =& 10\,\mathbb{E}[x_1] + \mathbb{E}[x_2] + 20\,(\mathbb{E}[x_3] - 0.5) + 10\,\mathbb{E}[x_4] + \\ & 5\,\mathbb{E}[x_5] + \mathbb{E}[\epsilon] = 13\,, \\ Var[y(\mathbf{x})] =& 10^2\,Var[x_1] + Var[x_2] + 20^2\,Var[x_3] + 10^2\,Var[x_4] + \\ & 5^2\,Var[x_5] + Var[\epsilon] = 638/12 \approx 53.1667\,. \end{aligned} \tag{4.4}$$

Del mismo modo, para (4.2)

$$\begin{aligned} \mathbb{E}[y(\mathbf{x})] =& 0.5\,\mathbb{E}[x_1] - \mathbb{E}[x_2] + 100\,\mathbb{E}[x_3] + 10\,\mathbb{E}[x_4] - \\ & 54\,\mathbb{E}[x_5] - E[\epsilon] = 27.75\,, \\ Var[y(\mathbf{x})] =& 0.5^2\,Var[x_1] + Var[x_2] + 100^2\,Var[x_3] + 10^2\,Var[x_4] + \\ & 54^2\,Var[x_5] + Var[\epsilon] = 1085.771\,. \end{aligned} \tag{4.5}$$

Finalmente, con (4.3)

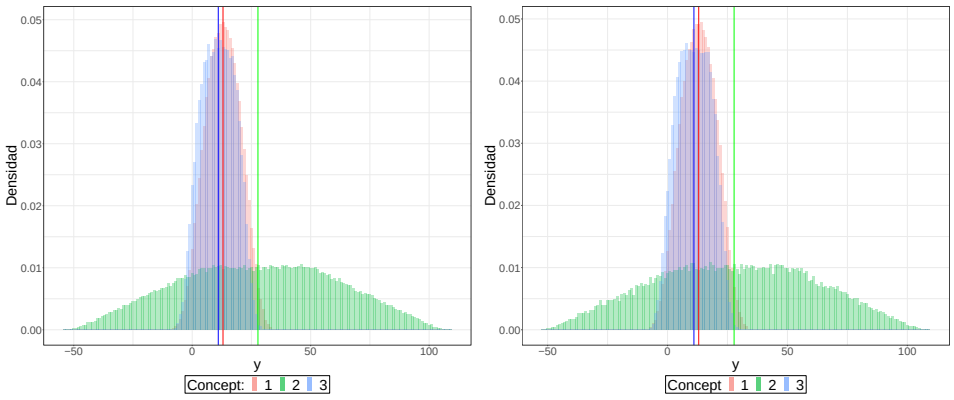
$$\begin{aligned} \mathbb{E}[y(\mathbf{x})] =& 3\,\mathbb{E}[x_1] + 22\,\mathbb{E}[x_2] - 8\,(\mathbb{E}[x_3] - 0.5) + 2\,\mathbb{E}[x_4] - \\ & 5\,\mathbb{E}[x_5] - E[\epsilon] = 11\,, \\ Var[y(\mathbf{x})] =& 3^2\,Var[x_1] + 22^2\,Var[x_2] + 8^2\,Var[x_3] + 2^2\,Var[x_4] + \\ & 5^2\,Var[x_5] + Var[\epsilon] = 598/12 \approx 49.8333\,. \end{aligned} \tag{4.6}$$

Puede comprobarse que cuadran perfectamente con los resultados de tomar la media y varianza muestral para y. Resulta entonces evidente cómo se producen cambios en los

[1] Resultados triviales para la distribución uniforme. En el caso general, tomando el intervalo $[a, b]$ se tiene:

$$\text{media} = \frac{a+b}{2}\,, \qquad \text{varianza} = \frac{(b-a)^2}{12}\,.$$

parámetros de la distribución de probabilidad $p(y \mid \mathbf{x})$ (sin cambios en $p(\mathbf{x})$), lo que implica *real concept drift* y por tanto es necesario tratarlo. Podemos visualizarlo a través de los histogramas para la variable y, de forma que tras normalizar los resultados y estableciendo una distinción entre *concepts* obtenemos la Figura 4.1.



(a) (b)

Figura 4.1: Histogramas para la variable y en el caso del Dataset II (a) y el Dataset III (b). Con lineas verticales de colores indicamos el valor promedio para cada definición de *concept*.

Notar la similitud de resultados y se aprecia cómo el cambio secuencial entre los diferentes *concepts* afecta a la distribución subyacente de la variable objetivo. Podría detectarse el *concept drift* mediante test estadísticos que establecieran la diferencia entre las distribuciones, como puede ser el Test de Kolmogorov-Smirnoff[2]. No obstante, para hacer una detección explícita del *concept drift* haremos uso de los métodos y algoritmos expuestos anteriormente.

En este caso no resulta necesario realizar un tratamiento previo de los datos y, además, es importante fijarse en que el hecho de que la dependencia de la variable objetivo y con las variables independientes $\mathbf{x}$ sea lineal (más ruido) nos facilita mucho la obtención del modelo predictivo. Basta considerar un simple modelo de regresión lineal múltiple[21, 22, 23], que desarrollaremos a continuación.

4.1. Regresión Lineal

La regresión lineal es probablemente el modelo estadístico más tratado y a pesar de su simpleza tiene gran aplicabilidad dentro del mundo de los modelos predictivos. Establece que la relación de dependencia entre la variable dependiente unidimensional, $y \in \mathbb{R}$ y las variables independientes, $\mathbf{x} = (x_1, ..., x_p)^T \in \mathbb{R}^p$, es lineal, incorporando un término $\epsilon \sim N(0, \sigma)$ que da cuenta del ruido aleatorio de las observaciones. El caso con $p = 1$ se conoce como regresión lineal simple, mientras que si $p > 1$ tenemos una regresión lineal múltiple.

[2]Se trata de una prueba no paramétrica que determina la bondad de ajuste de dos distribuciones de probabilidad entre sí. En este caso permite establecer que la diferencia entre distribuciones de probabilidad y por ende la diferencia de *concepts*.

La dependencia entre variables se expresa como

$$y = \beta_0 + \beta_1\, x_1 + ... + \beta_p\, x_p + \epsilon\,, \tag{4.7}$$

donde $\boldsymbol{\beta} = (\beta_0, \,, \beta_1, ..., \beta_p)^T$ son los parámetros que especifican cada una de las posibles hipótesis $h_{\boldsymbol{\beta}}(\mathbf{x}) \in \mathcal{H}$. Una vez fijado el valor de $\boldsymbol{\beta}$ (y elegida la hipótesis), podremos expresar la predicción para una nueva instancia $\mathbf{x}_j = (x_{j1}, ..., x_{jp})^T$ como

$$\hat{y}_j = h_{\boldsymbol{\beta}}(\mathbf{x}_j) = \beta_0 + \beta_1\, x_{j1} + ... + \beta_p\, x_{jp}\,. \tag{4.8}$$

Para obtener el valor de $\boldsymbol{\beta}$ hacemos uso de una muestra de tamaño n, es decir, nuestro conjunto de entrenamiento es $\mathcal{S} = \{(\mathbf{x}_1,\, y_1), .., (\mathbf{x}_n,\, y_n)\}$, de forma que

$$y_i = \beta_0 + \beta_1\, x_{i1} + ... + \beta_p\, x_{ip} + \epsilon_i\,, \qquad i = 1, ..., n\,. \tag{4.9}$$

Es común agrupar las n ecuaciones tomando una representación matricial

$$\mathbf{y} = \mathbf{X}\,\boldsymbol{\beta} + \boldsymbol{\epsilon}\,, \tag{4.10}$$

donde

$$\mathbf{y} = \begin{pmatrix} y_1 \\ y_2 \\ ... \\ y_n \end{pmatrix}\,, \quad \mathbf{X} = \begin{pmatrix} 1 & x_{11} & ... & x_{1p} \\ 1 & x_{21} & ... & x_{2p} \\ ... & ... & ... & ... \\ 1 & x_{n1} & ... & x_{np} \end{pmatrix}\,, \quad \boldsymbol{\beta} = \begin{pmatrix} \beta_0 \\ \beta_1 \\ ... \\ \beta_p \end{pmatrix}\,, \quad \boldsymbol{\epsilon} = \begin{pmatrix} \epsilon_1 \\ \epsilon_2 \\ ... \\ \epsilon_n \end{pmatrix}\,. \tag{4.11}$$

Existen diferentes métodos para obtener $\boldsymbol{\beta}$ y con ello ajustar el modelo a la información proporcionada por el conjunto de entrenamiento. Sin embargo, la forma más común de proceder es realizar una estimación por medio del criterio de mínimos cuadrados ordinario. Así, buscamos minimizar la pérdida cuadrática cometida, es decir, dado

$$g(\boldsymbol{\beta}) = \sum_{i=1}^{n} L(y_i,\, \hat{y}_i) = \sum_{i=1}^{n} (y_i - \hat{y}_i)^2 = ||\mathbf{y} - \mathbf{X}\,\boldsymbol{\beta}||^2 = (\mathbf{y} - \mathbf{X}\,\boldsymbol{\beta})^T (\mathbf{y} - \mathbf{X}\,\boldsymbol{\beta}) \tag{4.12}$$

tomamos como parámetros del modelo

$$\hat{\boldsymbol{\beta}} = \arg\min \big(g(\boldsymbol{\beta})\big)\,. \tag{4.13}$$

Para hallar $\hat{\boldsymbol{\beta}}$ diferenciamos esta expresión e igualamos a $\mathbf{0}$

$$\frac{\partial g(\boldsymbol{\beta})}{\partial \boldsymbol{\beta}} = -2\,\mathbf{X}^T\mathbf{y} + 2\,\mathbf{X}^T\mathbf{X}\,\hat{\boldsymbol{\beta}} = \mathbf{0}\,. \tag{4.14}$$

Por consiguiente

$$\hat{\boldsymbol{\beta}} = (\mathbf{X}^T\mathbf{X})^{-1}\mathbf{X}^T\mathbf{y} \qquad \text{si existe } (\mathbf{X}^T\mathbf{X})^{-1} \tag{4.15}$$

y la predicción del modelo lineal para nuestra instancia $\mathbf{x}_j$ se expresa

$$\hat{y}_j = h_{\hat{\boldsymbol{\beta}}}(\mathbf{x}_j) = \hat{\beta}_0 + \hat{\beta}_1\, x_{j1} + ... + \hat{\beta}_p\, x_{jp}\,. \tag{4.16}$$

Estos resultados pueden usarse para definir una hipótesis que se adapte correctamente al primer *concept*. Entonces, una vez ocurra el *concept drift* la implementación de alguno de los mecanismos de detección para monitorizar el error precuencial indicará la necesidad de modificar nuestra hipótesis y en este caso procederemos a realizar un re-entrenamiento. Es decir, estamos considerando una adaptación **activa** en entornos con *concept drift*. También puede realizarse una adaptación **pasiva**, dada la fácil implementación de un algoritmo incremental basado en la regresión lineal. Se conoce como algoritmo de Widrow-Hoff y coincide con la aplicación de técnicas de Descenso de Gradiente sobre el modelo de regresión clásico, de forma que los parámetros $\boldsymbol{\beta}$ se actualizan en cada etapa.

El Descenso de Gradiente o *Gradient Descent*[22, 23, 3] es un algoritmo de optimización de primer orden para encontrar el mínimo de una función objetivo diferenciable $F(\mathbf{x})$. Se basa en la observación de que si la función $F(\mathbf{x})$ es diferenciable en la vecindad del punto $\mathbf{a}$, entonces $F(\mathbf{x})$ decrece de forma más rápida al desplazarse desde $\mathbf{a}$ en dirección opuesta a su gradiente, es decir $-\nabla F(\mathbf{a})$. Por consiguiente, si

$$\mathbf{a}_{n+1} = \mathbf{a}_n - \eta \nabla F(\mathbf{a}_n) \tag{4.17}$$

para un paso o tasa de aprendizaje, $\eta \in \mathbb{R}^+$, lo suficientemente pequeña, se cumple la condición $F(\mathbf{a}_n) \geq F(\mathbf{a}_{n+1})$. De esta forma, para llegar al mínimo, se comienza tomando $\mathbf{x}_0$ como nuestro punto inicial y consideramos la secuencia $\mathbf{x}_0$, $\mathbf{x}_1$, $\mathbf{x}_2$, ... tal que

$$\mathbf{x}_{n+1} = \mathbf{x}_n - \eta_n \nabla F(\mathbf{x}_n), \qquad n \geq 0\,. \tag{4.18}$$

Tenemos así una secuencia monótona tal que $F(\mathbf{x}_0) \geq F(\mathbf{x}_1) \geq F(\mathbf{x}_2) \geq ...$ y esperamos que $\{\mathbf{x}_n\}$ converja al mínimo local buscado. Notar la dependencia en n de η_n, indicando la posibilidad de variar la tasa de aprendizaje con cada iteración.

En el caso que nos atañe, tratando un modelo de regresión lineal, la función objetivo a minimizar es la función de pérdida cuadrática, que en la etapa t toma la forma

$$L(y_t, \hat{y}_t) = (y_t - \hat{y}_t)^2\,. \tag{4.19}$$

Notar que a diferencia de $g(\boldsymbol{\beta})$, en este caso solo aparece una única etiqueta y predicción. La técnica se conoce entonces como Descenso de Gradiente *Estocástico* y comenzamos definiendo un valor inicial $\boldsymbol{\beta}_0$. Considerando las nuevas tuplas que van llegando $\mathcal{S}_t = \{(\mathbf{x}_t,\, y_t)\}$, con $t > 0$, buscamos adaptar los parámetros $\boldsymbol{\beta}$ del modelo de regresión lineal utilizando el gradiente de esta función, evaluado en dicha tupla

$$\begin{aligned} \nabla L(y_t, \hat{y}_t) = \left(\frac{\partial L(y,\hat{y})}{\partial \beta_0}, \frac{\partial L(y,\hat{y})}{\partial \beta_1}, ..., \frac{\partial L(y,\hat{y})}{\partial \beta_p} \right)^T \Bigg|_{(\mathbf{x}_t,\, y_t)} = \\ = 2\big((\hat{y}_t - y_t)\,,\, (\hat{y}_t - y_t)\, x_{t1}, ...,\, (\hat{y}_t - y_t)\, x_{tp} \big)^T\,. \end{aligned} \tag{4.20}$$

Una vez establecida la tasa de aprendizaje η (en nuestro caso hemos optado por tomar η constante e igual a 0.005) tenemos

$$\boldsymbol{\beta}_t = \boldsymbol{\beta}_{t-1} - \eta \nabla L(y_t, \hat{y}_t)\,, \qquad \text{con } t > 0\,. \tag{4.21}$$

Podemos ver la implementación en forma de algoritmo en el Apéndice A.

4.2. Abrupt Concept Drift (Dataset II)

Aplicamos estos desarrollos al Dataset II con *concept drift* abrupto.

4.2.1. Adaptación activa

En este caso definiremos un modelo predictivo inicial y solo será actualizado en caso de detectarse *concept drift*. Para ello, puesto que disponemos del Dataset I, podemos consideralo como nuestro conjunto de entrenamiento inicial y emplearlo para crear un modelo que se adapte al primer *concept*. A continuación se muestra un resumen del modelo de regresión.

	Estimación	Std. Error	t value	Pr $(>\|t\|)$
β_0	-9.995022	0.008125	-1230.2	$< 2 \cdot 10^{-16}$
β_1	9.9955523	0.007060	1415.8	$< 2 \cdot 10^{-16}$
β_2	0.9944626	0.007064	140.8	$< 2 \cdot 10^{-16}$
β_3	19.999440	0.007052	2835.8	$< 2 \cdot 10^{-16}$
β_4	9.990476	0.007059	1415.3	$< 2 \cdot 10^{-16}$
β_5	5.005923	0.007062	708.9	$< 2 \cdot 10^{-16}$

Tabla 4.1: Cuadro resumen de los parámetros del modelo.

Los parámetros de (4.1) se encuentran perfectamente dentro del intervalo de confianza de las estimaciones. Así mismo, Pr $(>|t|)$ es menor que $2 \cdot 10^{-16}$ en todos los casos y podemos rechazar que los parámetros sean nulos con el 95 % de confianza. Por otro lado el $p-$valor es también inferior a $2 \cdot 10^{-16}$, indicando que el modelo se ajusta correctamente a los datos y el valor $R^2 = 0.9813$ indica que somos capaces de explicar un 98 % de la variabilidad de y en el conjunto de entrenamiento. Además, representamos el plot cuantil-cuantil en la Figura 4.2 y muestra como la hipótesis de errores/residuos normales se cumple perfectamente.

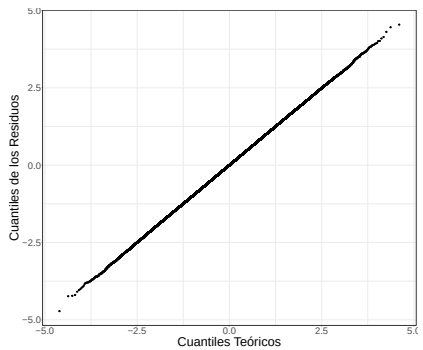


Figura 4.2: Plot cuantil-cuantil de los residuos frente a los valores teóricos (normales).

Utilizamos el modelo así obtenido para empezar a computar las predicciones sobre el Dataset II. Para ello consideraremos que en cada etapa t se tiene una única tupla $\mathcal{S}_t = \{(\mathbf{x}_t, y_t)\}$ con su correspondiente predicción $\hat{y}_t$. De esta forma hasta la aparición de *concept drift* en la instancia 75000 el modelo ofrece buenas predicciones, como permite apreciar la Figura 4.3. Observamos los buenos resultados del modelo en el primer *concept* y cómo el *drift* afecta a la capacidad predictiva del modelo.

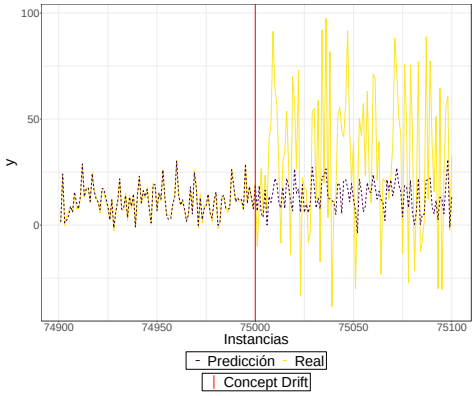


Figura 4.3: Comparativa entre valores reales y predicciones.

Para poder actuar al respecto y adaptar nuestro modelo al nuevo *concept* cuanto antes necesitamos conocer el error cometido con cada predicción. Para ello tomamos un enfoque precuencial usando una función de pérdida cuadrática (2.1) y consideraremos además diferentes *fading factors* (3.11) para ver cómo pueden afectar a la detección del *concept drift*. Nos remitimos a la Figura 4.4, donde el error en el primer *concept* es mínimo y una vez ocurre el *drift* notamos cómo aumenta, más rápidamente cuanto menor es el valor de α, pues menos afectan las predicciones anteriores, más precisas. En el caso de $\alpha = 1$, es decir, dando igual importancia a todas las predicciones anteriores, la variabilidad del error es menor, lo que puede llegar dificultar la detección. No obstante, dada la simplicidad del problema tratado, resulta posible indicar el cambio de *concept* por simple inspección visual.

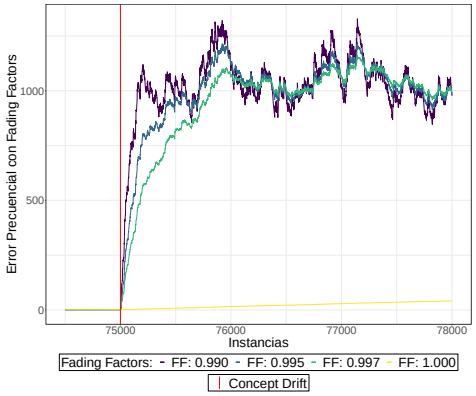
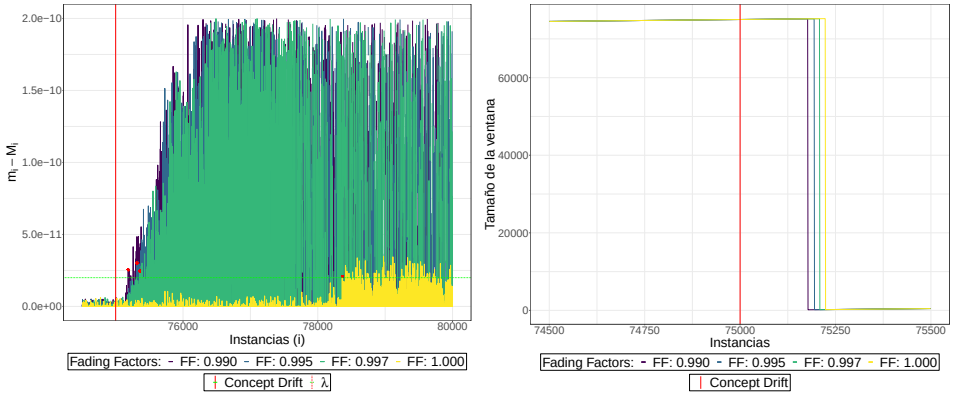


Figura 4.4: Error precuencial para diferentes *fading factors*.

Este error entra como *input* en los diferentes métodos de detección explícita, y al tratar problemas de regresión haremos uso del Test de Page-Hinkley y de ADWIN, implementando ambos a modo de comparación.

Para el Test de Page-Hinkley monitorizamos la variable $m_t - M_t$, como puede verse en la Figura 4.5a. En todos los casos se ha establecido un idéntico valor del parámetro que controla la magnitud de cambios permitidos, $\delta = 10^{-11}$ y con puntos rojos indicamos las instancias en que se detecta el *concept drift*. Notar cómo la elección del valor de λ puede ocasionar tanto falsas alarmas como retrasos en la detección del *concept drift* si no se hace correctamente, especialmente notable en el caso de $\alpha = 1$.

ADWIN permite mostrar la evolución en el tamaño de las ventanas, como se aprecia en la Figura 4.5b (se ha limitado la representación a la cercanía del *concept drift* pues el crecimiento previo es totalmente lineal al ser instancias de un mismo *concept*).



(a) (b)

Figura 4.5: Aplicación del Test de Page Hinkley (a) y del algoritmo ADWIN (b).

En la Tabla 4.2 aparecen los puntos en los que se detectó *concept drift* según cada algoritmo y con los diferentes *fading factors*.

Fading Factor (α)	Punto de detección (PH Test)	Punto de detección (ADWIN)
0.990	75186	75178 (160)
0.995	75316	75195 (177)
0.997	75358	75209 (185)
1.000	78363	75224 (195)

Tabla 4.2: Instancias en que se detectó *concept drift*. Para ADWIN la detección se indica cuando comienza a decrecer el tamaño de la ventana; entre paréntesis se indica el tamaño una vez detectado el cambio.

Notar que todos los puntos son posteriores a la instancia 75000 (que marca la verdadera ocurrencia del *concept drift*), lo que indica la presencia de un retardo en la detección inevitable. Sin embargo, un menor valor de α permite una detección más temprana del *drift*, tanto más cuanto menor sea α. Así mismo, se aprecia cómo ADWIN ofrece los mejores

resultados y además permite definir una ventana con la que elegir aquellos datos más relevantes para re-entrenar nuestro modelo y adaptarlo al nuevo *concept* (Page-Hinkley solo indica el cambio). Para todo α, la ventana solo incluye instancias posteriores a la instancia 75000, es decir, pertenecientes al nuevo *concept* y es aquí donde entra la adaptación activa, redefiniendo nuestro modelo de regresión lineal.
Por evitar repeticiones innecesarias y dada la detección más temprana, tomaremos la ventana dada en el caso de $\alpha = 0.990$ y el resumen del modelo resultante se muestra en la Tabla 4.3.

	Estimación	Std. Error	t value	Pr $(>\|t\|)$
β_0	0.71619	0.26313	2.722	0.00724
β_1	-0.03163	0.23828	-0.133	0.89458
β_2	-0.94677	0.25774	-3.673	0.00033
β_3	99.55415	0.23691	420.212	$< 2 \cdot 10^{-16}$
β_4	9.88866	0.23591	41.917	$< 2 \cdot 10^{-16}$
β_5	-54.33132	0.24858	-218.568	$< 2 \cdot 10^{-16}$

Tabla 4.3: Cuadro resumen de los parámetros del nuevo modelo.

Notar cómo en este caso los resultados no son tan buenos cómo los de nuestro primer modelo. Esto es debido a la menor cantidad de datos empleados (160 vs 300000) aunque podrían mejorarse los resultados considerando re-entrenamientos o actualizaciones de $\boldsymbol{\beta}$ posteriores. No obstante, al ser un problema sencillo resulta inncesario. Es más, el plot cuantil-cuantil de la Figura 4.6 muestra como la hipótesis de residuos normales se cumple perfectamente. Un $p-$value menor que $2 \cdot 10^{-16}$ y $R^2 = 0.9994$ indica una correcta adaptación al nuevo *concept*.

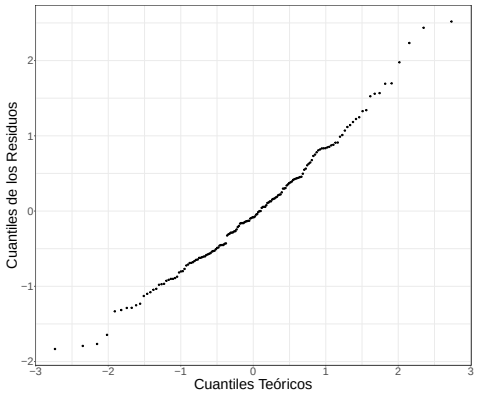
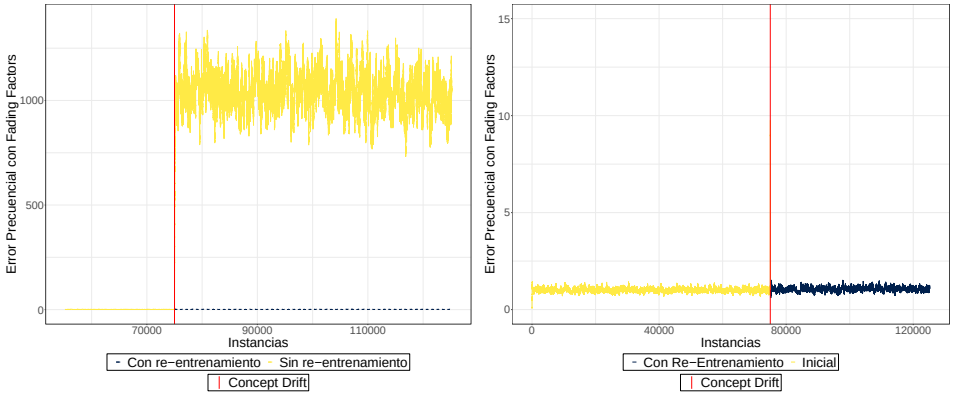


Figura 4.6: Plot cuantil-cuantil para el nuevo modelo de regresión lineal.

Aplicando este nuevo modelo sobre las tuplas entrantes puede observarse como recuperamos los bajos valores del error predictivo. En la Figura 4.7a mostramos una comparativa del error precuencial sin adaptación y con adaptación via re-entrenamiento. Por otro lado, en la Figura 4.7b aparece la curva de error que nuestro algoritmo adaptativo mostraría en la práctica (limitando los valores del eje de ordenadas para facilitar la visualización). Es decir, tras unas pocas etapas con error muy elevado dado el *concept drift*,

la detección y actuación ante el mismo permite recuperar buenos resultados predictivos. Por comodidad se ha representado solamente el caso con $\alpha = 0.990$.



(a) (b)

Figura 4.7: (a) Comparación de errores precuenciales con $\alpha = 0.990$. (b) Representación gráfica del error precuencial con $\alpha = 0.990$ tras la adaptación al nuevo *concept*.

En la cercanía de la instancia 150000 tenemos otro cambio abrupto de *concept*, pero consideramos que no merece la pena discutirlo con la profundidad con que se trató el primero, puesto que el procedimiento a seguir es análogo. Por tanto, obviamos el tratamiento de este *drift* así como el de los siguientes y mostramos directamente los resultados de la adaptación en la Figura 4.8. De nuevo se ha elegido $\alpha = 0.990$, limitando el eje de ordenadas.

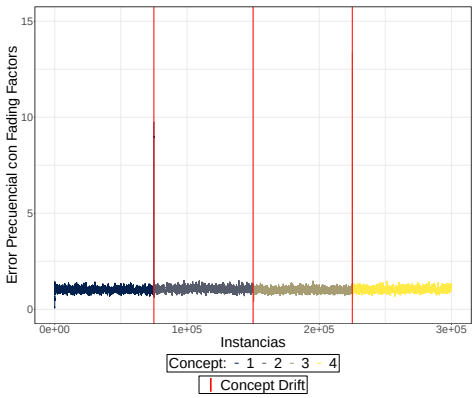


Figura 4.8: Error precuencial ($\alpha = 0.990$) resultante de la adaptación a los sucesivos cambios de *concept*. Recordemos que el último *concept* es idéntico al segundo.

4.2.2. Adaptación pasiva

En este caso emplearemos la adaptación incremental de la regresión lineal, de modo que cada nueva tupla $\mathcal{S}_t = \{(\mathbf{x}_t, y_t)\}$ se usará para evaluar la predicción del modelo y actualizarlo en concordancia, sin hacer posterior uso de ella (no almacenamos nada). Además, tomaremos las primeras 1000 instancias para realizar una estimación inicial de los parámetros $\boldsymbol{\beta}_0$.

Puede verse así, en la cercanía de la instancia donde ocurre el *drift*, cómo el modelo muestra su capacidad adptativa (Figura 4.9). Antes del *concept drift* el modelo realiza predicciones correctas y una vez ocurrido, se tarda un cierto número de etapas en recuperar la precisión.

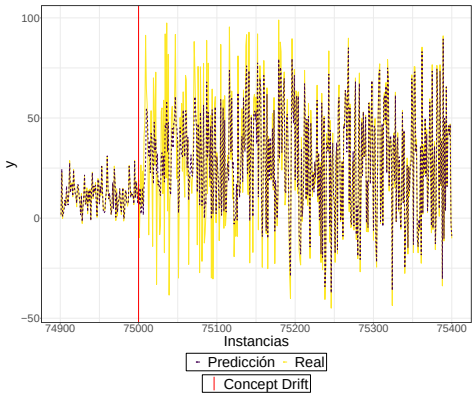
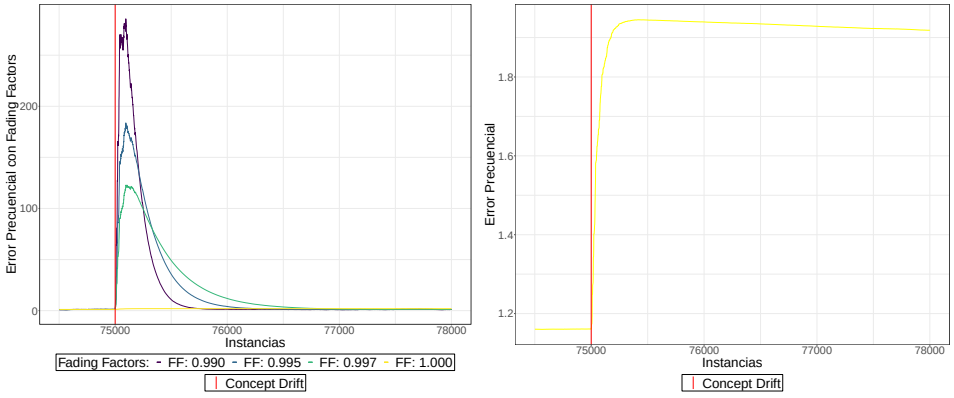


Figura 4.9: Predicciones frente a valores reales.

Tenemos entonces un modelo capaz de adaptarse al nuevo *concept* sin necesidad de haber detectado explícitamente el *concept drift*. Esto queda además reflejado en el error, calculado de forma precuencial y considerando de nuevo diferentes *fading factors*, como puede apreciarse en la Figura 4.10. Vuelve a suceder que, cuanto menor es el factor α, es decir, cuanta menor importancia se le da a los valores anteriores, más rápido aumenta el error una vez se produce el *drift* y mejor se aprecia; es más, para poder visualizar el caso $\alpha = 1$ necesitamos representarlo por separado.



(a) (b)

Figura 4.10: (a) Errores precuenciales para diferentes *fading factors*. (b) Error precuencial con $\alpha = 1$ por separado para apreciarlo mejor.

Además, como se procedió siguiendo el enfoque activo, mostramos la adaptación a lo largo de los diferentes *concepts* en la Figura 4.11. También se ha usado $\alpha = 0.990$ y se ha limitado el eje de ordenadas.

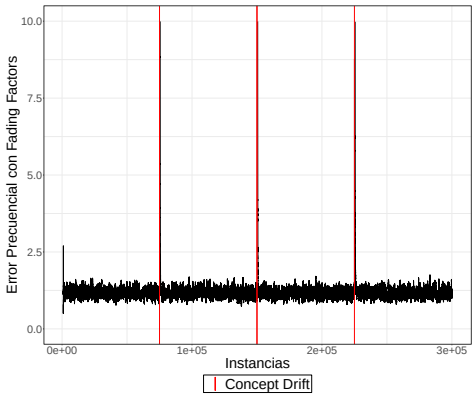
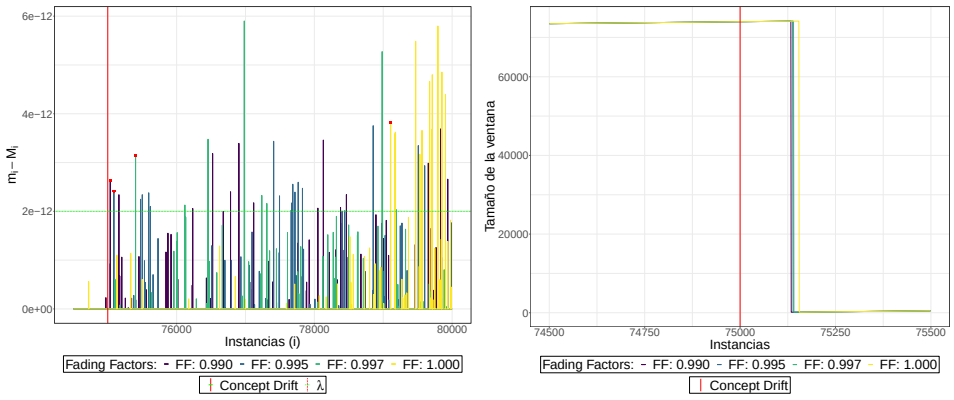


Figura 4.11: Error precuencial ($\alpha = 0.990$) resultante de la adaptación a los sucesivos cambios de *concept*.

Al tratarse de una adaptación pasiva no es necesario realizar una detección explícita del *concept drift*. No obstante, resulta una práctica interesante y comenzamos usando el Test de Page-Hinkley, cuyos resultados al monitorizar la variable $m_t - M_t$ pueden verse en la Figura 4.12a, con $\delta = 1.5 \cdot 10^{-11}$. También aplicamos el método ADWIN, mostrando la evolución de los tamaños de las ventanas en la Figura 4.12b.



(a) (b)

Figura 4.12: Aplicación del Test de Page Hinkley (a) y del algoritmo ADWIN (b).

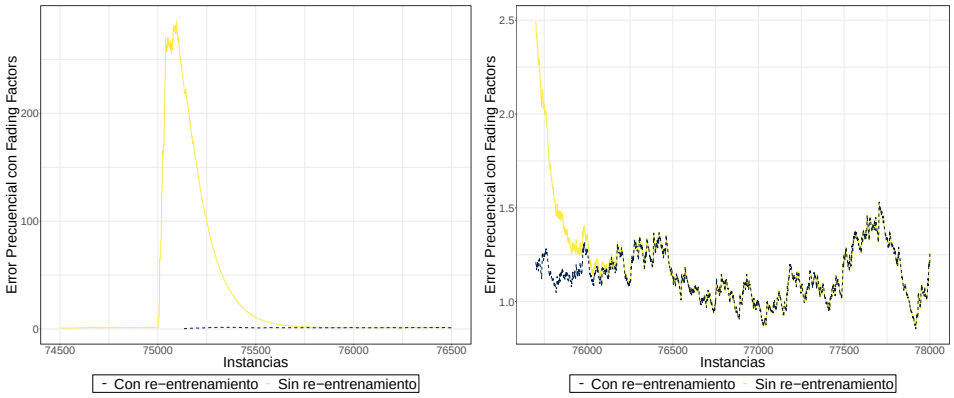
Un resumen de los puntos en los que se detectó la presencia de *concept drift* según los diferentes algoritmos y con los diferentes *fading factors* se muestra en la Tabla 4.4.

Fading Factor (α)	Punto de detección (PH Test)	Punto de detección (ADWIN)
0.990	75036	75134 (126)
0.995	75089	75139 (131)
0.997	75404	75141 (134)
1.000	79108	75155 (145)

Tabla 4.4: Instantes de detección según los diferentes métodos. Entre paréntesis mostramos los tamaños de la ventana tras la detección de *drift* (en caso de almacenar información).

Como en el caso activo, el aumento de α para el Test de Page-Hinkley conlleva una detección más tardía; también se aprecia con ADWIN aunque la magnitud del retardo es menor.

No tiene sentido usar ADWIN para definir el tamaño de las ventanas con las que llevar a cabo un re-entrenamiento (hemos especificado que no almacenamos las instancias) pero se ha hecho con motivos ilustrativos. Así, tomamos la primera ventana y comparamos el error obtenido de forma precuencial usando $\alpha = 0.990$, mostrando los resultados obtenidos en la Figura 4.13. Apreciamos entonces como el error tras realizar un re-entrenamiento al detectar *concept drift* acaba por coincidir con el obtenido sin el re-entrenamiento, al cabo de varias etapas. Esto esta totalmente de acuerdo con lo expresado en la Figura 4.9, remarcando la falta de necesidad de re-entrenar.



(a) (b)

Figura 4.13: Comparación de errores precuenciales con y sin re-entrenamiento (a) y vemos cómo terminan por igualarse tras pocas instancias (b).

Hemos visto cómo tratar el caso abrupto mediante una adaptación tanto activa como pasiva. Puesto que una vez ocurre el *drift* solo hay presentes instancias del nuevo *concept*, un re-entrenamiento del modelo con objetivo de adaptarlo cuanto antes resulta lo más adecuado, al ser directamente implementable, y menos costoso computacionalmente.

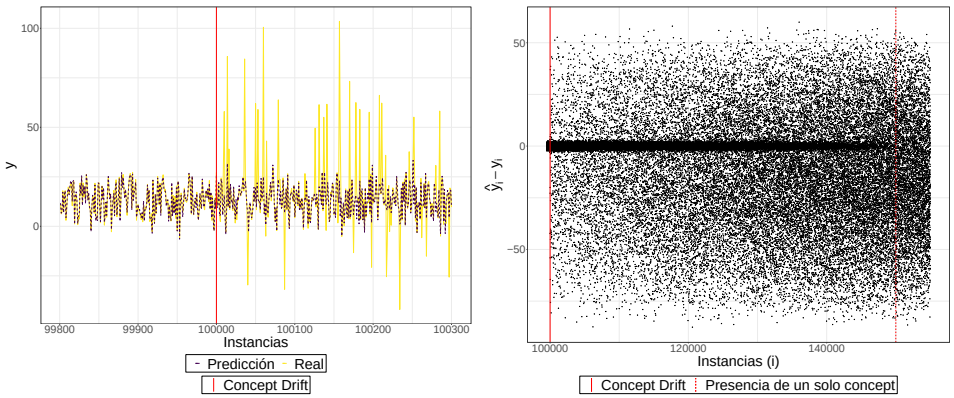
4.3. Gradual Concept Drift (Dataset III)

Una vez tratado el caso de *concept drift* abrupto, realizamos un tratamiento con cambios más graduales, en el Dataset III.

4.3.1. Adaptación Activa

De nuevo comenzamos considerando una adaptación activa y, puesto que el *concept* inicial coincide con el definido en el Dataset I, procederemos como en el caso anterior reutilizando ese modelo de regresión lineal. De esta forma, podemos comprobar cómo el *concept drift* afecta a las predicciones, en la Figura 4.14a. También resulta interesante computar la diferencia entre predicciones y valores reales, como muestra la Figura 4.14b. Se observa el carácter gradual ya que al coexistir ambos *concepts*, las instancias pertenecientes al primero ofrecen buenas

predicciones, mientras que aquellas pertenecientes al segundo presentan grandes desviaciones. Tras la instancia 150000, al existir un único *concept* solo están presentes estas últimas.



(a) (b)

Figura 4.14: Aplicación del Test de Page Hinkley (a) y del algoritmo ADWIN (b).

Dada la mezcla de *concepts* no se aprecia un incremento en el error tan notable como en el caso abrupto de la Figura 4.4. Efectivamente, para ver esto volvemos a calcular el error precuencial con diferentes *fading factors*, representado en la Figura 4.15. Encontramos como el error no tiende a estabilizarse hasta que solo hay presente un único *concept*.

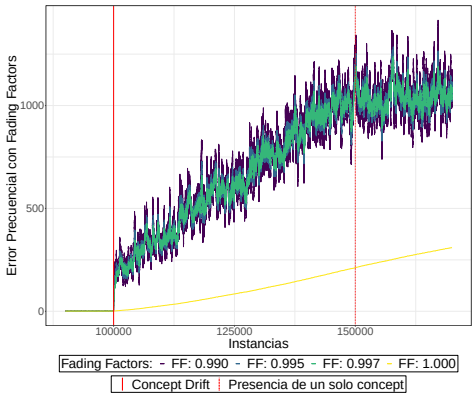
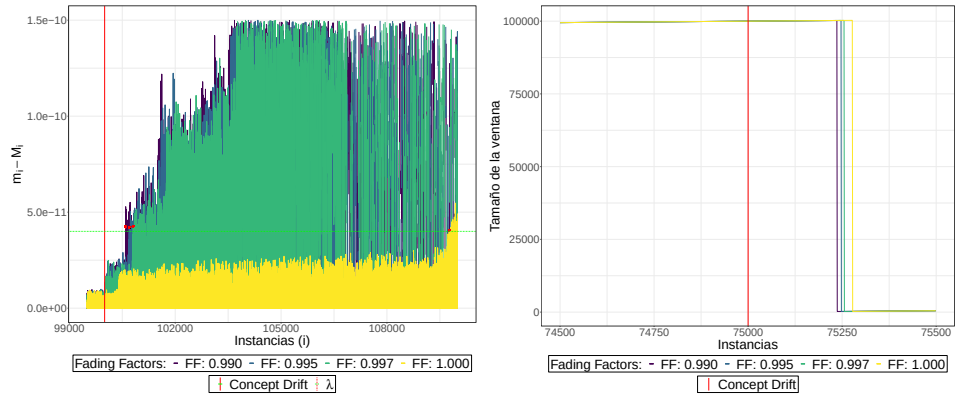


Figura 4.15: Error precuencial con diferentes *fading factors*.

Los métodos de detección explícita de *concept drift* entran ahora en juego para llevar a cabo al adaptación del modelo. En el caso del Test de *Page-Hinkley*, se muestran los resultados de la detección en la Figura 4.16a, con $\delta = 10^{-12}$ y vuelve a quedar patente la necesidad de establecer un valor de λ adecuado para una correcta detección del *concept drift*. Por otro lado, empleamos el algoritmo ADWIN sobre los diferentes errores precuenciales para detectar el *drift* y mostramos la evolución del tamaño de la ventana en Figura 4.16b. De nuevo, con un menor valor de α encontramos que el cambio en el *concept* se detecta con anterioridad.
Una comparativa de los puntos de detección según cada método puede verse en la Tabla 4.5. Una vez más ADWIN ofrece los mejores resultados para la detección de *concept drift* y además permite tener una ventana con instancias recientes para re-entrenar el modelo.



(a) (b)

Figura 4.16: Aplicación del Test de Page Hinkley (a) y del algoritmo ADWIN (b).

Fading Factor (α)	Punto de detección (PH Test)	Punto de detección (ADWIN)
0.990	100592	100237 (225)
0.995	100697	100248 (233)
0.997	100808	100257 (234)
1.000	109764	100278 (243)

Tabla 4.5: Instantes de detección según los diferentes métodos. Para ADWIN, entre paréntesis mostramos los tamaños de la ventana tras la detección de *drift*.

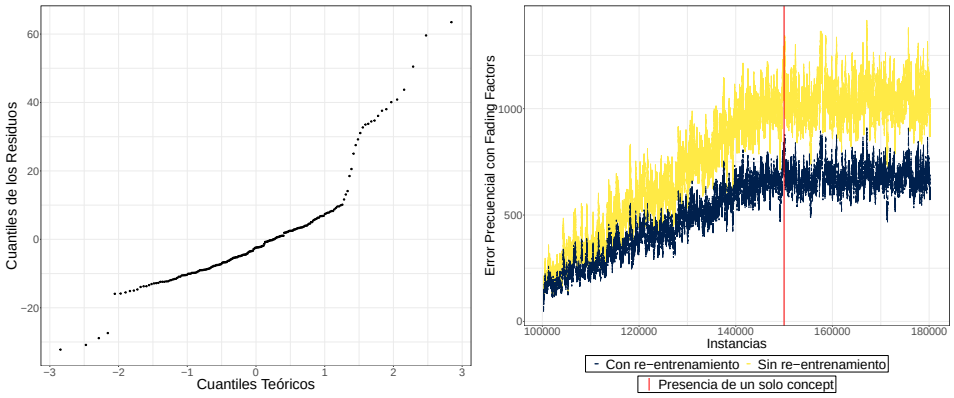
Tomando la primera ventana procedemos a reconstruir/re-entrenar nuestro modelo predictivo, obteniendo los resultados de la Tabla 4.6.

	Estimación	Std. Error	t value	Pr ($>\|t\|$)
β_0	-14.300	4.207	-3.399	0.000804
β_1	10.915	3.570	3.057	0.002514
β_2	3.138	3.298	0.951	0.342516
β_3	37.858	3.317	11.415	$< 2 \cdot 10^{-16}$
β_4	12.278	3.381	3.632	0.000351
β_5	-3.996	3.366	-1.187	0.236466

Tabla 4.6: Cuadro resumen de los parámetros del modelo.

Como puede observarse, los resultados no dan lugar a un buen modelo. Además, el valor de $R^2 = 0.443$ es muy inferior al obtenido en el caso abrupto y el plot cuantil-cuantil de la Figura 4.17a muestra cómo no se cumple la hipótesis de errores/residuos normales.

Estamos por tanto ante un modelo que no es capaz de adaptarse al nuevo *concept* y sus predicciones no serán de utilidad. Esto puede verse también computando el error precuencial (con $\alpha = 0.990$), mostrando los resultados en la Figura 4.17b. Vemos, al compararlo con el error sin re-entrenar, como en ambos casos no se da una correcta adaptación al *concept* (hay mezcla), aumentando el error hasta estabilizarse en presencia de un único *concept*.

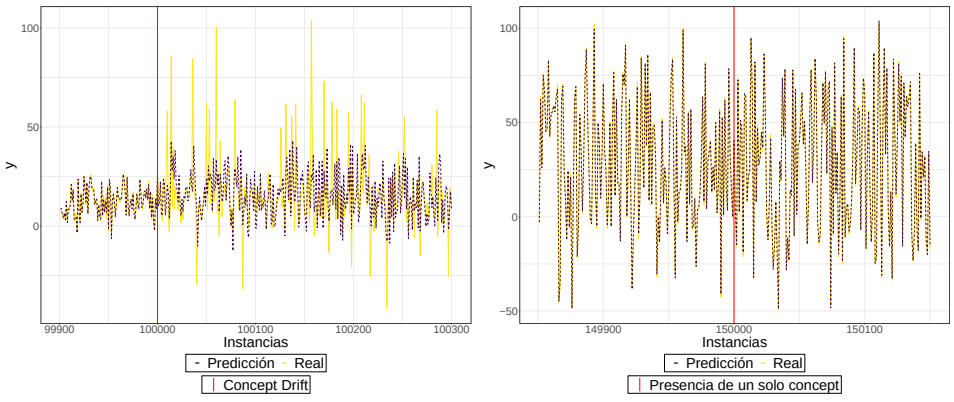


(a) (b)

Figura 4.17: (a) Plot cuantil-cuantil para el nuevo modelo de regresión lineal. (b) Comparación de errores precuenciales con y sin re-entrenamiento.

4.3.2. Adaptación Pasiva

Probemos ahora con una adaptación pasiva. Para ello volveremos a utilizar un modelo incremental, equivalente al planteado para el caso del Dataset II. Realizamos una estimación inicial de los parámetros $\boldsymbol{\beta}_0$ con las primeras 1000 instancias y las iremos actualizando a medida que van llegando nuevas. Así, se aprecia el efecto del *concept drift* sobre las predicciones en la Figura 4.18. Hemos considerado dos subfiguras para una mejor visualización del método adaptativo. Así, nada más ocurre el *drift*, el modelo tiene dificultades para adaptarse debido a la mezcla de *concepts*. No obstante, a partir de la instancia 150000, cuando solo hay un único *concept* presente, el modelo es capaz de adaptarse correctamente.



(a) (b)

Figura 4.18: Valores reales frente a predicciones al iniciarse el drift (a) y tras establecerse un solo *concept* (b).

En términos de error precuencial, podemos atender a la Figura 4.19. Una vez más, un valor de α menor permite ver la mayor variabilidad del error, comprobando así la adaptabilidad del modelo. La coexistencia de *concepts* se traduce en un elevado error hasta la presencia de un único *concept* donde tras adaptarse, el error se reduce notablemente. Es más, puede

apreciarse en la gráfica que si las líneas verticales solida y a rayas coincidiesen, recuperaríamos la Figura 4.10, donde no existe mezcla de *concepts* al ser un cambio abrupto.

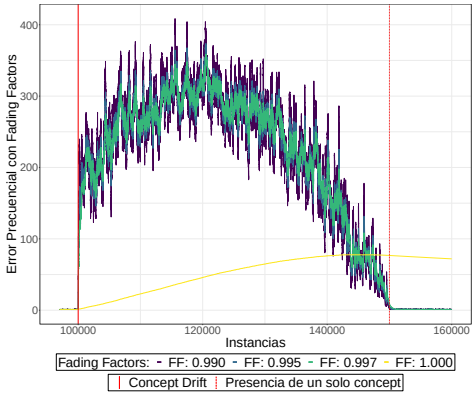


Figura 4.19: Errores precuenciales con diferentes *fading factors*.

La adaptación continua a lo largo de los diferentes *concepts* se muestra en la Figura 4.20 para un *fading factor* $\alpha = 0.990$. Como ya se indicó, la presencia de un único *concept* permite una correcta adaptación.

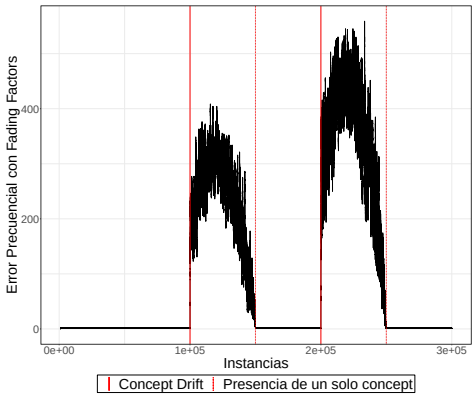
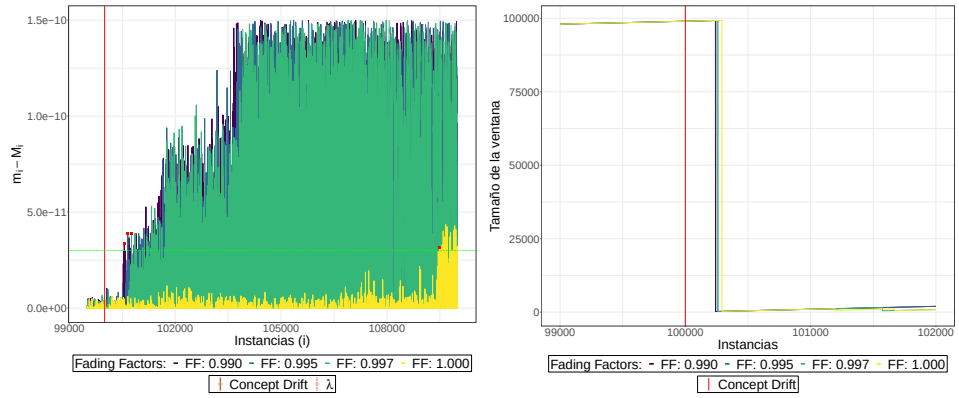


Figura 4.20: Error precuencial ($\alpha = 0.990$) resultante de la adaptación a los sucesivos cambios graduales de *concept*.

Aunque ya vimos que no es necesario en un enfoque pasivo, volvemos a hacer uso de los algoritmos de detección explícita de *drift*. Para ello comenzamos de nuevo con el Test de *Page-Hinkley*, cuyos resultados se muestran en la Figura 4.21a, con $\delta = 1.5 \cdot 10^{-11}$. El uso del algoritmo ADWIN sobre los diferentes errores precuenciales para detectar el *drift* permite mostrar la evolución del tamaño de la ventana en la Figura 4.21b.
De nuevo, con un menor valor de α encontramos que el cambio en el *concept* se detecta antes. Una comparativa de los puntos de detección según cada método puede verse en la Tabla 4.7.
Como ha ido ocurriendo con anterioridad, ADWIN permite una detección más rápida del *concept drift*. No obstante, queda patente la mayor dificultad de detectar cambios graduales, pues en comparación con los resultados de detección en el caso activo vemos como los algoritmos precisan de más instancias.



(a) (b)

Figura 4.21: Aplicación del Test de Page Hinkley (a) y del algoritmo ADWIN (b).

Fading Factor (α)	Punto de detección (PH Test)	Punto de detección (ADWIN)
0.990	100556	100241 (210)
0.995	100656	100254 (222)
0.997	100760	100262 (231)
1.000	109491	100293 (256)

Tabla 4.7: Instantes de detección según los diferentes métodos. Para ADWIN, entre paréntesis mostramos los tamaños de la ventana tras la detección de *drift*.

ADWIN nos proporciona una vez más una ventana que permite quedarnos únicamente con datos posteriores a la instancia 100000 (donde ocurre el *concept drift*). Resulta interesante volver a estimar los valores de $\boldsymbol{\beta}$ con esta ventana y proseguir con la adaptación incremental. Sucede así que el error precuencial con $\alpha = 0.990$ pronto iguala al obtenido sin re-entrenamiento (Figura 4.22), mostrado una vez más que es innecesario proceder así.

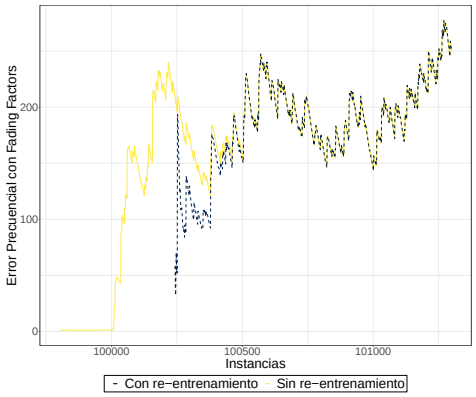


Figura 4.22: Comparación de errores precuenciales con y sin re-entrenamiento.

Podemos concluir así que una adaptación pasiva resulta mucho más adecuada para tratar con *concept drift* gradual, como ya se indicó en el capítulo anterior.

Capítulo 5

Casos Prácticos: Clasificación

Puesto que conforma el tipo de problemas más tratado y estudiado en ámbito del *concept drift* encontrar diferentes conjuntos de datos con los que empezar a experimentar no resulta complicado. Concretamente nos referimos a una extensa colección de datasets[24].
Procedemos a analizar un conjunto en particular, escogiendo así el creado usando una función sinusoidal. Tenemos entonces dos variables predictoras continuas, x_1 y x_2, uniformemente distribuidas en el intervalo $[0, 1]$ y la variable objetivo correspondiente es una variable categórica que representa dos clases (en binario 0 y 1). Definimos así un *concept* inicial en el que tomamos con $y = 1$ aquellos puntos que cumplan $x_1 < \sin(x_2)$ y con $y = 0$ en caso contrario. Una vez ocurre el *drift* se invierte la clasificación, de forma que $y = 1$ para aquellos puntos que cumplan $x_1 > \sin(x_2)$. Se establece así la creación de dos conjuntos, uno que presente *concept drift* abrupto y otro con *concept drift* gradual. En ambos casos se usarán 20000 instancias, definiendo un cambio de *concept* de forma abrupta en la instancia 10000 y un cambio gradual que se inicia en el punto 8500. En este caso hay coexistencia de *concepts* durante 2000 instancias (similar a cómo se procedió en el caso de regresión.)
También es necesario indicar que existe un equilibrio de clases a lo largo de los diferentes *concepts*, de modo que ambas tienen igual representación.

De nuevo resulta innecesario pre-procesar los datos puesto que estamos tratando casos relativamente simples. Así mismo, dado que la variable objetivo presenta solamente dos clases, haremos uso de un modelo de regresión logística[21, 22, 23].

5.1. Regresión Logística

Consideremos una variable objetivo categórica con dos clases, $y = c_1, c_2$. Resulta conveniente darles un valor numérico, de modo que realizamos una asignación binaria

$$y = c_1 \Longleftrightarrow y = 0\,,$$

$$y = c_2 \Longleftrightarrow y = 1\,.$$

Notar que podría hacerse de forma inversa y la probabilidad a posteriori de una de las clases puede escribirse, de acuerdo con el Teorema de Bayes[1], como

$$\begin{aligned} p(y=1\,|\,\mathbf{x}) \equiv p(1\,|\,\mathbf{x}) = \frac{p(\mathbf{x}\,|\,1)\,p(1)}{p(\mathbf{x})} = \frac{p(\mathbf{x}\,|\,1)\,p(1)}{p(\mathbf{x}\,|\,0)\,p(0)+p(\mathbf{x}\,|\,1)\,p(1)} = \\ = \frac{1}{1+\exp(-a)} = \sigma(a)\,, \end{aligned} \tag{5.1}$$

donde hemos definido

$$a = \log\left(\frac{p(\mathbf{x}\,|\,1)\,p(1)}{p(\mathbf{x}\,|\,0)\,p(0)}\right) \tag{5.2}$$

y $\sigma(a)$ se conoce como función logística (o *logistic sigmoid*), que aparece representada en la Figura 5.1. *Sigmoid* significa que toma forma de "S".

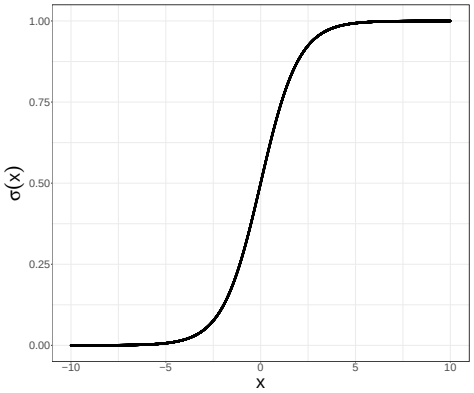


Figura 5.1: Gráfico de la función logística/*logistic sigmoid*.

Esta función satisface la siguiente propiedad[2]

$$\sigma(-a) = 1 - \sigma(a) \tag{5.3}$$

y su función inversa[2] es

$$a = \sigma^{-1}(\sigma(a)) = \log\left(\frac{\sigma(a)}{1-\sigma(a)}\right) = Logit(\sigma(a))\,, \tag{5.4}$$

también conocida como función *Logit*, de forma que $Logit(\sigma(a)) = \sigma^{-1}(\sigma(a))$. Esta expresión, como se muestra en (5.2), representa el logaritmo del cociente de probabilidades de ambas clases[2], $\log\big(p(1\,|\,\mathbf{x})/p(0\,|\,\mathbf{x})\big)$, también llamados *log-odds*. Eliminando el logaritmo tenemos los *odds*, dados por $p(1\,|\,\mathbf{x})/p(0\,|\,\mathbf{x})$.

[1]Consideremos un espacio muestral Ω generado por dos variables aleatorias X y Y. El Teorema de Bayes para dos eventos, $\{X=x\}$ y $\{Y=y\}$, establece

$$p(X=x\,|\,Y=y) = \frac{p(Y=y\,|\,X=x)\,p(X=x)}{p(Y=y)}\,.$$

[2]La demostración de este resultado puede consultarse en el Apéndice B.

Con todo lo desarrollado, la regresión logística para las dos clases considera que, dado un vector de entrada $\mathbf{x}_j = (x_{j1}, ..., x_{jp})^T$, la probabilidad de $y_j = 1$ puede escribirse haciendo actuar la función logística sobre una combinación lineal de las de las features

$$p(1 \,|\, \mathbf{x}_j) = \sigma(\beta_0 + \beta_1 \, x_{j1} + ... + \beta_p \, x_{jp}) = \frac{1}{1 + \exp\big(- (\beta_0 + \beta_1 \, x_{j1} + ... + \beta_p \, x_{jp})\big)} \tag{5.5}$$

y vemos que, como en el caso de la regresión lineal, los parámetros $\boldsymbol{\beta} = (\beta_0, \, \beta_1, ..., \beta_p)^T$ especifican las posibles hipótesis $h_{\boldsymbol{\beta}} \in \mathcal{H}$. Notar que podrían haberse llamado de otra forma pero así establecemos un paralelismo con la regresión lineal.
Así mismo, tenemos que para $y_j = 0$

$$p(0 \,|\, \mathbf{x}_j) = 1 - p(1 \,|\, \mathbf{x}_j) = \frac{\exp\big(- (\beta_0 + \beta_1 \, x_{j1} + ... + \beta_p \, x_{jp})\big)}{1 + \exp\big(- (\beta_0 + \beta_1 \, x_{j1} + ... + \beta_p \, x_{jp})\big)} \, . \tag{5.6}$$

También podemos expresar los *log-odds*

$$\log\left(\frac{p(1 \,|\, \mathbf{x}_j)}{p(0 \,|\, \mathbf{x}_j)}\right) = \log\left(\frac{p(1 \,|\, \mathbf{x}_j)}{1 - p(1 \,|\, \mathbf{x}_j)}\right) = \beta_0 + \beta_1 \, x_{j1} + ... + \beta_p \, x_{jp} \tag{5.7}$$

y los *odds*

$$\frac{p(1 \,|\, \mathbf{x}_j)}{p(0 \,|\, \mathbf{x}_j)} = \frac{p(1 \,|\, \mathbf{x}_j)}{1 - p(1 \,|\, \mathbf{x}_j)} = \exp\big(\beta_0 + \beta_1 \, x_{j1} + ... + \beta_p \, x_{jp}\big) \, . \tag{5.8}$$

De nuevo necesitamos obtener los parámetros del modelo para fijar la hipótesis. Procedemos realizando una estimación máximo verosímil, definiendo la verosimilitud para una muestra (conjunto de entrenamiento) $\mathcal{S} = \{(\mathbf{x}_1, \, y_1), .., (\mathbf{x}_n, \, y_n)\}$ como

$$\begin{aligned} \mathcal{L}(\boldsymbol{\beta}, \, y, \, \mathbf{x}) &= \prod_{i=1}^{n} p(1 \,|\, \mathbf{x}_i)^{y_i=1} p(0 \,|\, \mathbf{x}_i)^{y_i=0} = \prod_{i=1}^{n} p(1 \,|\, \mathbf{x}_i)^{y_i=1} \big(1 - p(1 \,|\, \mathbf{x}_i)\big)^{y_i=0} \\ &= \prod_{i=1}^{n} \left(\frac{1}{1 + e^{-(\beta_0 + \beta_1 \, x_{i1} + ... + \beta_p \, x_{ip})}}\right)^{y_i} \left(\frac{e^{-(\beta_0 + \beta_1 \, x_{i1} + ... + \beta_p \, x_{ip})}}{1 + e^{-(\beta_0 + \beta_1 \, x_{i1} + ... + \beta_p \, x_{ip})}}\right)^{1 - y_i} \, . \end{aligned} \tag{5.9}$$

No obstante puede resultar más sencillo trabajar con el logaritmo de esta expresión, teniendo así la *log*-verosimilitud

$$\log(\mathcal{L}) = \sum_{i=1}^{n} y_i \, (\beta_0 + \beta_{i1} \, x_{i1} + ... + \beta_p \, x_{ip}) - \sum_{i=1}^{n} \log\left(1 + e^{\beta_0 + \beta_{i1} \, x_{i1} + ... + \beta_p \, x_{ip}}\right) \, . \tag{5.10}$$

Maximizando una expresión o la otra encontramos la ecuación de verosimilitud

$$\frac{\partial \mathcal{L}(\boldsymbol{\beta}, \, y, \, \mathbf{x})}{\boldsymbol{\beta}} = \mathbf{0} \tag{5.11}$$

cuya solución[21] requiere

$$\sum_{i=1}^{n} \big(y_i - p(1 \,|\, \mathbf{x}_i)\big) = 0 \Longleftrightarrow \sum_{i=1}^{n} y_i = \sum_{i=1}^{n} \frac{1}{1 + \exp\big(- (\hat{\beta}_0 + \hat{\beta}_1 \, x_1 + ... + \hat{\beta}_p \, x_p)\big)} \, , \tag{5.12}$$

$$\sum_{i=1}^{n} \big(y_i - p(1 \,|\, \mathbf{x}_i)\big)\mathbf{x}_i = \mathbf{0} \Longleftrightarrow \sum_{i=1}^{n} y_i \, \mathbf{x}_i = \sum_{i=1}^{n} \frac{\mathbf{x}_i}{1 + \exp\big(- (\hat{\beta}_0 + \hat{\beta}_1 \, x_1 + ... + \hat{\beta}_p \, x_p)\big)} \tag{5.13}$$

y permite obtener la estimación buscada $\hat{\boldsymbol{\beta}} = (\hat{\beta}_0,\, \hat{\beta}_1, ..., \hat{\beta}_p)^T$. Las probabilidades de pertenencia a cada clase predichas por el modelo para la nueva instancia $\mathbf{x}_j = (x_{j1}, ..., x_{jp})^T$ pueden escribirse entonces como

$$p(1\,|\,\mathbf{x}_j) = \frac{1}{1+\exp\big(-(\hat{\beta}_0 + \hat{\beta}_1\, x_{j1} + ... + \hat{\beta}_p\, x_{jp})\big)}\,, \tag{5.14}$$

$$p(0\,|\,\mathbf{x}_j) = 1 - p(1\,|\,\mathbf{x}_j) = \frac{\exp\big(-(\hat{\beta}_0 + \hat{\beta}_1\, x_{j1} + ... + \hat{\beta}_p\, x_{jp})\big)}{1+\exp\big(-(\hat{\beta}_0 + \hat{\beta}_1\, x_{j1} + ... + \hat{\beta}_p\, x_{jp})\big)}\,. \tag{5.15}$$

No obstante la asociación de una clase a cada instancia suele hacerse en base a un umbral u de modo que, para $\mathbf{x}_j$, el modelo toma la hipótesis

$$\hat{y}_j = h_{\hat{\boldsymbol{\beta}},\,u}(\mathbf{x}_j) = \begin{cases} c_2\,, & \text{si } p(1\,|\,\mathbf{x}_j) \geq u\,, \\ c_1\,, & \text{si } p(1\,|\,\mathbf{x}_j) < u\,. \end{cases} \tag{5.16}$$

Una elección sencilla puede ser tomar $u = 0.5$. No obstante, la mejor forma de proceder suele ser recurrir a las curvas ROC (*Receiver Operating Characteristic*)[3, 21], que son una representación gráfica, para cada valor posible del umbral, de la Sensibilidad frente a 1−Especificidad[3]. Se presentan entonces dos formas de determinar el umbral

$$u = \begin{cases} \text{valor que maximiza la suma (Especificidad + Sensibilidad)}\,, \\ \text{valor que minimiza la suma } \big((1-\text{Especificidad})^2 + (1-\text{Sensibilidad})^2\big)\,. \end{cases} \tag{5.17}$$

Como en regresión, este modelo puede usarse para llevar a cabo una adaptación **activa**, definiendo una hipótesis que se adapte al primer *concept*. Una vez sucede el cambio de *concept* y se detecta el *drift*, aparece la necesidad de modificar la hipótesis y procederemos a realizar una reconstrucción del modelo (por re-entrenamiento).

Para realizar una adaptación **pasiva** podemos aplicar de nuevo técnicas de Descenso de Gradiente Estocástico actualizando en cada etapa t los parámetros $\boldsymbol{\beta}$. En este caso la función de pérdida empleada es

$$\begin{aligned} L(y_t,\, p(1\,|\,\mathbf{x}_t)) &= -\,y_t \log\big(p(1\,|\,\mathbf{x}_t)\big) - (1-y_t)\log\big(1-p(1\,|\,\mathbf{x}_t)\big) = \\ &= -\,y_t\, \log\left(\frac{1}{1+e^{-(\beta_0+\beta_1\, x_{t1}+...+\beta_p\, x_{tp})}}\right) - \\ &\quad (1-y_t)\log\left(\frac{e^{-(\beta_0+\beta_1\, x_{t1}+...+\beta_p\, x_{tp})}}{1+e^{-(\beta_0+\beta_1\, x_{t1}+...+\beta_p\, x_{tp})}}\right)\,. \end{aligned} \tag{5.18}$$

Tras calcular el gradiente[2] de la expresión anterior

$$\begin{aligned} \nabla L(y_t,\, p(1\,|\,\mathbf{x}_t)) &= \left(\frac{\partial L(y,\, p(1\,|\,\mathbf{x}))}{\partial \beta_0}, \frac{\partial L(y,\, p(1\,|\,\mathbf{x}))}{\partial \beta_1}, ..., \frac{\partial L(y,\, p(1\,|\,\mathbf{x}))}{\partial \beta_p}\right)^T \Bigg|_{(\mathbf{x}_t,\, y_t)} = \\ &= \big((p(1\,|\,\mathbf{x}_t) - y_t)\,,\, (p(1\,|\,\mathbf{x}_t) - y_t)\, x_{t1}, ...,\, (p(1\,|\,\mathbf{x}_t) - y_t)\, x_{tp}\big)^T\,, \end{aligned} \tag{5.19}$$

[3]Definimos en cada caso

$$\text{Sensibilidad} = \frac{\#\text{ asignaciones correctas a la clase 0}}{\#\text{ asignaciones correctas a la clase 0} \;+\; \#\text{ asignaciones a la clase 1 en vez de 0}}\,,$$

$$\text{Especificidad} = \frac{\#\text{ asignaciones correctas a la clase 1}}{\#\text{ asignaciones correctas a la clase 1} \;+\; \#\text{ asignaciones a la clase 0 en vez de 1}}\,,$$

y una vez establecida la tasa de aprendizaje η ($= 0.5$) el valor inicial $\boldsymbol{\beta}_0$ se va actualizando

$$\boldsymbol{\beta}_t = \boldsymbol{\beta}_{t-1} - \eta \nabla L(y_t,\, p(1 \,|\, \mathbf{x}_t))\,, \qquad \text{con } t > 0\,. \tag{5.20}$$

Podemos ver la implementación en forma de algoritmo en el Apéndice A.

Así mismo, y a modo de cumplimentar lo desarrollado respecto a algoritmos incrementales, podemos incluir dos nuevos[2], el Perceptrón y el algoritmo Winnow.
El primer algoritmo considera un vector de pesos $\mathbf{w}_t$ (los parámetros del modelo) inicializado con $\mathbf{w}_0$. En cada etapa t, al recibir $\mathbf{x}_t$ el algoritmo predice la clase correspondiente usando el respectivo $\mathbf{w}_t$. Una vez recibido el valor real, si resulta en una clasificación errónea, $\hat{y}_t \neq y_t$, se actualizan los pesos añadiendo al valor actual una cantidad $\eta\, y_t \mathbf{x}_t$, con $\eta > 0$.
El algoritmo Winnow es similar al perceptrón, pero en vez de considerar una actualización de los pesos aditiva, en este caso es multiplicativa. Hace uso de un vector de pesos $\mathbf{w}_t$ cuyas componentes suman la unidad ($\sum_{i=1}^{n} |w_i^t| = 1$), definido inicialmente como un vector de pesos uniforme. En cada etapa t, si la predicción $\hat{y}_t$ no coincide con el valor real y_t, cada componente i de $\mathbf{w}_t$ se actualiza multiplicándola por un factor $\exp(\eta\, y_t\, x_{ti})$ y dividiendo el resultado por una constante de normalización para asegurar que sumen 1.

5.2. Abrupt Concept Drift

Aplicamos estos desarrollos al conjunto de datos con *concept drift* abrupto.

5.2.1. Adaptación Activa

Con motivo de aplicar todo lo desarrollado con anterioridad, trataremos las primeras 7000 instancias como conjunto histórico con las que entrenar un modelo de regresión logística. Consideraremos entonces que todas las instancias de etapas posteriores t van llegando de una en una, se realiza la predicción correspondiente $\hat{y}_t$ y el valor real y_t es conocido antes de que llegue la siguiente. En caso de detección explícita de *concept drift*, adaptamos nuestro modelo en concordancia, mediante un re-entrenamiento.
Nuestro modelo de regresión logística inicial muestra un resumen de su ajuste al conjunto de entrenamiento en la Tabla 5.1.

	Estimación	Std. Error	z value	$\Pr(>\|z\|)$
β_0	3.7303	0.3259	11.45	$< 2 \cdot 10^{-16}$
β_1	-98.8275	6.2831	-15.73	$< 2 \cdot 10^{-16}$
β_2	83.9254	5.3579	15.66	$< 2 \cdot 10^{-16}$

Tabla 5.1: Resumen del modelo de regresión logística.

Los resultados muestran que todos los coeficientes son significativos y distintos de 0. Además, podemos computar las predicciones sobre nuestro conjunto de entrenamiento usado

y comparar con los valores reales, como muestra la Tabla 5.2. La gran mayoría de instancias quedan clasificadas correctamente con este modelo tan sencillo.

		Clases Reales	
		0	1
Clases Predichas	0	3433	57
	1	67	3443

Tabla 5.2: Clases predichas frente a clases reales en el conjunto de entrenamiento.

Es más, podemos calcular algunas métricas que nos dan información adicional sobre la calidad del modelo como son la Sensibilidad y la Especificidad[4]

$$\text{Sensibilidad} = \frac{3433}{3433 + 67} = 0.980857143\,, \tag{5.21}$$

$$\text{Especificidad} = \frac{3443}{3443 + 57} = 0.983714286\,. \tag{5.22}$$

Obtenemos valores muy próximos a la unidad, lo que da cuenta del buen ajuste realizado. Es necesario mencionar que sería más apropiado calcular estas métricas para evaluar la bondad del ajuste sobre un conjunto de datos de prueba diferente del usado en el entrenamiento.

Una vez comprobada la validez del modelo podemos ir computando las predicciones y, mediante una función de pérdida 0-1 (2.2), analizar el cambio de *concept*. Puesto que el *drift* ocurre en la instancia 10000, mostramos los resultados en su cercanía en la Figura 5.2. Notar como una vez cambia el *concept* el modelo empleado comete muchas malas clasificaciones.

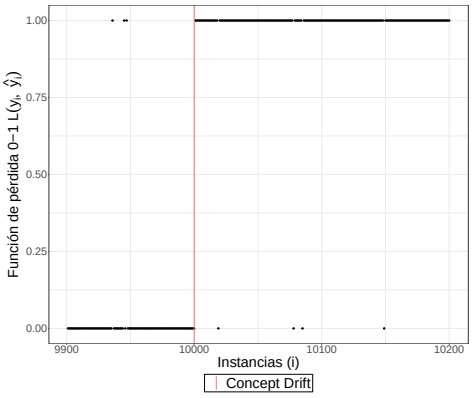


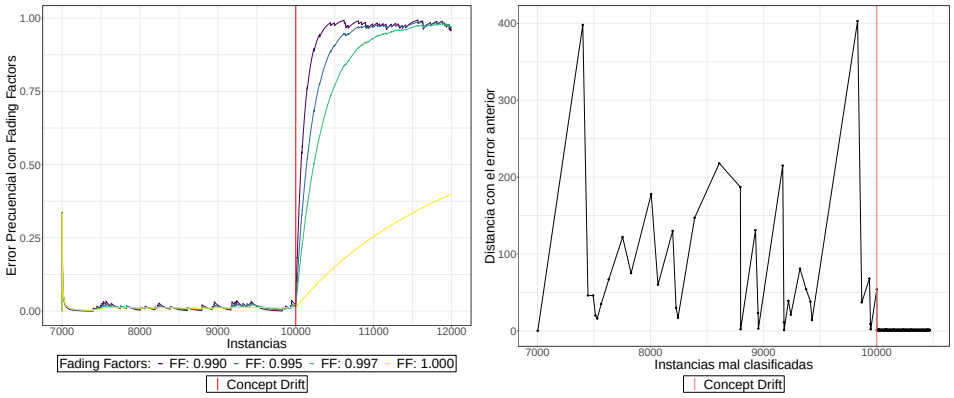
Figura 5.2: Función de pérdida 0-1.

Como ya se hizo en el caso de regresión, computamos el error para diferentes *fading factors* (3.11), representado en la Figura 5.3a y vemos cómo el incremento del error una vez cambia el *concept* es ciertamente notable.

Necesitamos que nuestro modelo sea capaz de adaptarse al nuevo *concept*. Para ello haremos uso de los métodos de detección explicita, concretamente DDM y EDDM (muy usados en clasificación y no los hemos empleado antes). El primero de ellos monitoriza la tasa de error

[4]Usadas para definir el umbral u que se usará en la asignación de clases.

(o el error precuencial con $\alpha = 1$). Por otro lado, EDDM utiliza la distancia entre dos errores consecutivos, la cual puede verse en la Figura 5.3b y una vez cambia el *concept*, notamos cómo disminuye sensiblemente.



(a) (b)

Figura 5.3: (a) Errores precuenciales con diferentes *fading factors*. (b) Distancia entre errores.

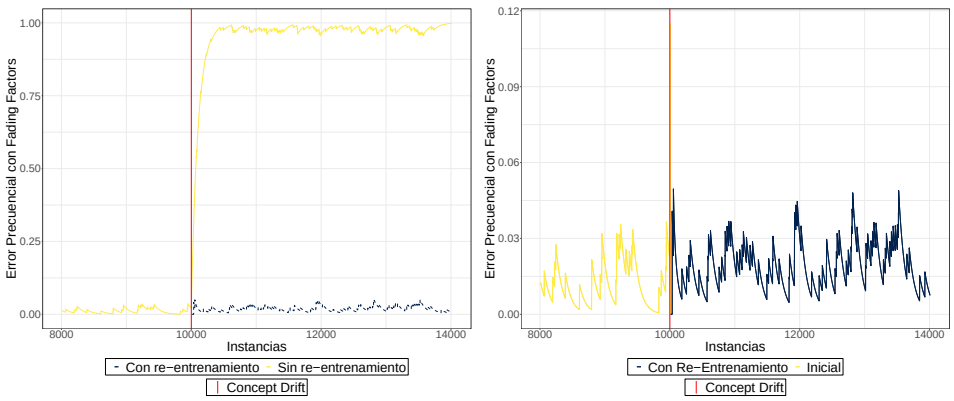
Los resultados de la detección pueden consultarse en la Tabla 5.3.

Punto de detección (DDM)	Punto de detección (EDDM)
10009 (10003)	10005 (10001)

Tabla 5.3: Instantes de detección según los diferentes métodos. Entre paréntesis aparece la instancia que marcó el nivel de alerta/*warning*.

La detección es en todo caso posterior a la instancia 10000 donde ocurre el *drift* y los algoritmos permiten disponer de una ventana para re-construir el modelo. Concretamente consideramos aquellas instancias desde que se indicó la señal de *warning* hasta que se alerta de *concept drift*. Sin embargo, dado que en ambos casos la detección es muy rápida (el modelo se adapta muy bien al primer *concept* y al cambiar enseguida se percibe el error), no hay suficientes datos almacenados en la ventana para re-entrenar el modelo. Un modo de proceder consiste entonces en re-entrenar el modelo con esa ventana, pero seguir almacenando instancias (hasta un número predefinido por ejemplo) y volver a entrenar el modelo con todos los datos disponibles. También se puede usar ADWIN, con algún computo del error precuencial anterior y usar esa ventana. No obstante, como ya se usó ADWIN en el caso de regresión, procederemos de la primera forma.

Usando la ventana proporcionada por DDM creamos primera versión del nuevo modelo y lo usaremos para predecir sobre las siguientes 50 instancias; entonces volveremos a entrenar el modelo con todas las instancias almacenadas. Procederemos así hasta analizar 500 instancias. El error precuencial con $\alpha = 0.990$ computado a lo largo de todo el proceso se muestra en la Figura 5.4, indicando una correcta adaptación.



(a) (b)

Figura 5.4: (a) Comparación entre errores precuenciales usando $\alpha = 0.990$, con y sin re-entrenamiento. (b) Errores ($\alpha = 0.990$) tras el re-entrenamiento al detectar *concet drift*.

5.2.2. Adaptación Pasiva

Por similitud con el caso regresión, usaremos una adaptación del modelo de regresión logística mediante técnicas de Descenso de Gradiente Estocástico, estimando $\boldsymbol{\beta}_0$ con las primeras 1000 instancias. Conforme nuestro modelo va computando las predicciones y actualizándose, calculamos el error de las mismas de forma precuencial, tomando una función de pérdida 0-1 para dar cuenta de las instancias mal clasificadas. Para facilitar la interpretación de los resultados mostramos únicamente la cercanía del primer *concept drift* en la Figura 5.5. Puede apreciarse cómo las malas clasificaciones aumentan de forma muy considerable una vez cambia el *concept*, con pocas clases predichas correctamente.

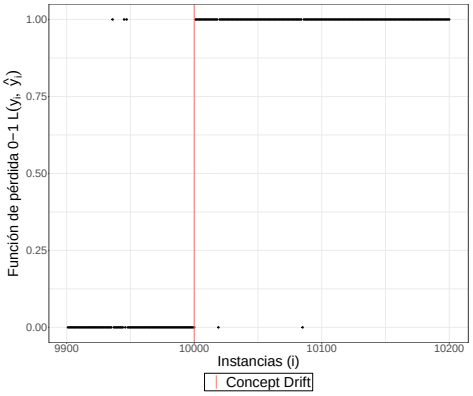
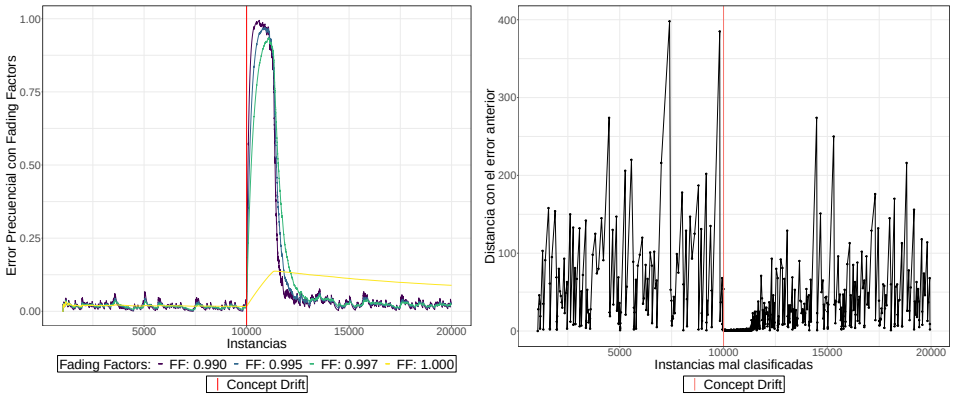


Figura 5.5: Función de pérdida 0-1.

La capacidad de adaptación del algoritmo queda patente al visualizar tanto el error precuencial, mostrado la Figura 5.6a, como la distancia entre errores de predicción consecutivos (Figura 5.6b). Notar cómo en comparación con el enfoque activo (Figura 5.3) el error no tiende a estabilizarse sino que decrece sin necesidad de implementar un mecanismo que detecte explícitamente el *concept drift*. Lo mismo ocurre con la distancia entre errores, que vuelve a aumentar tras un número de etapas.



(a) (b)

Figura 5.6: (a) Errores precuenciales con diferentes *fading factors*. (b) Distancia entre errores.

A pesar de no ser necesario, resulta interesante comprobar la detección explícita del *drift*, usando de nuevo tanto DDM como EDDM y mostramos los resultados en la Tabla 5.4.

Punto de detección (DDM)	Punto de detección (EDDM)
10023 (10010)	10039 (10016)

Tabla 5.4: Instantes de detección según los diferentes métodos. Entre paréntesis aparece la instancia que marcó el nivel de alerta/*warning*.

El punto exacto (según la creación de los datos) donde tienen lugar el *concept drift* es la instancia 10000, por lo que podemos concluir que los algoritmos implementados captan adecuadamente el cambio de *concept*. No obstante, la adaptación al nuevo *concept* es más lenta que en el caso abrupto, como puede verse al comparar la Figura 5.6a con la Figura 5.4.

5.3. Gradual Concept Drift

Una vez comprobada la aplicabilidad de todo lo desarrollado en el caso de *concept drift* abrupto, evaluamos su utilidad en la detección y adaptación con un *drift* más gradual.

5.3.1. Adaptación Activa

Usando de nuevo un conjunto de 7000 instancias para entrenar el primer modelo podemos ver el efecto del *concept drift* sobre las predicciones mediante el uso de una función de pérdida 0-1, como muestra la Figura 5.7.

Así mismo, el error precuencial se muestra en la Figura 5.8a junto con la distancia entre errores consecutivos en la Figura 5.8b. Notar cómo el modelo empeora mucho las predicciones una vez queda establecido un único *concept*. Esto es así puesto que mientras coexistan los *concepts* algunas de las instancias pertenecientes al anterior tendrán su clase objetivo predicha correctamente. Es más, si ambas líneas verticales coincidiesen, los resultados serían los mismos que en el caso abrupto, donde no hay mezcla de *concepts*.

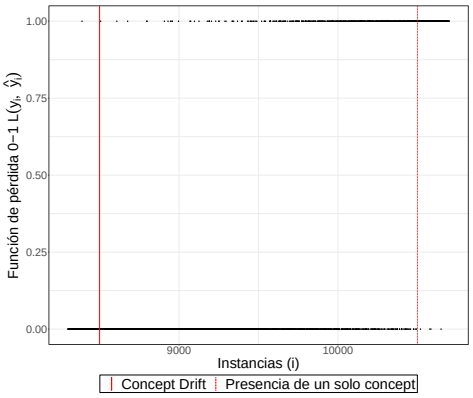
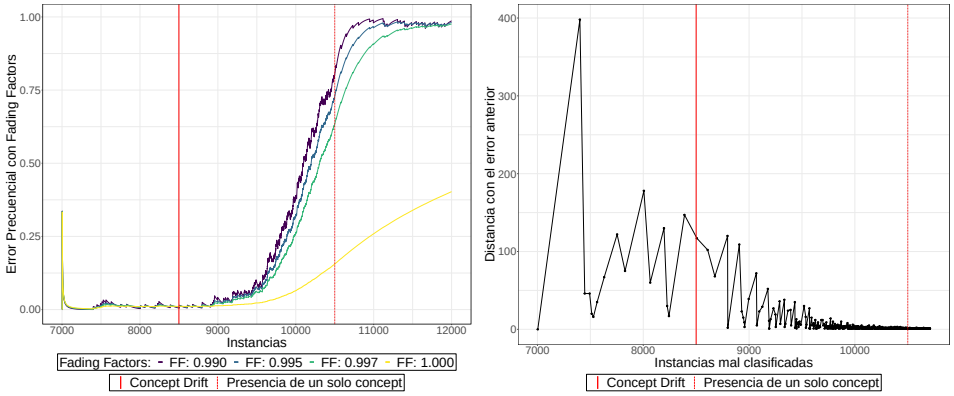


Figura 5.7: Función de pérdida 0-1.



(a) (b)

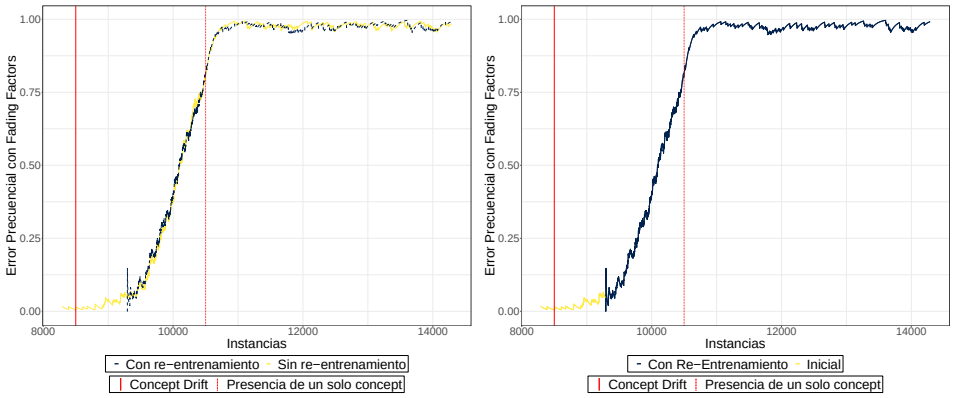
Figura 5.8: (a) Errores precuenciales con diferentes *fading factors*. (b) Distancia entre errores.

Aplicamos los algoritmos de detección explícita DDM y EDDM igual que en el caso anterior, mostrando los puntos de detección en la Tabla 5.5.

Punto de detección (DDM)	Punto de detección (EDDM)
9368 (9231)	9289 (9204)

Tabla 5.5: Instantes de detección según los diferentes métodos. Entre paréntesis aparece la instancia que marcó el nivel de alerta/*warning*.

En todo caso, la detección es posterior a la instancia 8500 donde ocurre el *drift*. Notar además, los mejores resultados de EDDM en la detección gradual, pues este era el principal objetivo buscado al proponerlo. Resulta entonces estar de acuerdo con todo lo desarrollado antes.
Estos algoritmos de detección permiten disponer de una ventana para re-construir el modelo. Hacemos uso de los resultados proporcionados por EDDM para definir una ventana (de mayor tamaño que en el caso abrupto) con la que re-entrenar el modelo y el error precuencial con $\alpha = 0.990$ se muestra en la Figura 5.9. Igual que sucedió en el caso de regresión, la coexistencia de *concepts* nos impide obtener un modelo predictivo apropiado (el error cometido en ambos casos es prácticamente el mismo). Podrían mostrarse los resultados del ajuste del modelo, pero resulta mucho más visual tomar la representación gráfica del error.



(a) (b)

Figura 5.9: (a) Comparación entre errores precuenciales usando $\alpha = 0.990$, con y sin re-entrenamiento. (b) Errores ($\alpha = 0.990$) tras el re-entrenamiento al detectar *concet drift*.

En vista de estos resultados desalentadores, procedemos con una adaptación pasiva.

5.3.2. Adaptación Pasiva

Actuamos ahora de forma incremental, actualizando los parámetros $\boldsymbol{\beta}$ en cada etapa. Con ello, el efecto del *concept drift* y la posterior adaptación del modelo puede apreciarse con la función de pérdida 0-1, como muestra la Figura 5.10.

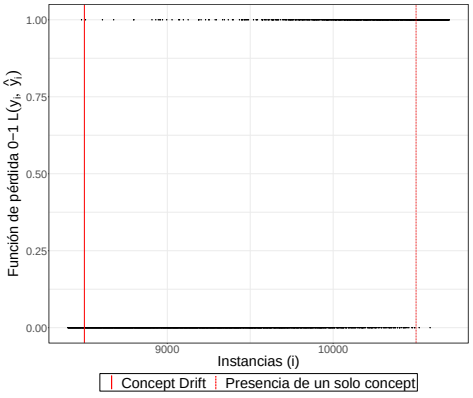
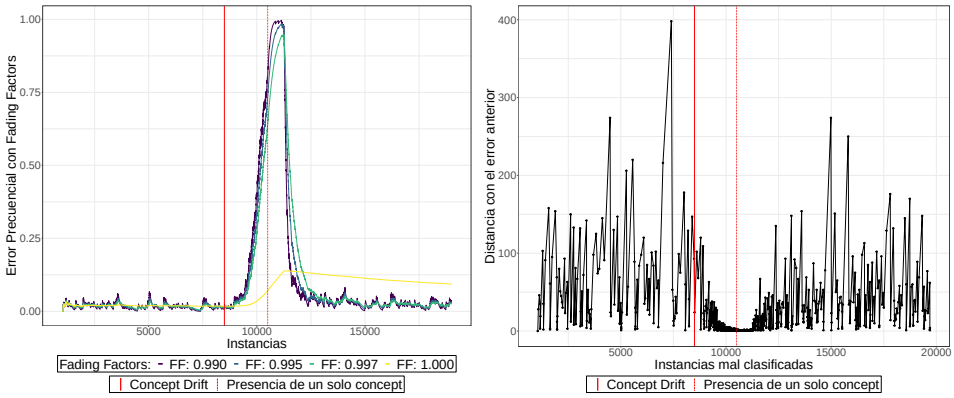


Figura 5.10: Función de pérdida 0-1.

Así mismo, computamos el error precuencial en la Figura 5.11a junto con la distancia entre dos malas clasificaciones sucesivas, mostrada en la Figura 5.11b. Podemos apreciar ciertas diferencias con respecto a la Figura 5.6, dado que en este caso se mezclan instancias pertenecientes al anterior *concept* con instancias del nuevo. Así, tanto el aumento del error como la disminución de la distancia son mucho más suaves y menos marcados que en el caso anterior. A su vez, la capacidad de adaptación del algoritmo permite recuperar un buen rendimiento transcurridas ciertas etapas.



(a) (b)

Figura 5.11: (a) Errores precuenciales con diferentes *fading factors*. (b) Distancia entre errores.

En ningún momento ha sido necesaria la detección explícita de *concept drift* para garantizar la adaptabilidad del algoritmo. No obstante resulta interesante considerarlo y por ello volvemos a aplicar los anteriores algoritmos de detección, obteniendo así los resultados de la Tabla 5.6.

Punto de detección (DDM)	Punto de detección (EDDM)
9478 (9250)	9454 (9250)

Tabla 5.6: Instantes de detección según los diferentes métodos. Entre paréntesis aparece la instancia que marcó el nivel de alerta/*warning*.

Según la forma en que se generaron los datos, el punto exacto en que ocurre el *drift* es la instancia 8500, por lo que podemos concluir que los algoritmos implementados, una vez más, son capaces de capturar adecuadamente el cambio de *concept*.

Como ya sucedió con los análisis planteados en problemas de regresión, el problema de *concept drift* gradual queda mejor resuelto siguiendo metodologías de adaptación pasiva. Sin embargo, el caso abrupto se puede tratar de ambas formas, aunque un enfoque pasivo puede conllevar cierto retraso hasta la correcta adaptación al nuevo *concept*. En el caso activo dependemos de la correcta detección del *drift*.

Capítulo 6

Conclusiones y Trabajo a Futuro

En este trabajo se ha definido el problema del aprendizaje en escenarios afectados por *concept drift*. Es decir, buscamos algoritmos capaces de trabajar en escenarios de aprendizaje online, donde la distribución subyacente de los datos no se asume estacionaria. Este problema es común en muchos campos, desde la medicina hasta la industria y para dar cuenta de la validez de los métodos desarrollados se han analizado casos de regresión y clasificación, tratando conjuntos de datos con *drifts* abruptos y graduales. En ambas situaciones, la adaptación del modelo predictivo empleado se ha realizado siguiendo un enfoque tanto **activo** como **pasivo**, lo que nos ha permitido corroborar ciertas afirmaciones previas:

- En casos de *concept drift* abrupto, se obtienen mejores resultados mediante algoritmos capaces de realizar una adaptación **activa**. Esto así puesto que el nuevo *concept* queda establecido tras muy pocas instancias y por tanto los mecanismos de detección explícita son capaces de detectar el cambio de forma efectiva y con ello reconstruir el modelo con una nueva hipótesis adecuada al *concept* presente tras un número (que esperamos sea lo suficientemente) reducido de etapas.
  No obstante sigue siendo totalmente válido tomar un enfoque pasivo, aunque pueda tardar un número mayor de etapas en lograr la adaptación.

- En situaciones con *concept drift* gradual, se suceden etapas donde se mezclan *concepts* hasta que finalmente acaba por establecerse la presencia de uno solo. Esto dificulta la detección explícita de la etapa donde se inicia el *drift* y también complica la reconstrucción del modelo pues el conjunto de datos empleado para ello contiene información del primer *concept* y por tanto irrelevante una vez se establezca el nuevo *concept*. La mejor forma de proceder consiste entonces en tomar un enfoque **pasivo**, adaptando el modelo de forma gradual mientras sucede el *drift*.

También resulta de gran importancia hacer notar la falta literatura sobre *concept drift* en lo referente a regresión, lo que dificultó los desarrollos inicialmente pero finalmente fuimos capaces de tratarlo adecuadamente.

Así mismo, podemos establecer varias posibles directivas a seguir en una continuación futura del trabajo expuesto en este documento:

- Estudio y desarrollo de nuevos algoritmos y técnicas para la adaptación al *concept drift*. Resulta esencial conocer los últimos adelantos acontecidos en un entorno de investigación siempre cambiante y en continua evolución (como nuestros datos). Esto es especialmente evidente en el caso de regresión, apenas tratado y con mucho futuro, sobre todo en entornos industriales (e.g. medidas numéricas de diferentes sensores).

- Implementación en R. Es notable la falta de librerías en R que permitan hacer un tratamiento del *concept drift* y, dada la familiaridad con este lenguaje de programación (muy usado a lo largo del Máster), hemos optado por llevar a cabo la implementación de varios de estos métodos, concretamente aquellos descritos en el trabajo. A futuro podemos considerar ampliar el repertorio e incluso crear nuestra propia librería/paquete de funciones de acceso público.

- Tratamiento de conjuntos de datos reales. Hemos analizado casos sintéticos, donde la ocurrencia del *drift* se conocía de antemano y la definición de *concept* en cada momento era relativamente simple. Los resultados han sido totalmente satisfactorios, pero nuestra atención ha estado siempre centrada en una implementación y aplicación industrial del tratamiento de *concept drift*. Es lógico por tanto que el siguiente paso consista en tratar casos reales, donde el estudio de los datos resulta mucho más complicado, necesitando casi siempre de un tratamiento previo de los mismos y una elección cuidadosa del modelo predictivo a emplear.

- Caracterización más profunda del *concept drift*. Aunque a lo largo del trabajo nos hemos centrado en la detección y adaptación en escenarios con (posible) *concept drift*, existen artículos[9] que tratan de desarrollar y/o sentar las bases para un estudio del propio *drift*. Podemos entonces profundizar en la definición de algunas de sus características:

  - "Cúando" ocurre el *concept drift*, es decir, en qué momento se produce el cambio de *concept* y cuantas etapas ocurren hasta el establecimiento de un único *concept* (si es que ocurre). Retrasos en la detección o falsas alarmas pueden complicar o incluso impedir la adaptación del algoritmo al nuevo *concept*.
  - "Cómo" es el *concept drift*, refiriéndose a la severidad del mismo. Formalmente se expresa como $\Delta = \delta\left(p_t(\mathbf{x}, y), p_{t+1}(\mathbf{x}, y)\right)$, con δ una función (no negativa) que permite medir la discrepancia entre ambas distribuciones y t indica la etapa en que ocurre el *drift*. Así, cuanto mayor el valor de Δ, más severo es el *concept drift* y más se diferencian los *concepts*.

Capítulo 7

Bibliografía

[1] Debi Prasanna Acharjya and M. Kalaiselvi Geetha. Internet of things: Novel advances and envisioned applications. 2017.

[2] M. Mohri, A. Rostamizadeh, and A. Talwalkar. *Foundations of Machine Learning*. Adaptive Computation and Machine Learning series. MIT Press, 2012. URL: https://books.google.es/books?id=maz6AQAAQBAJ.

[3] José Tomás Alcalá Nalvaiz y Beatriz Lacruz Casaucau. Apuntes de la asignatura de introducción a la minería de datos, 2021-2022.

[4] Charanjeet Singh and Anuj Sharma. A review of online supervised learning. *Evolving Systems*, pages 1–22, 07 2022. doi:10.1007/s12530-022-09448-y.

[5] Steven C.H. Hoi, Doyen Sahoo, Jing Lu, and Peilin Zhao. Online learning: A comprehensive survey. *Neurocomputing*, 459:249–289, 2021. URL: https://www.sciencedirect.com/science/article/pii/S0925231221006706, doi:https://doi.org/10.1016/j.neucom.2021.04.112.

[6] João Gama, Pedro Medas, Gladys Castillo, and Pedro Rodrigues. Learning with drift detection. volume 8, pages 286–295, 09 2004. doi:10.1007/978-3-540-28645-5_29.

[7] João Gama, Indrė Žliobaitė, Albert Bifet, Mykola Pechenizkiy, and Hamid Bouchachia. A survey on concept drift adaptation. *ACM Computing Surveys (CSUR)*, 46, 04 2014. doi:10.1145/2523813.

[8] Supriya Agrahari and Anil Kumar Singh. Concept drift detection in data stream mining: A literature review. *Journal of King Saud University - Computer and Information Sciences*, 2021. URL: https://www.sciencedirect.com/science/article/pii/S1319157821003062, doi:https://doi.org/10.1016/j.jksuci.2021.11.006.

[9] Jie Lu, Anjin Liu, Fan Dong, Feng Gu, João Gama, and Guangquan Zhang. Learning under concept drift: A review. *CoRR*, abs/2004.05785, 2020. URL: https://arxiv.org/abs/2004.05785, arXiv:2004.05785.

[10] Simona Micevska, Ahmed Awad, and Sherif Sakr. Sddm: an interpretable statistical concept drift detection method for data streams. *Journal of Intelligent Information Systems*, 56, 06 2021. doi:10.1007/s10844-020-00634-5.

[11] Andrés L. Suárez-Cetrulo, David Quintana, and Alejandro Cervantes. A survey on machine learning for recurring concept drifting data streams. *Expert Systems with Applications*, 213:118934, 2023. URL: https://www.sciencedirect.com/science/article/pii/S0957417422019522, doi:https://doi.org/10.1016/j.eswa.2022.118934.

[12] João Gama, Raquel Sebastião, and Pedro Pereira Rodrigues. On evaluating stream learning algorithms. *Mach. Learn.*, 90(3):317–346, mar 2013. doi:10.1007/s10994-012-5320-9.

[13] Hassan Mehmood, Panos Kostakos, Marta Cortes, Theodoros Anagnostopoulos, Susanna Pirttikangas, and Ekaterina Gilman. Concept drift adaptation techniques in distributed environment for real-world data streams. *Smart Cities*, 4(1):349–371, 2021. URL: https://www.mdpi.com/2624-6511/4/1/21, doi:10.3390/smartcities4010021.

[14] E. S. Page. Ccontinuous inspection schemes. *Biometrika*, 41(1-2):100–115, 06 1954. arXiv:https://academic.oup.com/biomet/article-pdf/41/1-2/100/1243987/41-1-2-100.pdf, doi:10.1093/biomet/41.1-2.100.

[15] Hayet Mouss, M.Djamel Mouss, Kinza Mouss, and Sefouhi Linda. Test of page-hinckley, an approach for fault detection in an agro-alimentary production system. pages 815 – 818 Vol.2, 08 2004. doi:10.1109/ASCC.2004.184970.

[16] Manuel Baena-García, José Campo-Ávila, Raúl Fidalgo-Merino, Albert Bifet, Ricard Gavald, and Rafael Morales-Bueno. Early drift detection method. 01 2006.

[17] T.M. Mitchell. *Machine Learning*. McGraw-Hill international editions - computer science series. McGraw-Hill Education, 1997. URL: https://books.google.es/books?id=xOGAngEACAAJ.

[18] Albert Bifet and Ricard Gavaldà. *Learning from Time-Changing Data with Adaptive Windowing*, pages 443–448. URL: https://epubs.siam.org/doi/abs/10.1137/1.9781611972771.42, arXiv:https://epubs.siam.org/doi/pdf/10.1137/1.9781611972771.42, doi:10.1137/1.9781611972771.42.

[19] Wassily Hoeffding. Probability inequalities for sum of bounded random variables. 1963.

[20] Chandima Nadungodage, Yuni Xia, Pranav Vaidya, Yu Chen, and John Lee. Online multi-dimensional regression analysis on concept-drifting data streams. *International Journal of Data Mining, Modelling and Management*, 6:217, 01 2014. doi:10.1504/IJDMMM.2014.065146.

[21] Norberto Corral Blanco. Apuntes de la asignatura de modelización estadística, 2021-2022.

[22] Christopher M. Bishop. *Pattern Recognition and Machine Learning*. Springer, New York, 2007.

[23] Trevor Hastie, Robert Tibshirani, and Jerome Friedman. *The Elements of Statistical Learning*. Springer Series in Statistics. Springer New York Inc., New York, NY, USA, 2001.

[24] Jesús López Lobo. Synthetic datasets for concept drift detection purposes, 2020. doi:10.7910/DVN/5OWRGB.

[25] Gregory Ditzler, Manuel Roveri, Cesare Alippi, and Robi Polikar. Learning in nonstationary environments: A survey. *IEEE Computational Intelligence Magazine*, 10:12–25, 2015.

[26] Imen Khamassi, M. Sayed Mouchaweh, Moez Hammami, and Khaled Ghédira. Self-adaptive windowing approach for handling complex concept drift. *Cognitive Computation*, 06 2015. doi:10.1007/s12559-015-9341-0.

[27] Raquel Sebastião and José Maria Fernandes. Supporting the page-hinkley test with empirical mode decomposition for change detection. In Marzena Kryszkiewicz, Annalisa Appice, Dominik Ślkezak, Henryk Rybinski, Andrzej Skowron, and Zbigniew W. Raś, editors, *Foundations of Intelligent Systems*, pages 492–498, Cham, 2017. Springer International Publishing.

[28] P. Domingos and G. Hulten. Mining High-Speed Data Streams. In Ismail Parsa, Raghu Ramakrishnan, and Sal Stolfo, editors, *Proceedings of the ACM Sixth International Conference on Knowledge Discovery and Data Mining*, pages 71–80. ACM Press, 2000.

[29] Marília Lima, Manoel Neto, Telmo Silva Filho, and Roberta A. de A. Fagundes. Learning under concept drift for regression—a systematic literature review. *IEEE Access*, 10:45410–45429, 2022. doi:10.1109/ACCESS.2022.3169785.

Lista de Figuras

Lista de Tablas

Anexos

Anexos A

Algoritmos Tratados

Algoritmos para la detección explícita de *concept drift*:

Algoritmo 1 Algoritmo para el Test de *Page-Hinkley*

Entrada: Parámetros δ y λ, errores (precuenciales) de predicción en cada etapa $e(t)$
Inicialización: sum $= 0$, $m_T = 0$, $M_T = 1$
for t desde $t = 1$ hasta detectar *concept drift* **do**
  sum $=$ sum $+ e(t)$
  $m_T = m_T + e(t) - \dfrac{\text{sum}}{t} - \delta$
  if $m_T < M_T$ **then**
    $M_T = m_T$
  end if
  if $m_T - M_T \geq \lambda$ **then**
    Devuelve: "*concept drift* a partir de la etapa t"
  end if
end for

Algoritmo 2 Algoritmo para el *Drift Detection Method*

Entrada: Valor 0 o 1 en cada etapa, $e(t)$, indicando si la predicción resulta en una correcta clasificación o no, número mínimo de errores previos $\text{mín}_{\text{error}}$, niveles α y β
Inicialización: $p_{\text{mín}} = s_{\text{mín}} = 0.5\,, \qquad p_t = 0$
for t desde $t = 1$ hasta detectar *concept drift* **do**
  $p_t = (e(t) + (t-1)\,p_t)/t\,, \qquad s_t = \sqrt{p_t \cdot (1-p_t)/t}$
  if $\sum_t e(t) > \text{mín}_{\text{error}}$ **then**
    if $(p_t + s_t) \geq (p_{\text{mín}} + \alpha \cdot s_{\text{mín}})$ **then**
      Alerta: "posible *concept drift* a partir de la etapa t"
      Se almacena este ejemplo para reconstruir el modelo en caso de *drift*
    end if
    if $(p_t + s_t) \geq (p_{\text{mín}} + \beta \cdot s_{\text{mín}})$ **then**
      Devuelve: "ha ocurrido *concept drift* en la etapa t"
      Se reconstruye el modelo con los datos almacenados
    else if $(p_t + s_t) < (p_{\text{mín}} + s_{\text{mín}})$ **then**
      $p_{\text{mín}} = p_t\,, \qquad s_{\text{mín}} = s_t$
    end if
  end if
end for

Algoritmo 3 Algoritmo para el *Early Drift Detection Method*

Entrada: Número de etapas entre los errores $d(e)$, número mínimo de errores previos $\text{mín}_{\text{error}}$, niveles α y β
Inicialización: $p_{\text{máx}} = s_{\text{máx}} = 1$
for e (que indica el número de errores) desde $e = 1$ hasta detectar *concept drift* **do**
  $p_e = \sum_e d(e)/e\,, \qquad s_e = \sqrt{\sum_e (d(e) - p_e)^2/e}$
  if $e > \text{mín}_{\text{error}}$ **then**
    if $((p_e + 2 \cdot s_e)/(p_{\text{máx}} + 2 \cdot s_{\text{máx}})) < \alpha$ **then**
      Alerta: "posible *concept drift* a partir del error e"
      Se almacena este ejemplo para reconstruir el modelo en caso de *drift*
    end if
    if $((p_e + 2 \cdot s_e)/(p_{\text{máx}} + 2 \cdot s_{\text{máx}})) < \beta$ **then**
      Devuelve: "ha ocurrido *concept drift* en la etapa que marca el error e"
      Se reconstruye el modelo con los datos almacenados
    else if $(p_t + s_t) > (p_{\text{máx}} + s_{\text{máx}})$ **then**
      $p_{\text{máx}} = p_t\,, \qquad s_{\text{máx}} = s_t$
    end if
  end if
end for

Algoritmo 4 Algoritmo para *ADaptive WINdowing*

Entrada: Ventana inicial W de tamaño n conteniendo errores (precuenciales) de predicción en cada etapa $\{e(1), ..., e(n)\}$, parámetro δ
for t desde $t = n + 1$ hasta detectar *concept drift* **do**
  $W = W \cup \{e(t)\}$
  for toda partición de W en W_0 y W_1 $(n_0 + n_1 = n)$ **do**
    $\hat{\mu}_{W_0} = \sum_{i=1}^{n_0} e(i)/n_0\,, \qquad \hat{\mu}_{W_1} = \sum_{i=n_0+1}^{n} e(i)/n_1\,, \qquad \epsilon_{\text{cut}} = \sqrt{\frac{n_0 + n_1}{2\, n_0\, n_1} \cdot \log\left(\frac{4\, n}{\delta}\right)}$
    while $|\hat{\mu}_{W_0} - \hat{\mu}_{W_1}| \geq \epsilon_{cut}$ **do**
      Indica: "posible *concept drift*"
      $W = W_1$
    end while
  end for
  Devuelve: ventana adaptada W.
end for

Algoritmos incrementales para aprendizaje online en problemas de **regresión**:

Algoritmo 5 Algoritmo incremental (Widrow-Hoff/SGD con Regresión Lineal)

Fija la tasa de aprendizaje η
Inicializa $\boldsymbol{\beta} = \boldsymbol{\beta}_0 = (\beta_0^0,\, \beta_1^0, ..., \beta_p^0)^T$
for toda etapa $t > 0$ **do**
  Toma instancia $\mathbf{x}_t$
  Calcula la predicción $\hat{y}_t = \beta_0^{t-1} + \beta_1^{t-1}\, x_{t1} + ... + \beta_p^{t-1}\, x_{tp}$
  Recibe el el valor real y_t
  Actualiza los parámetros $\boldsymbol{\beta}_t = \boldsymbol{\beta}_{t-1} - 2\eta\big(\,(\hat{y}_t - y_t)\,,\, (\hat{y}_t - y_t)\, x_{t1}, ...,\, (\hat{y}_t - y_t)\, x_{tp}\big)^T$
end for

Algoritmos incrementales para aprendizaje online en problemas de **clasificación**:

Algoritmo 6 Algoritmo incremental (SGD con Regresión Logística)

Fija la tasa de aprendizaje η
Inicializa $\boldsymbol{\beta} = \boldsymbol{\beta}_0 = (\beta_0^0,\, \beta_1^0, ..., \beta_p^0)^T$
for toda etapa $t > 0$ **do**
  Toma instancia $\mathbf{x}_t$
  Calcula la probabilidad $p(1 \,|\, \mathbf{x}_t) = \sigma(\beta_0^{t-1} + \beta_1^{t-1}\, x_{t1} + ... + \beta_p^{t-1}\, x_{tp})$
  Recibe el el valor real y_t
  Actualiza $\boldsymbol{\beta}_t = \boldsymbol{\beta}_{t-1} - \eta\big(\,(p(1 \,|\, \mathbf{x}_t) - y_t)\,,\, (p(1 \,|\, \mathbf{x}_t) - y_t)\, x_{t1}, ...,\, (p(1 \,|\, \mathbf{x}_t) - y_t)\, x_{tp}\big)^T$
end for

Algoritmo 7 Algoritmo incremental (perceptrón)

Fija la tasa de aprendizaje η
Inicializa $\mathbf{w} = \mathbf{w}_0$
for toda etapa $t > 0$ **do**
  Toma instancia $\mathbf{x}_t$
  Calcula la predicción $\hat{y}_t = \text{signo}(\mathbf{w} \cdot \mathbf{x}_t) = \begin{cases} -1 & \text{si } \mathbf{w} \cdot \mathbf{x}_t \leq 0 \\ 1 & \text{si } \mathbf{w} \cdot \mathbf{x}_t > 0 \end{cases}$
  Recibe el el valor real y_t
  if $\hat{y}_t \neq y_t$ **then**
    $\mathbf{w}_t = \mathbf{t\text{-}1} + \eta\, y_t \mathbf{x}_t$
  else
    $\mathbf{w}_t = \mathbf{w}_{t-1}$
  end if
end for

Algoritmo 8 Algoritmo incremental (Winnow)

Fija la tasa de aprendizaje η
Inicializa $\mathbf{w} = \mathbf{w}_0 = (w_1^0, ..., w_n^0)^T = (1/n, ..., 1/n)^T$
for toda etapa $t > 0$ **do**
  Toma instancia $\mathbf{x}_t$
  Calcula la predicción $\hat{y}_t = \text{signo}(\mathbf{w} \cdot \mathbf{x}_t) = \begin{cases} -1 & \text{si } \mathbf{w} \cdot \mathbf{x}_t \leq 0 \\ 1 & \text{si } \mathbf{w} \cdot \mathbf{x}_t > 0 \end{cases}$
  Recibe el el valor real y_t
  if $\hat{y}_t \neq y_t$ **then**
    for i $= 1$ **hasta** n **do**
      $w_i^t = \dfrac{w_i^{t-1} \exp(\eta\, y_t\, x_{ti})}{\sum_{k=1}^n w_k^{t-1} \exp(\eta\, y_t\, x_{tk})}$
    end for
  else
    $\mathbf{w}_t = \mathbf{w}_{t-1}$
  end if
end for

Anexos B

Regresión Logística: Demostraciones

En este apéndice demostramos la obtención de las expresiones mostradas en el Capítulo 5.

Dem: $\sigma(-a) = 1 - \sigma(a)$

$$\begin{aligned} \sigma(-a) &= \frac{1}{1+\exp(a)} = \frac{1}{1+\exp(a)}\frac{\exp(-a)}{\exp(-a)} = \frac{\exp(-a)}{1+\exp(-a)} = \\ &= 1 - \frac{1}{1+\exp(-a)} = 1 - \sigma(a)\,. \end{aligned} \tag{B.1}$$

Dem: $a = Logit\big(\sigma(a)\big)$

$$\begin{aligned} \sigma(a) = \frac{1}{1+\exp(-a)} &\iff \sigma(a)\big(1+\exp(-a)\big) = 1 \iff \\ \iff \sigma(a)\,\exp(-a) = 1 - \sigma(a) &\iff \exp(a) = \frac{\sigma(a)}{1-\sigma(a)} \iff \\ &\iff a = \log\left(\frac{\sigma(a)}{1-\sigma(a)}\right)\,. \end{aligned} \tag{B.2}$$

Dem: $a = \log(p(1\,|\,\mathbf{x})/p(0\,|\,\mathbf{x}))$

Usando el Teorema de Bayes, tenemos

$$p(\mathbf{x}\,|\,1)\,p(1) = p(1\,|\,\mathbf{x})\big(p(\mathbf{x}\,|\,0)\,p(0) + p(\mathbf{x}\,|\,1)\,p(1)\big)\,, \tag{B.3}$$

$$p(\mathbf{x}\,|\,0)\,p(0) = p(0\,|\,\mathbf{x})\big(p(\mathbf{x}\,|\,0)\,p(0) + p(\mathbf{x}\,|\,1)\,p(1)\big)\,, \tag{B.4}$$

lo que permite expresar a como

$$\begin{aligned} a &= \log\left(\frac{p(\mathbf{x}\,|\,1)\,p(1)}{p(\mathbf{x}\,|\,0)\,p(0)}\right) = \log\left(\frac{p(1\,|\,\mathbf{x})}{p(0\,|\,\mathbf{x})}\frac{p(\mathbf{x}\,|\,0)\,p(0) + p(\mathbf{x}\,|\,1)\,p(1)}{p(\mathbf{x}\,|\,0)\,p(0) + p(\mathbf{x}\,|\,1)\,p(1)}\right) = \\ &= \log\left(\frac{p(1\,|\,\mathbf{x})}{p(0\,|\,\mathbf{x})}\right)\,. \end{aligned} \tag{B.5}$$

Dem: $\nabla L(y_t,\, p(1\,|\,\mathbf{x}_t)) = \big(\,(p(1\,|\,\mathbf{x}_t) - y_t)\,,\, (p(1\,|\,\mathbf{x}_t) - y_t)\,x_{t1}, ...,\, (p(1\,|\,\mathbf{x}_t) - y_t)\,x_{tp}\big)^T$ donde $L(y_t,\, p(1\,|\,\mathbf{x}_t))$ toma la forma

$$L(y_t,\, p(1\,|\,\mathbf{x}_t)) = -y_t \log\big(p(1\,|\,\mathbf{x}_t)\big) - (1 - y_t) \log\big(1 - p(1\,|\,\mathbf{x}_t)\big)\,. \tag{B.6}$$

Definimos $\vec{x}_t = (1,\, x_{t1}, ..., x_{tp})^T$ de modo que podemos (diferenciarlo de $\mathbf{x}_t$ y) expresar más cómodamente

$$p(1\,|\,\mathbf{x}_t) = \frac{1}{1 + \exp\big(-(\beta_0 + \beta_1\, x_{t1} + ... + \beta_p\, x_{tp})\big)} = \frac{1}{1 + \exp(-\vec{x}_t^T \cdot \boldsymbol{\beta})} \tag{B.7}$$

y por tanto

$$\frac{d\,p(1\,|\,\mathbf{x}_t)}{d\boldsymbol{\beta}} = \frac{\vec{x}_t\, \exp(-\vec{x}_t^T \cdot \boldsymbol{\beta})}{\big(1 + \exp(-\vec{x}_t^T \cdot \boldsymbol{\beta})\big)^2} = \vec{x}_t\, p(1\,|\,\mathbf{x}_t)\big(1 - p(1\,|\,\mathbf{x}_t)\big)\,, \tag{B.8}$$

$$\frac{d\,\big(1 - p(1\,|\,\mathbf{x}_t)\big)}{d\boldsymbol{\beta}} = -\frac{d\,p(1\,|\,\mathbf{x}_t)}{d\boldsymbol{\beta}} = -\vec{x}_t\, p(1\,|\,\mathbf{x}_t)\big(1 - p(1\,|\,\mathbf{x}_t)\big)\,. \tag{B.9}$$

Así mismo

$$\frac{d\,\log\big(p(1\,|\,\mathbf{x}_t)\big)}{d\boldsymbol{\beta}} = \frac{1}{p(1\,|\,\mathbf{x}_t)} \frac{d\,p(1\,|\,\mathbf{x}_t)}{d\boldsymbol{\beta}} = \vec{x}_t\,\big(1 - p(1\,|\,\mathbf{x}_t)\big)\,, \tag{B.10}$$

$$\frac{d\,\log\big(1 - p(1\,|\,\mathbf{x}_t)\big)}{d\boldsymbol{\beta}} = \frac{1}{1 - p(1\,|\,\mathbf{x}_t)} \frac{d\,\big(1 - p(1\,|\,\mathbf{x}_t)\big)}{d\boldsymbol{\beta}} = -\vec{x}_t\, p(1\,|\,\mathbf{x}_t)\,. \tag{B.11}$$

Finalmente

$$\begin{aligned}
\nabla L(y_t,\, p(1\,|\,\mathbf{x}_t)) &= \frac{d\,L(y_t,\, p(1\,|\,\mathbf{x}_t))}{d\boldsymbol{\beta}} = \\
&= -y_t \frac{d\,\log\big(p(1\,|\,\mathbf{x}_t)\big)}{d\boldsymbol{\beta}} - (1 - y_t) \frac{d\,\log\big(1 - p(1\,|\,\mathbf{x}_t)\big)}{d\boldsymbol{\beta}} = \\
&= -y_t\, \vec{x}_t\,\big(1 - p(1\,|\,\mathbf{x}_t)\big) + (1 - y_t)\, \vec{x}_t\, p(1\,|\,\mathbf{x}_t) = \vec{x}_t\big(p(1\,|\,\mathbf{x}_t) - y_t\big) = \\
&= \big(\,(p(1\,|\,\mathbf{x}_t) - y_t)\,,\, (p(1\,|\,\mathbf{x}_t) - y_t)\,x_{t1}, ...,\, (p(1\,|\,\mathbf{x}_t) - y_t)\,x_{tp}\big)^T
\end{aligned} \tag{B.12}$$

Anexos C

Código Empleado

A continuación se incluye una recopilación de los diferentes códigos en R utilizados en la implementación de los diferentes métodos de detección explícita de *Concept Drift*.

C.1. Page-Hinkley Test

```
### Page-Hinkley Test ###

# Devuelve el instante (indice) en que se detecta Concept Drift
# errors como un vector
Page_Hinkley <- function(errors, delta = 0, lambda = 0){
  m <- dif <- warning <- c()
  M <- 0
  for(i in 1:length(errors)){
    m[i] <- sum(errors[1:i] - mean(errors[1:i]) - delta)
    if(m[i] < M) M <- m[i]
    dif[i] <- m[i] - M

    if(dif[i] > lambda){
      cat("Se supera lambda en la iteracion ", i)
      cat("\n")
      warning <- c(warning, i)
    }
  }
  return(list(dif = dif, warning = warning))
}
```

C.2. Drift Detection Method (DDM)

```
### Driftt Detection Method ###

# Modeliza el  error de clasificacion segun una distribucion binomial
# min errors indica el numero minimo de errores previos detectados
DDM <- function(error, window_length=1, min_errors=30, alpha=2, beta=3){
  # Valores iniciales altos para que sean facilmente reemplazables
  p_min <- s_min <- 1
  p <- s <- warning <- CD <- c()
  for(i in head(seq(1, length(error), window_length), -1)){
    p_i <- sum(error[1:(i + window_length - 1)])/(i + window_length - 1)
    s_i <- sqrt(p_i*(1-p_i)/(i + window_length - 1))

    # numero minimo de errores detectados para empezar
    if(sum(error[1:(i + window_length - 1)]) >= min_errors){
      if((p_i + s_i) >= (p_min + alpha*s_min)){
        cat("Warning en la iteracion ", i + window_length - 1)
        cat("\n")
        warning <- c(warning, i + window_length - 1)
        if((p_i + s_i) >= (p_min + beta*s_min)){
          cat("Concept Drift en la iteracion ", i + window_length - 1)
          cat("\n")
          CD <- c(CD, i + window_length - 1)
        }
      }
      else if((p_i + s_i) < (p_min + s_min)){
        p_min <- p_i
        s_min <- s_i
      }
    }
    p <- c(p, p_i)
    s <- c(s, s_i)
  }
  return(list(p = p, s = s, warning = warning, CD = CD))
}
```

C.3. Early Drift Detection Method (EDDM)

```
### Early Drift Detection Method ###

# Mejora la deteccion de Gradual Concept Drift
EDDM <- function(error, min_errors = 30, alpha = 0.95, beta = 0.90){
  # Valores inciales bajos para que sean facilmente reemplazables
  p_max <- s_max <- 0.5
  p <- s <- dist <- warning <- CD <- c()
  pos_error <- which(error == 1)
  for(i in 1:(length(pos_error)-1)){
    dist[i] <- pos_error[i+1] - pos_error[i]
    p[i] <- mean(dist)
    s[i] <- sd(dist)

    if(i >= min_errors){
      if(((p[i] + 2*s[i])/(p_max + 2*s_max)) < alpha){
        cat("Warning en la iteracion ", pos_error[i+1])
        cat("\n")
        warning <- c(warning, pos_error[i+1])
        if(((p[i] + 2*s[i])/(p_max + 2*s_max)) < beta){
          cat("Concept Drift en la iteracion ", pos_error[i+1])
          cat("\n")
          CD <- c(CD, pos_error[i+1])
        }
      }
      if((p[i] + 2*s[i]) > (p_max + 2*s_max)){
        p_max <- p[i]
        s_max <- s[i]
      }
    }
  }
  return(list(p = p, s = s, warning = warning, CD = CD))
}
```

C.4. ADaptive WINdowing

```
### ADaptive WINdowing ###

# Funcion previa para evaluar que se cumpla la condicion de
# distribuciones lo suficientemente distintas
# Devuelve booleanos:
#   0 distribuciones iguales
#   1 distribuciones distintas
subwindows <- function(window, cut, delta=0.2, type=2, min_size=1){

  if(length(window) == min_size) return(0)
  if(max(window) != min(window))
    W <- (window - min(window))/(max(window) - min(window))
  else W <- window

  W_0 <- W[1:cut]
  W_1 <- W[(cut+1):length(W)]
  mean_0 <- mean(W_0)
  mean_1 <- mean(W_1)
  m <- (length(W_0)*length(W_1))/(length(W_0)+length(W_1))
  Delta <- delta/length(W)
  eps1 <- sqrt((1/(2*m))*log(4/Delta))
  eps2 <- sqrt((2/m)*var(W)*log(2/Delta)) + (2/(3*m))*log(2/Delta)

  if(type == 1) return(abs(mean_0 - mean_1) > eps1)
  else if(type == 2) return(abs(mean_0 - mean_1) > eps2)

  # En caso de no devolver nada
  stop("Parametros mal especificados")
}
```

```
# newdata como un vector
ADWIN <- function(window, newdata, delta=0.2, type=1, dif=20,
                  min_size=1, entry_size=1){

  ventanas <- c(length(window))
  for(i in head(seq(1,length(newdata),entry_size),-1)){
    # Se incluyen los nuevos valores en la ventana
    window <- c(window, newdata[i:(i+entry_size-1)])
    cut <- min_size
    while(cut <= (length(window)-min_size)){
      cut_check <- cut
      while(subwindows(window, cut_check, delta=delta, type=type,
            min_size=min_size) & (cut_check >= min_size)){
        window <- window[-1]
        # Mantenemos siempre la ventana W_1
        cut_check <- cut_check - 1
        #cut <- min_size - 1
      }
      # Para volver a evaluar todas las divisiones en subventanas
      # Se repite el check en uno de los valores
      cut <- cut + 1
    }
    ventanas <- c(ventanas, length(window))
    print(c(i,length(window)))

    # Para indicar concept drift si la diferencia supera un valor
    #if((ventanas[i] - ventanas[i+1]) > dif){
    #  cat("Posible Concept Drift en la iteracion ", i)
    #  cat("\n")
    #}
    #print(mean(window))
  }
  return(ventanas)
}
```

```
# Version que elimina todas las componentes de W_0
# Nos quedamos con W_1
ADWIN_all <- function(window, newdata, delta=0.2, type=1, dif=20,
                      min_size=1, entry_size=1){

  ventanas <- c(length(window))
  for(i in head(seq(1,length(newdata),entry_size),-1)){
    # Se incluyen los nuevos valores en la ventana
    window <- c(window, newdata[i:(i+entry_size-1)])
    cut <- min_size
    while(cut <= (length(window)-min_size)){
      if(subwindows(window,cut,delta=delta,type=type,min_size=min_size)){
        # Nos quedamos solamente con W_1
        window <- window[(cut+1):length(window)]
        cut <- min_size - 1
      }
      # Para volver a evaluar todas las divisiones en subventanas
      # Se repite el check en uno de los valores
      cut <- cut + 1
    }
    ventanas <- c(ventanas, length(window))
    print(c(i,length(window)))

    # Para indicar concept drift si la difrencia supera un valor
    #if((ventanas[i] - ventanas[i+1]) > dif){
    #  cat("Posible Concept Drift en la iteracion ", i)
    #  cat("\n")
    #}
    #print(mean(window))
  }
  return(ventanas)
}
```