TRABAJO FIN DE MÁSTER

Máster DATCOM: Ciencia de Datos

Árboles de clasificación monotónica sobre flujos de datos.

Autor
Carlos Manuel Sequí Sánchez(alumno)

Directores
Salvador García López(tutor)

Escuela Técnica Superior de Ingenierías Informática y de Telecomunicación

—

Granada, septiembre de 2019

Árboles de clasificación monotónica sobre flujos de datos.

Autor
Carlos Manuel Sequí Sánchez

Directores
Salvador García López

Departamento de Ciencias de la Computación e I.A.

Árboles de clasificación monotónica sobre flujos de datos.

Carlos Manuel Sequí Sánchez(alumno)

Palabras clave: Monotonic, data streams, decision tree, Hoeffding tree, classification, MOA

Resumen

Con el fin de trabajar con temas de mi agrado, crear algo novedoso y aprovechar los conocimientos adquiridos a lo largo del máster, decidí aceptar la propuesta de mi tutor Salvador, y realizar un algoritmo mediante el uso de árboles de clasificación con información subyacente del problema basada en restricciones monotónicas sobre flujos de datos.

Monotonic classification trees on data streams.

Carlos Manuel Sequí Sánchez(student)

Keywords: Monotonic, data streams, decision tree, Hoeffding tree, classification, MOA

Abstract

In order to work with topics that I like, create something new and take advantage of the knowledge acquired throughout the master, I decided to accept the proposal of my tutor Salvador, and make an algorithm by using classification trees with underlying information of the problem based on monotonic restrictions on data streams.

Yo, **Carlos Manuel Sequí Sánchez**, alumno de la titulación Máster DATCOM: Ciencia de Datosde la **Escuela Técnica Superior de Ingenierías Informática y de Telecomunicación**, con DNI 20486926K, autorizo la ubicación de la siguiente copia de mi Trabajo Fin de Máster en la biblioteca del centro para que pueda ser consultada por las personas que lo deseen.

Fdo: Carlos Manuel Sequí Sánchez

Granada a 1 de septiembre de 2019.

D. **Salvador García López(tutor1)**, Profesor del Departamento de Ciencias de la Computación e I.A. de la Universidad de Granada.

Informa:

Que el presente trabajo, titulado ***Árboles de clasificación monotónica sobre flujos de datos.***, ha sido realizado bajo su supervisión por **Carlos Manuel Sequí Sánchez**, y autoriza la defensa de dicho trabajo ante el tribunal que corresponda.

Y para que conste, expide y firma el presente informe en Granada a 1 de Septiembre de 2019 .

El director:

Salvador García López

Agradecimientos

Llegado a este punto, agradezco la paciencia e interés puesto en mi aprendizaje a todos los profesores que han formado parte, durante todo este año, de poner en mis manos la semilla de conocimiento que me servirá para lanzarme al mundo profesional, así como a mi tutor Salvador García López, quien se ha encargado de ayudarme y supervisar este TFG. Agradezco a toda mi familia y, con mayor énfasis a mis padres y a mi hermano, el interés y el apoyo que me han ofrecido desde el primer momento, aunque no entiendan del todo las "letras raras" en la pantalla de mi ordenador cuando trabajo, o que no nos enseñen a "hackear" cosas. Por último, agradezco el haber prolongado el contacto en el ámbito académico con los amigos que hice durante el grado, con buena compañía todo ha sido más sencillo, ya sabéis.

Índice general

1. Capítulo 1. Introducción y objetivos **1**

2. Capítulo 2. Background en problemas **3**
2.1. Data streaming classification 3
2.1.1. Tipos de algoritmos para data stream 4
2.1.2. Aproximación y aleatorización 4
2.1.3. Ventanas de tiempo 5
2.1.4. Sampling 6
2.1.5. Sinopsis, bocetos y resúmenes 6
2.1.6. Problemas en el aprendizaje sobre data streams 7
2.1.7. Requisitos y funcionamiento de data streams. 8
2.2. Ordinal and monotonic classification 9
2.2.1. Restricciones monotónicas 10
2.2.2. Métodos de clasificación no paramétricos 12

3. Capítulo 3. Background en algoritmos de árboles de decisión **15**
3.1. Fundamentos de árboles de decisión 15
3.1.1. Terminología: 17
3.1.2. ¿Regresión o clasificación?: similitudes y diferencias. . 17
3.1.3. Ventajas e inconvenientes de los árboles de decisión . 18
3.1.4. Creación del árbol 19
3.2. Hoeffding Trees y otros algoritmos de data streaming 22
3.2.1. Hoeffding tree (VFDT) 22
3.2.2. Otros algoritmos de data streaming 24
3.3. Árboles de decisión monotónicos 27
3.3.1. Problema: 27
3.3.2. ¿Cómo creamos un árbol de decisión monotónico? . . 27
3.3.3. Método de evaluación de monotonicidad y precisión . 28

4. Capítulo 4. Propuesta **31**

5. Capítulo 5. Software desarrollado y uso **33**

6. Capítulo 6. Experimentos **35**
6.1. Framework 35
6.2. Resultados 35
6.3. Análisis 35

7. Capítulo 7. Conclusiones y trabajo futuro **37**

Índice de figuras

2.1. Resumen entre las diferencias principales entre el procesamiento estándar de una base de datos y el procesamiento de flujo de datos. [11] 4
2.2. Ciclo de clasificación para flujos de datos junto con los requisitos utilizados en cada paso de los descritos anteriormente. . 9

3.1. Ejemplo de árbol de decisión [2] 16
3.2. Partes de un árbol de decisión [3] 16
3.3. Recursive Binary Splitting in regression trees [8] 19
3.4. Recursive Binary Splitting in classification trees (classification error rate) [7] 20
3.5. Recursive Binary Splitting in classification trees (gini) [9] . . 20
3.6. Recursive Binary Splitting in classification trees (entropia) [9] 21
3.7. Cálculo del límite de Hoeffding [4] 24
3.8. Ciclo CVFDT [10] 25
3.9. Definición de W [12] 28
3.10. Definición índice de no-monotonicidad [12] 28
3.11. Definición de order-ambiguity-score [12] 29
3.12. Definición de total-ambiguity-score [12] 29

Capítulo 1

Introducción y objetivos

El objetivo principal de este documento es, primeramente, dotar al lector de los conocimientos necesarios sobre las distintas técnicas a utilizar para el desarrollo del algoritmo, con el fin de llegar a entender el propósito de este proyecto.
En este punto se tratarán los temas de clasificación con restricciones monotónicas, la clasificación sobre flujos de datos, y el background correspondiente a la clasificación mediante el uso de árboles de decisión, así como la combinación de algunas de estas técnicas entre sí, tales como la del uso de árboles de clasificación sobre flujos de datos.

Una vez hayamos dotado al lector de la contextualización del problema, el siguiente objetivo es la descripción inicial de la propuesta, seguido de la explicación del software desarrollado para tal propósito, así como su uso.

Finalmente, aparecerá en este documento una exposición de los experimentos realizados con la propuesta, además de comparaciones de sus resultados con los de los algoritmos que citaremos más adelante, con el propósito de observar si la propuesta cumple el cometido de resultar ser mejor en los aspectos que deseemos.
Acompañado de estas comparaciones y para finalizar el documento, presentaremos también una serie de conclusiones y trabajos futuros a desarrollar para continuar con esta línea de trabajo.

Capítulo 2

Background en problemas

2.1. Data streaming classification

Los sistemas tradicionales basados en el uso de memoria, entrenados de una forma fija mediante conjuntos de entrenamiento y los cuales generan modelos estáticos, no están preparados para procesar los datos altamente detallados disponibles en procesos como, por ejemplo, el continuo análisis de datos generados por los sensores de una máquina que trabaja sin descanso, lo cual crea una gran cantidad de datos que ha de ser procesada de forma rápida con el fin de generar modelos predictivos consistentes que se adapten a situaciones cambiantes y puedan reaccionar de forma rápida y eficaz a dichos cambios.

El Machine Learning extrae conocimiento en forma de modelos y patrones de unos datos de naturaleza cambiante. Hoy en día la generación de datos, gracias a las capacidades tecnológicas de las que disfrutamos, se produce a altas velocidades, tanto es así, que se pone, en cuanto a velocidad, por delante del procesamiento de dichos datos, lo cual quiere decir que generamos datos a mayor velocidad de lo que las capacidades computacionales que tenemos ahora mismo nos permiten procesarlos.
Desde este punto de vista, en estos casos conviene modelar los datos como flujos de datos transitorios en lugar de como tablas de datos persistentes.

	Databases	Data streams
Data access	Random	Sequential
Number of passes	Multiple	Single
Processing time	Unlimited	Restricted
Available memory	Unlimited	Fixed
Result	Accurate	Approximate
Distributed	No	Yes

Figura 2.1: Resumen entre las diferencias principales entre el procesamiento estándar de una base de datos y el procesamiento de flujo de datos. [11]

2.1.1. Tipos de algoritmos para data stream

Existen dos tipos distintos de algoritmos que trabajan sobre flujos de datos:

- **Insert-only model**: donde los datos entran al sistema de forma secuencial.
- **Insert-delete model**: donde los elementos que entran pueden ser eliminados o actualizados.

Desde el punto de vista de los sistemas de control de flujo de datos(DSMS), existen varios problemas que requieren técnicas de procesamiento no exactas para evaluar el flujo continuo de datos que requieren una cantidad ilimitada de memoria.

Estos algoritmos de procesamiento flujos de datos producen soluciones aproximadas dentro de un rango de error admisible para ciertas aplicaciones, con una alta probabilidad, relajando así las restricciones a la hora de obtener una solución exacta.

Los sistemas de control de flujos de datos han desarrollado un conjunto de técnicas que almacenan resúmenes de datos compactos suficientes para resolver consultas. Estas aproximaciones requieren un equilibrio entre el accuracy y la cantidad de memoria usada para almacenar los resúmenes, con una restricción adicional de tiempo de procesado de los datos.

2.1.2. Aproximación y aleatorización

Dentro del marco del data streaming, como ya hemos dicho, está permitido ofrecer respuestas aproximadas dentro de un pequeño rango de error

(ϵ), con una pequeña probabilidad de fallo (δ) para obtener respuestas con una probabilidad de que 1-δ se encuentre en el intervalo de radio ϵ.

Los algoritmos que usan estas aproximacióon y aleatorización son referidos por dichos (ϵ,δ).
la idea consiste básicamente en mapear cada espacio grande de entrada en una sinopsis pequeña.
La aproximación y randomización han sido usadas en solventar problemas como minería de reglas de asociación, items frecuentes, k-means...

2.1.3. Ventanas de tiempo

Para la realización del cómputo estadístico referente al modelo de flujos, no nos interesa el total de los datos existentes, si no los más recientes, entendiendo que son los que mejor explican la situación a la que nos enfrentamos y pudiendo, de esta forma, deshacernos de grandes cantidades de datos que no nos son útiles.

Las técnicas más simples para este tipo de tratamiento de datos, utilizan una ventana deslizante de tamaño fijo, con un funcionmiento FIFO (first in fist out).

Definimos dos tipos de ventana deslizante:

- **Basada en secuencia**: donde el tamaño de ventana queda definido por el número de observaciones del data set (tamaño fijo o variable en el tiempo).
- **Basada en marca de tiempo**: donde el tamaño de ventana está definido en términos de duración. Una ventana de este tipo de tamaño t consiste en todos los elementos cuya marca de tiempo se situa dentro del intervalo de tiempo t del actual periodo de tiempo.

El hecho de monitorizar, analizar y extraer conocimiento de flujos de datos de alta velocidad, puede hacer que existan diversos niveles de **granularidad** a la hora de almacenar los datos. Conforme más antiguos son los datos que disponemos, mayor granularidad requeriremos en la información (es decir, menor precisión). Cuanto más reciente sean los datos, el grano ha de ser más fino, ya que requerimos más precisión al tratarlos debido a que son más importantes (este es llamado el **modelo de ventana de tiempo inclinado**).

Ejemplo de algoritmo de ventana de tiempo

AdWin-ADaptive sliding WINdow: mantiene una ventana variable con respecto a los items recientemente vistos con la propiedad de que la ven-

tana tiene un tamaño maximal estadísticamente consistente con la hipótesis de que no haya habido un cambio en la media del valor dentro de la ventana. Un fragmento viejo de la ventana se desecha si hay alguna evidencia de que tiene un valor distinto al del resto de la ventana.

2.1.4. Sampling

El sampling (o muestreo) consiste en la selección del subconjunto de datos a analizar en intervalos periódicos, utilizado para calcular estadísticas del flujo (valores esperados).
Este tipo de técnicas reduce la cantidad de datos a procesar, por tanto, el coste computacional.
Como contra a su uso, podemos decir que pueden ser una fuente de errores, por ejemplo, en aplicaciones dedicadas a la detección de valores extremos o anomalías, ya que, a la hora de realizar el sampling podemos estar eliminando dichas instancias. El problema principal es obtener una muestra representativa.

Técnicas de muestreo:

- **Random sampling**: muestreo aleatorio de los datos (todas las instancias con la misma probabilidad de ser escogidas).
- **Reservoir sampling**: consiste en mantener una muestra de tamaño K de reserva. A medida que fluyen los datos, cada nuevo elemento tiene una probabilidad k/n (donde n son los datos visualizados hasta el momento) de reemplazar un antiguo dato.
- **Load shedding**: elimina secuencias del flujo de datos cuando se producen cuellos de botellas en las capacidades de procesamiento.

2.1.5. Sinopsis, bocetos y resúmenes

A continuación describimos tres métodos de compactación de información para la generación de modelos sobre los ya comentados conjuntos de datos reducidos para data streaming:

- **Sinopsis**: estructuras de datos compactas que resumen datos para su posterior consulta.
- **Data sketching**: herramienta de reducción de dimensionalidad. Usa proyecciones aleatorias de datos con cierta dimensión d a un espacio de cierto conjunto de dimensiones.

- **Data stream summary (by Cirnide and Muthukrishnan)**: usado para aproximaciones (ϵ,δ) para resolver consultas de rango, consultas puntuales y consultas innerproduct.

2.1.6. Problemas en el aprendizaje sobre data streams

El objetivo de la minería de datos es la habilidad de mantener de forma permanente un modelo de decisión preciso. Este problema requiere algoritmos que se adapten a los datos conforme esten disponibles para poder aprender de ellos. Además, los datos desactualizados han de ser olvidados para dejar de tenerlos en cuenta a la hora de crear el modelo, cosa que ha de ocurir en la presencia de información con una distribución no estacionaria presente. El aprendizaje en flujos de datos requiere por tanto algoritmos incrementales de aprendizaje que tengan en cuenta el llamado çoncept drift".

La solución a estos problemas requieren nuevas técnicas de muestreo y randomización, y nuevos algoritmos aproximados, incrementales y decrementales. Algunas propiedades deseables para algoritmos de flujos de datos:

- **Incrementalidad**
- **Aprendizaje online**
- **Tiempo constante de procesado de cada ejemplo**
- **Un solo escaneo sobre el conjunto de datos de training**
- **Tener en cuenta el concept drift**

Los algoritmos de aprendizaje incrementales y decrementales requieren una permanente actualización del modelo de decisión conforme llegan datos nuevos. Esta habilidad de actualizar el modelo mediante las propiedades de los nuevos datos es importante, pero no suficiente, ya que también es necesaria la habilidad de olvidar información anticuada para dar un giro en el aprendizaje realizado, dejando de tener en cuenta los items antiguos: decremental learning.

Evidentemente existe una balanza entre la ganancia en rendimiento ofrecido por el algoritmo y la manutención de la característica de actualización de este, lo que hace que el cómputo realizado por el algoritmo sea más complejo. Ante esta balanza, con el fin de no acrecentar el cómputo, haciendo que el algoritmo decida de forma dinámica qué información borrar y cuál no, surge la ya comentada técnica de ventana deslizante.

De forma general, es complicado asumir que, en el manejo de flujos de datos durante un largo tiempo, estemos tratando con datos acordes a una

distribución de probabilidad estacionaria. En sistemas complejos y en largos periodos de tiempo, debemos esperar cambios en la distribución de los items.

Una aproximación natural para estas tareas incrementales son los algoritmos de aprendizaje adaptativo, algoritmos incrementales que tienen en cuenta el concept drift. El concept drift en sí, se refiere al cambio de concepto que sufren los datos a la largo del tiempo, cada vez con cierta permanencia mínima. Hay algoritmos que implementan el olvido de información antigua teniendo en cuenta este cambio de concepto, lo que los hace mucho más precisos que los propios algoritmos que realizan la eliminación de información en forma de tamaño de ventana prefijado.
Con el uso de los algoritmos de detección de concept drift podemos averiguar cuando y por qué ha cambiado el comportamiento del flujo de datos.
Estos algoritmos no poseen la información de mundo cerrado de la que disponen los algoritmos convencionales para el tratamiento de datos estáticos, si no que han de ser capaces de adaptarse a un mundo abierto cambiante de datos para diferenciar entre cambio de concepto y ruido en los datos.

A la hora de evaluar los resultados en el contexto de los flujos de datos, es interesante tener en cuenta la evolución del acierto de nuestro algoritmo a lo largo del tiempo con los cambios de concepto acaecidos.

2.1.7. Requisitos y funcionamiento de data streams.

A modo de resumen, planteamos los requisitos primordiales para la clasificación de data streams de la siguiente forma:

1. Procesamiento de un ejemplo en cada instante de tiempo e inspección de este tan solo una vez.
2. Limitado uso de memoria.
3. Trabajo en un tiempo limitado.
4. Estar listo para predecir en cualquier momento.

De la misma forma, describimos el ciclo de clasificación para flujos de datos

1. El algoritmo toma el siguiente ejemplo del flujo
2. El algoritmo procesa el ejemplo actualizando sus estructuras de datos. En este punto no se ha de exceder los límites de memoria y ha de ser lo más rápido posible.
3. El algoritmo está listo para aceptar el siguiente ejemplo.

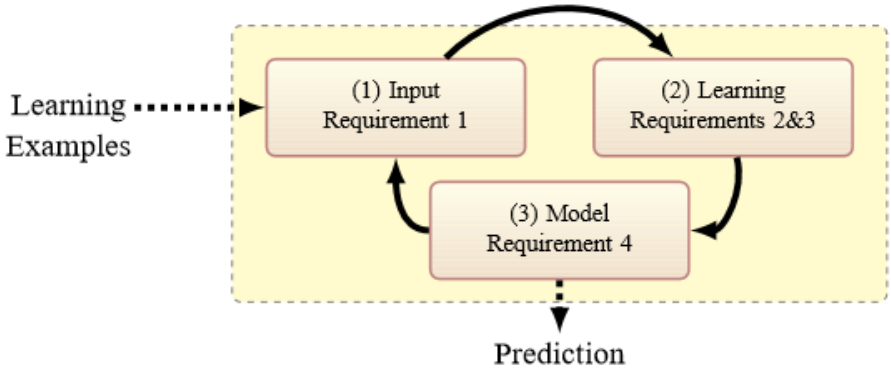

Figura 2.2: Ciclo de clasificación para flujos de datos junto con los requisitos utilizados en cada paso de los descritos anteriormente.

Para el procedimiento de evaluación de los algoritmos de aprendizaje, mientras que los modelos batch tradicionales utilizan un conjunto de datos de train y otro de test de reserva (holdout) para realizar dicha evaluación, en los algoritmos dedicados al data streaming se utiliza el método Interleaved Test-Then-Train o Prequential, mediante el cual cada una de las instancias que se reciben se utilizan como instancia de test para, posteriormente, usarla como nuevo dato de aprendizaje (train). De esta forma no es necesario mantener un conjunto de datos de reserva exclusivo para validar, haciendo que el uso de los datos disponibles sea máximo, además de que ayuda a crear una representación más visual de la evolución de la precisión del algoritmo a lo largo del tiempo.

2.2. Ordinal and monotonic classification

Comenzamos este apartado definiendo primeramente un par de conceptos básicos para adentrarnos de forma correcta en el significado de la clasificación ordinal y monotónica:

- **Principio de dominancia**: a mayor valor en atributos de una instancia, mayor será el valor de la clase a la que se asigna dicha instancia. Usando el termino de "relación de dominancia" decimos que una instancia x domina a otra instancia x' cuando cada una de las variables de entrada de x (atributos de x) son mayores o iguales que cada uno de los de x', se denota x>=x' y por tanto x tendrá asignada una etiqueta de clase mayor que x'.
- **Función monótona**: una función es monótona si x>=x' -> h(x)>=h(x').

Es decir, si x domina a x', la inferencia de clase de x ha de ser superior a la de x'.

Una vez definidos estos conceptos, podemos describir el sentido de la clasificación ordinal con restricciones monotónicas así como su diferencia con respecto a la clasificación ordinal simple:

- La **clasificación ordinal con restricciones monotónicas** maneja conocimiento subyacente del problema sobre clases ordenadas, atributos ordenados y una relación monotónica entre la evaluación de los atributos de una instancia y la asignación de esta a una clase.
- Si no hay relación de monotonía en la asociación de una clase a una instancia, pero las clases si poseen un orden, entonces se considera **clasificación ordinal** simplemente.

Con las restricciones de monotonía presentes se puede trabajar con una amplia variedad de funciones sin temor a que introduzcan más restricciones que la de monotonía: es posible hacer inferencia de la clase sobre todas las funciones monótonas.
La clasificación monotónica puede ser directa (más habitaciones, precio mayor de una casa), o inversa (más polución, precio menor de la casa).
Normalmente en problemas de clasificación monotónica reales, las restricciones monotónicas son consideradas en un subconjunto de características del dataset, no en todos los atributos.

Ejemplos de uso de monotonicidad:

- Comparación de dos compañías donde una domina sobre la otra en términos de todos los indicadores financieros. Debido a esto, la compañía dominante ha de tener una evaluación final superior a la compañía dominada. Un uso de esto, es la predicción de la calificación crediticia usada por los bancos.
- House pricing: el precio de una casa será superior cuantas más habitaciones posea, mejor sea la calidad del aire acondicionado y menor sea la polución en el ambiente.

2.2.1. Restricciones monotónicas

La motivación del uso de restricciones monotónicas viene dada por los siguientes aspectos:

- El tamaño del espacio de la hipótesis es reducido, lo que facilita el proceso de aprendizaje.

- Otras métricas además de la precisión, como la consistencia con respecto a estas restricciones, pueden ser usadas por los expertos para aceptar o rechazar el modelo. Estas técnicas de evaluación de restricciones monotónicas las veremos más adelante con el fin de poder evaluar la consistencia de estas.

Las restricciones impuestas a continuación, son restricciones con respecto a la probabilidad de distribución en la generación de datos, además de imposiciones sobre la función de perdida bajo las cuales el clasificador óptimo de Bayes es monótono.

Dominancia estocástica

El principio de dominancia no siempre se aplica en la práctica de forma tan restrictiva, por lo que hemos de hablar en términos probabilísticos a la hora de referirnos a dichas restricciones.

Decimos entonces que, siendo 'k' una de las posibles clases a tomar en el dominio por una instancia 'x', y siendo 'y' la etiqueta asignada a dicha instancia 'x', si la restricción monótona nos dice que x>=x', entonces la dominancia estocástica nos dice que P(y<=k—x)<=P(y<=k—x'). Es decir, que la probabilidad de que el valor asignado (y) a la instancia dominante (x) sea mayor que cierto valor fijado de la clase (k), es mayor que la probabilidad de que el valor asignado (y) a la instancia dominada(x') sea mayor que ese mismo cierto valor de la clase fijado.

La relación de dominancia estocástica entre distribuciones se denota así:

$$x \succeq x' \Longrightarrow P(y|x) \succeq P(y|x')$$

Donde P(y|x) y P(y|x') denotan las distribuciones condicionales de la clase en x y x'.

Clasificador monótono de Bayes

En el problema de clasificación el objetivo es encontrar el clasificador más parecido al clasificador de Bayes, es decir, esta es nuestra función objetivo. Sabiendo esto, se convierte en requisito el hecho de que este también aplique las restricciones de monotonía que hemos enunciado.

Problema: aunque la distribución de probabilidad tiene restricciones monotónicas, el clasificador de Bayes no siempre las mantiene. Para solucionar este problema y mantener la monotonía en el clasificador de Bayes, han de imponerse las siguientes restricciones a la función de perdida (L):

- L(y,k+1)-L(y,k) >= L(y+1,k+1)-L(y+1,k)
  Esta característica de la función de pérdida es necesaria en la clasificación con restricciones monotónicas, si no no tendría sentido minimizar el riesgo dentro de la clase de las funciones monótonas.
  (Demostrado en [1])

- La siguiente definición de convexidad es necesaria también para mantener la restricción de monotonía en el clasificador de Bayes:

  - Siendo L(y,k) = c(y-k) (con c(0)=0)
  - La función c(k) es convexa si, para todo k entre -(k-1) y (k-1): c(k) <= (c(k-1)+c(k+1))/2
  - El clasificador de Bayes es monótono si y solo si c(k) (que es la V-shaped loss function) es convexa.

2.2.2. Métodos de clasificación no paramétricos

Los métodos no paramétricos son así llamados porque explotan la clase de todas las funciones monótonas. Estos métodos no hacen ninguna asunción más sobre el modelo que la de las restricciones monotónicas.

Aproximación Plug-In

Pretenden **estimar la distribución condicional de la clase**. Provienen de la clasificación isotónica (monótona creciente o decreciente)

Hemos de construir un método para estimar P(y|x), sabiendo que P(y|x) posee dos ventajas:

1. La distribución condicional permite la determinación de la predicción óptima para cualquier función de pérdida.

2. La distribución condicional mide la confianza de la predicción.

Problema de la clasificación binaria y la regresión isotónica.

En la aproximación plug-in se propone usar un vector de estimadores de densidad condicional p = (p1,....,pn), el cual es una regresión isotónica del vector de etiquetas y = (y1,...,yn). Es decir, p nos da la probabilidad de que x pertenezca a cada una de las clases existentes en y.

Dicho vector p es la solución del problema: SUM((yi-pi)$\hat{2}$) sujeto a las restricciones de monotonicidad (Xi>=Xj -> pi>=pj). Por ello p minimiza el error cuadrático en el conjunto de los vectores monótonos p=(p1,..,pn) para cada x.

La elección de la función de error (función de perdida de error cuadrático) parece ser arbitraria. Puede verse que haciendo uso de otras funciones de perdida, se llega al mismo resultado.

La regresión isotónica es un problema de optimización cuadrática con restricciones lineales, por ello puede ser resuelta de forma eficiente con la mayoría de los resolutores de optimización de propósito general.

Problema multiclase.

Está basado en la regresión isotónica multiclase y, la idea es descomponer el problema de K-clases en varios problemas binarios y aplicar regresión isotónica a cada uno de los problemas. Está demostrado que la descomposición del problema de estimación de probabilidad para el caso de multiclase, siempre forma una adecuada distribución de probabilidad, es decir, que siempre son no negativos y la suma es igual a 1.

Aproximación directa

Consideramos la clasificación directa basada en la **minimización del riesgo empírico** dentro de la clase de todas las funciones monótonas.
Aunque este tipo de funciones no se pueden describir con un número finito de parámetros, la minimización del riesgo puede realizarse debido a que solo estamos interesados en valores de funciones monótonas en ciertos puntos, los incluidos en D (training set).

Una función monótona minimizando el riesgo empírico puede obtenerse resolviendo el siguiente problema de optimización:

- Minimizar: SUM(L(yi,di)).
- Teniendo en cuenta las restricciones de monotonía.
- Donde di son variables del problema (valores de la función monótona óptima en puntos de D)

El problema puede tener **otra interpretación** interesante: reetiquetar las instancias para hacer el dataset monótono de forma que las etiquetas de las instancias sean lo mas parecidas a las del conjunto original, donde esta similitud es medida en términos de la función de pérdida. Estas nuevas etiquetas serán los nuevos valores óptimos de las variables di. Este reetiquetado puede realizarse en el proceso de preprocesamiento y corresponde a la **corrección del error no paramétrico**.

Como el problema de clasificación no paramétrica se asimila al de la regresión isotónica (exceptuando que ahora se considera una salida discreta), será llamado ahora "clasificación isotónica" y su solución optima dŝerá llamada "clasificación óptima de y"

Capítulo 3

Background en algoritmos de árboles de decisión

3.1. Fundamentos de árboles de decisión

Los **árboles de decisión** [14] son un tipo de algoritmos de aprendizaje supervisado (tanto para clasificación como para regresión) utilizado en diversos ámbitos como la inteligencia artificial, las finanzas, el marketing, etc. Dado un conjunto de datos se fabrican diagramas de construcciones lógicas en forma de ramificaciones de árboles, muy similares a los sistemas de predicción basados en reglas, que sirven para representar y categorizar una serie de condiciones que ocurren de forma sucesiva, para la resolución de un problema.

Adentrándonos en los aspectos más técnicos de este tipo de modelos de predicción, cabe destacar que las variables de entrada y de salida pueden ser tanto categóricas como continuas y que divide el espacio de los predictores (variables independientes) en regiones distintas y no superpuestas, tal como veremos en la siguiente figura.

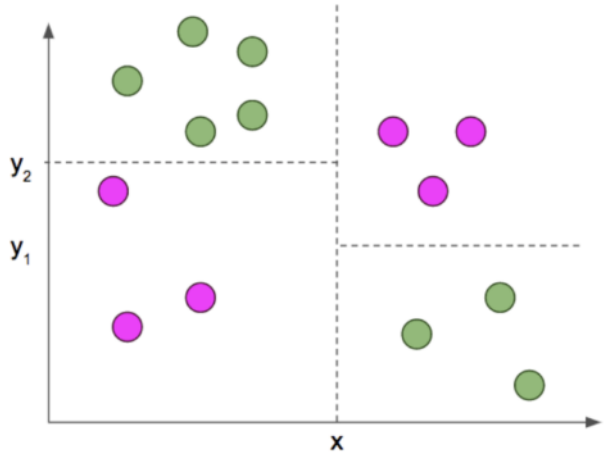

Figura 3.1: Ejemplo de árbol de decisión [2]

Estas divisiones se realizan creando sobre la población (el conjunto de datos) subconjuntos lo más homogéneos posible entre las muestras que componen un grupo y lo más heterogéneo posible entre los distintos subconjuntos.
Para la efectuación de esta separación, el algoritmo se basa en las variables de entrada más significativas, es decir, las que mejor separan las muestras.

A continuación podemos observar las diferentes partes de un árbol de decisión.

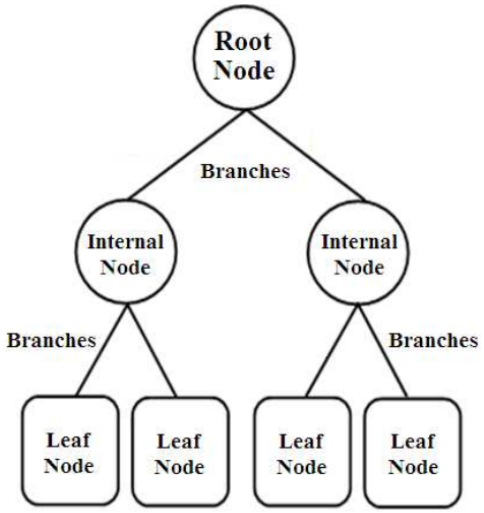

Figura 3.2: Partes de un árbol de decisión [3]

3.1.1. Terminología:

- **Nodo raíz**: Es el primero de los nodos del árbol y forma la población completa.
- **Ramificación**: Son las ramas que conectan todos los nodos del árbol por donde pasan las muestras para ser clasificadas.
- **Nodo de decisión**: Son aquellos donde las muestras se evaluan para decidir por qué rama continuar el camino hacia la solución.
- **Nodo terminal/hoja**: Estos son los nodos solución, una vez la muestra llega a este tipo de nodo, el proceso de evaluación de esta ya ha finalizado, por lo que habrá sido clasificada en alguno de los grupos categóricos existentes.
- **Poda**: Consiste en cortar u obviar una rama del árbol en la creación de un árbol, basándonos en cierta propiedad escogida, para evitar el recorrido del árbol completo y ahorrar de esta forma costos computacionales, así como para hacer frente al sobreajuste.
- **Rama/subárbol**: Es el conjunto de nodos y ramas completo que queda estrictamente por debajo de un nodo escogido del árbol total.
- **Nodos padre e hijo**: Dado un nodo del árbol, sus nodos hijo son todos aquellos que quedan conectados directamente a él únicamente en el nivel inferior siguiente. De esta forma, esos nodos hijo, comparten ese mismo padre.

3.1.2. ¿Regresión o clasificación?: similitudes y diferencias.

Similitudes

Ya sabemos, por ejemplo, que un árbol de decisión divide el espacio de los predictores en regiones no solapadas mediante el uso de los predictores más significativos.

Los árboles de decisión actúan bajo la llamada **separación binaria recursiva**, basada en un método greedy el cual decidirá en cada momento cuál será la mejor separación en el instante actual para encontrar el mejor árbol. El término 'binaria' hace alusión al tipo de división acaecido en cada nodo, es decir, que cada nodo divide en dos el espacio de los predictores. El término 'recursiva' se refiere a que el algoritmo realiza este proceso de forma reiterada hasta llegar a un criterio de parada predefinido.

Este proceso nos conduce a la generación de un árbol completo si no hacemos uso de criterios de parada, lo que nos lleva de forma directa al

problema del sobreajuste, obteniendo un modelo de una pésima calidad a la hora de evaluar nuevos datos. Por esto mismo es necesario difinir criterios de parada que realicen podas sobre el árbol para generar modelos lo suficientemente genéricos que eviten ese overfitting.

Diferencias

Las diferencias entre ambos modelos son bastante evidentes: para conjuntos de datos donde se utiliza una variable dependiente continua, utilizamos árboles de regresión, mientras que cuando la variable dependiente es categórica, usamos árboles de clasificación.
Dado esto, el **valor de los nodos hoja** no pueden ser calculados de la misma forma para ambas técnicas, por lo que, en **árboles de regresión**, utilizamos la **media** del valor de salida de las muestras que caen en dicho nodo hoja, mientras que en **árboles de clasificación** utilizamos la **moda** para asignar un valor de salida a nuevas muestras.

3.1.3. Ventajas e inconvenientes de los árboles de decisión

Ventajas

- Fáciles de comprender a la hora de interpretar los resultados.
- El tipo de dato utilizado no es una limitación.
- Es un método no paramétrico, es decir, en el que no es necesario hacer suposiciones sobre el espacio de distribución y la estructura del clasificador.
- Resulta útil a la hora de detectar la relevancia de los predictores aún habiendo una gran cantidad de estos.
- No son influidos por outliers ni valores perdidos (hasta cierto punto), por lo que requieren una menor limpieza de datos en comparación con otros métodos.

Inconvenientes

- Producen sobreajuste, por lo que hay que tener cuidado con ello mediante el uso de restricciones y la aplicación de poda.
- A la hora de trabajar con variables continuas, el árbol de decisión pierde información en el momento en el que categoriza dichas variables para la generación del árbol.

- No son del todo competentes con los mejores algoritmos de aprendizaje supervisado en cuanto a precisión en la predicción, es decir, no resultan ser tan efectivos ensambladores o SVM por ejemplo.
- Son sensibles al ruido en los datos, ya que este puede modificar de forma significativa la estructura del árbol.

3.1.4. Creación del árbol

Como ya sabemos, un árbol comienza desde un nodo raíz donde se encuentra clasificada toda la población y, conforme vamos profundizando por las ramas inferiores, vamos obteniendo subconjuntos cada vez más y más homogéneos con respecto a la variable de salida.

Para hacer posible esto necesitamos que nuestro modelo tome, por cada nodo, una decisión de separación de los datos basada en la ganancia de pureza (esa homogeneidad en los subconjuntos) al utilizar uno u otro predictor en cada uno de los nodos de decisión para crear esas particiones del espacio sucesivas.
Es decir, para cada nodo, se evalúa mediante unos medidores de pureza, cual es el predictor o característica del conjunto de datos en dicho instante que separa de mejor forma los datos que tenemos en ese momento con respecto a la variable de salida. Se escogerá en cada nivel, el predictor que mayor pureza ofrezca al árbol de decisión.

De esta forma vamos construyendo de manera progresiva ramas y más ramas del árbol haciendo uso de una técnica greedy de selección de característica a evaluar en cada nivel del árbol para la toma de decisión a la hora de generar nuevos nodos hijos.

¿Cómo medimos esa ganancia de homogeneidad?

Para los **árboles de regresión** sabemos que el objetivo de cada decisión del árbol en la creación de nuevas separaciones es minimizar la función RSS (**Residual Sum of Squares**), una medida de error usada también en la regresión lineal. Es por ello que en cada nodo se escogerá una forma de particionar los datos mediante el uso de un predictor u otro, atendiendo a cuál minimiza en mayor medida dicha fórmula.

$$\mathrm{RSS} = \sum_{m=1}^{M} \sum_{i \in R_m} (y_i - \hat{y}_{R_m})^2$$

Figura 3.3: Recursive Binary Splitting in regression trees [8]

Como el problema que nos concierne en este caso es el de **árboles de clasificación**, no entraremos en más detalles acerca de esta fórmula, ya que RSS no puede ser utilizado como criterio de separación binaria en este tipo de árboles.

Una aproximación natural a RSS es el '**ratio de error en la clasificación**', basado simplemente en la fracción de las observaciones de training en dicha región que no pertenecen a la clase más común de esta. Se calcula de la siguiente forma:

$$E = 1 - \max(\hat{p}_{mk})$$

Figura 3.4: Recursive Binary Splitting in classification trees (classification error rate) [7]

Donde $\widehat{P}$mk es la proporción $\widehat{P}$ de las observaciones de train de la región **m** que pertenecen a la clase **k**. Por tanto, el máximo de $\widehat{P}$mk hace alusión a la proporción máxima de elementos de train que siguen la moda en dicha partición.
Por desgracia esta medida de error no es lo suficientemente sensible conforme el árbol crece, por lo que es preferible el uso de otras medidas como el índice Gini y la Entropía.

Ya sabemos que buscamos nodos con una distribución de clase lo más homogénea posible. Con el fin de medir esa pureza en cada nodo, podremos utilizar los siguientes métodos ya mencionados:

- **Índice GINI**: nos indica como de pura es una región del espacio. En este caso la pureza la definimos como la proporción de items de la región que pertenecen a una misma clase. Si la región contiene un índice alto de pureza, entonces el índice Gini será bajo (muy próximo a 0), siguiendo la siguiente fórmula:

$$G = \sum_{c=1}^{C} \hat{\pi}_{mc}(1 - \hat{\pi}_{mc})$$

Figura 3.5: Recursive Binary Splitting in classification trees (gini) [9]

Donde $\hat{\pi}$mc nos indica la proporción $\hat{\pi}$ de items pertenecientes a la misma clase **c** en la región/nodo **m**.

- **Entropía (E-score)**: Se encarga también de la medida de homogeneidad de un nodo. En este caso, los resultados obtenidos al aplicar la siguiente fórmula a cada nodo para observar el nivel de entropía, quedan más visibles con respecto a la medida Gini (es decir, se ve de forma más clara el nivel de homogeneidad de un nodo). Una entropía = 0 significa homogeneidad total, una entropia = 1 significa homogeneidad nula.

$$D = -\sum_{c=1}^{C} \hat{\pi}_{mc} \log \hat{\pi}_{mc}$$

Figura 3.6: Recursive Binary Splitting in classification trees (entropia) [9]

¿Cómo evitamos el overfitting?

Una vez hemos escogido nuestra estrategia de creación del árbol, necesitamos indicarle al algoritmo cuándo ha de terminar de construirlo el uso de restricciones (prepruning) y su posterior poda (postpruning) para evitar de esta forma un sobreajuste a los datos.

Prepruning: establecimiento de parámetros

- Definir un número de observaciones mínimo sobre un nodo para que sea considerada una ramificación sobre él.
- Definir un número mínimo de observaciones sobre un nodo hoja.
- Establecer una profundidad vertical máxima para el árbol.
- Limitar el número máximo de nodos hoja.
- Parar si la expansión del nodo actual no mejora la medida de pureza utilizada actual.

Proceso de Postpruning:

1. Crear un árbol muy grande con o sin restricciones de prepruning.
2. Recorrer el árbol de abajo hacia arriba para ir cortando las hojas que nos dan ganancias negativas.

De esta forma podemos mantener ramas que, sin el proceso de postpruning podrían haber sido recortadas, pero que nos llevan a soluciones mejores que las que se ofrecen si este proceso por culpa del uso de la técnica greedy.

3.2. Hoeffding Trees y otros algoritmos de data streaming

Como ya sabemos, los algoritmos dedicados al data streaming han de seguir los siguientes requisitos:

- Procesar una muestra en cada momento y hacerlo tan solo una vez.
- Usar una cantidad de memoria limitada.
- Trabajar en un tiempo limitado.
- Estar listo para la predicción en cualquier momento.

Además, nuestro algoritmo ha de estar dotado de técnicas de detección de cambios en la distribución de los datos para evitar la disminución de la precisión en la predicción cuando esto suceda.

3.2.1. Hoeffding tree (VFDT)

Un árbol de Hoeffding es un algoritmo de inducción de árbol de decisión incremental capaz de aprender de flujos de datos masivos, suponiendo que la distribución que genera ejemplos es estacionaria, es decir, que no cambia con el tiempo. Los árboles Hoeffding explotan el hecho de que una pequeña muestra a menudo puede ser suficiente para elegir un atributo de división óptimo. Esta idea está respaldada matemáticamente por el Hoeffding bound, que cuantifica el número de observaciones (en nuestro caso, ejemplos) necesarias para estimar algunas estadísticas dentro de una precisión prescrita (en nuestro caso, la bondad de un atributo). [5]

Algunas de las técnicas de clasificación para data streaming tienen los siguientes **problemas**:

- Son altamente sensibles a la demanda de ejemplos.
- Carecen de alta eficiencia, siendo en algunos casos más lentos que un algoritmo batch.

Ante estos problemas se plantea **Hoeffding-tree** ya que:

- El aprendizaje de un Hoeffding-tree toma un tiempo constante en cada nuevo ejemplo, lo que lo hace adecuado para el aprendizaje de flujos de datos.
- Los árboles resultantes son similares a los creados con un batch learner convencional.

Hoeffding bound.

Hulten y Domingos presentan un método general para aprender de bases de datos grandes y arbitrarias. Este método consiste en derivar un límite superior para la pérdida del learner en función del número de ejemplos usados en cada paso del algoritmo. De esta forma, se minimiza el número de ejemplos requeridos en cada paso del algoritmo, a la vez que se garantiza que el modelo obtenido no difiere de forma significante de aquel que se obtendría con todos los datos. Esta metodologia de datos se ha aplicado de forma exitosa en k-means, clustering jerárquico de variables, árboles de decision, etc.

Con el fin de cumplir con los requisitos establecidos al principio de este apartado para el tratamiento de flujos de datos, los autores proponen la cota Hoeffding para ser capaces de decidir la cantidad de instancias necesarias a evaluar para alcanzar un cierto nivel de confianza a partir del cual sabemos que no es necesario evaluar más ejemplos para seleccionar un atributo mediante el cual realizar la partición del árbol en el nodo actual.
Es decir, una vez alcanzada la cota de Hoeffding, el atributo que seleccionemos para el particionamiento del espacio de predictores, sera el mismo que seleccionariamos si analizásemos una infinidad de ejemplos con el clasificador (evidentemente, con cierto nivel de confianza).

La idea básica consiste en usar un conjunto pequeño de ejemplos para seleccionar el test de divisón para colocar en un nodo del árbol de decisión. Si tras ver un conjunto de ejemplos, la diferencia en resultados entre ambos test de división no satisface un test estadístico (Hoeffding bound), entonces VFDT procede a examinar más ejemplos.

En VFDT se aprende un árbol de decisión de forma recursiva reemplazando hojas por nodos de decisión. Cada hoja almacena las estadísticas necesarias sobre los valores de los atributos. Dichas estadísticas necesarias son aquellas que se necesitan por una función de evaluación heurística que realiza el cálculo del resultado de los test de división basada en el valor de los atributos. Cuando hay un ejemplo disponible, atraviesa el árbol desde la raíz hasta una hoja evaluando el atributo requerido en cada nodo y siguiendo la rama correspondiente al valor del atributo en el ejemplo. Cuando el ejemplo llega a una hoja, la estadística de las hojas por las que ha pasado han sido actualizadas. Entonces cada condición basada en los valores de los atributos ha sido evaluada.

El nuevo nodo de decisión tendrá tantos descendientes como el número de posibles valores tenga el atributo escogido (por lo que el árbol no es necesariamente binario). Los nodos de decisión tan solo contienen la información sobre el test de división instalado en ellos.

Problema: VFDT no incluye soporte para el concept-drift por lo que,

ante cambios en la distribución de los datos, los resultados del algoritmo pueden ser malos.

Cálculo de la cota:

Teniendo n variables independientes $r_1...r_n$ con un rango R y una media $\bar{r}$, el Hoeffding bound afirma con una probabilidad 1-δ que la media real es al menos $\bar{r}$-ϵ donde:

$$\epsilon = \sqrt{\frac{R^2 \ln(1/\delta)}{2n}}$$

Figura 3.7: Cálculo del límite de Hoeffding [4]

Sabiendo que δ es la tolerancia al fallo a la hora de escoger un atributo en un nodo dado.

3.2.2. Otros algoritmos de data streaming

Muchas bases de datos grandes presentan cambios en la distribución de los datos conforme avanza la generación de estos, es decir, posee un cambio de concepto en el tiempo, un **concept drift**. El hecho de que un algoritmo de flujos de datos no esté preparado para esos efectos cambiantes puede producir un empeoramiento de los resultados predictivos con el paso del tiempo, por lo que conviene tenerlo en cuenta a la hora de implementar esta clase de algoritmos.

Ante este problema, surgen técnicas como el uso de una **ventana deslizante** que tenga en cuenta los X ejemplos más recientes para asegurarnos de aprender siempre un modelo que tenga en cuenta el concepto actual se los datos, olvidando conceptos anteriores. Ante esta situación, ha de tenerse cuidado de escoger un valor adecuado de X, ya que ha de ser lo suficientemente grande como para tener un número suficiente de ejemplos con los que aprender el modelo y lo suficientemente pequeño como para abarcar un solo concepto de los datos.

CVFDT

Un algoritmo que utiliza este concepto de ventana deslizante es el llamado **Concept-adapting Very Fast Decision Tree (CVFDT)** que, evidentemente, contiene además las características del VFDT. A continuación la explicación de su funcionamiento[6]:

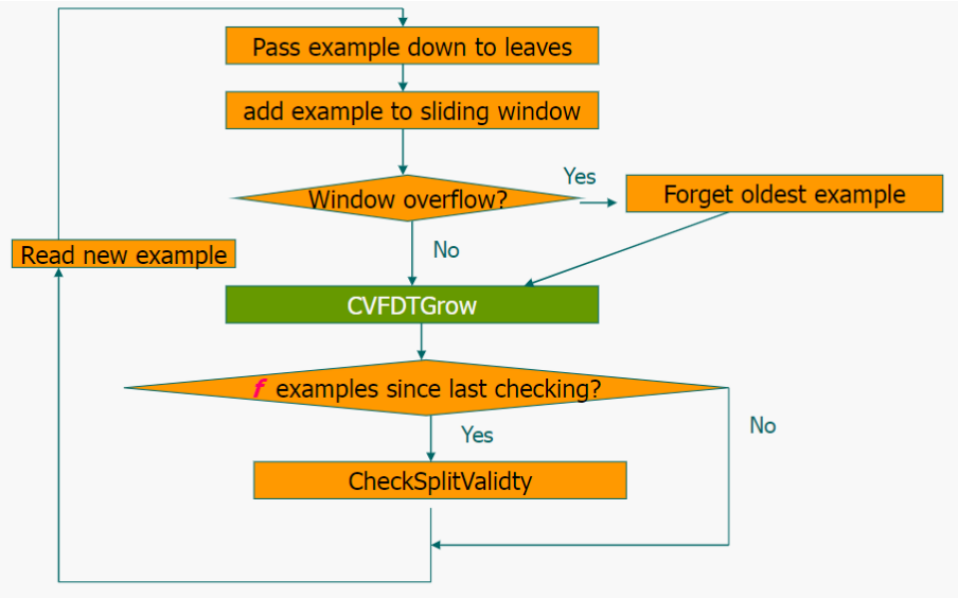

Figura 3.8: Ciclo CVFDT [10]

1. Entra un nuevo ejemplo al ciclo.
2. Se añade a la ventana deslizante.
3. Se comprueba si se excede la cantidad de ejemplos necesarios en dicho nodo para poder decidir el atributo apropiado para la división de los nodos hijos. En caso de excederse dicho tamaño de ventana, se elimina el ejemplo más antiguo.
4. CVFDTGrow: Se incrementan las estadísticas de cada uno de los nodos por los que pasa el ejemplo tanto en el árbol principal como en los subárboles. Si se ha alcanzado la cantidad suficiente de elementos analizados como para expandir un nodo hoja, escogemos el mejor de los atributos para expandir y se procede a ello.
5. CheckSplitValidity: Comprueba si comenzar un árbol alternativo o no basado en el atributo que mejor realiza la división en cada uno de los nodos. Se realiza una comprobación de las estadísticas de cada nodo observando si se produce un cambio de concepto con el nuevo ejemplo, en cuyo caso es posible que el atributo escogido anteriormente en dicho nodo posea una ganancia menor que otro, por lo que el algoritmo creará un subárbol alternativo cuya raíz será ese atributo que tenga mejor gain que el actual en dicho nodo.

Cada uno de los nodos de decisión ha de tener las estadísticas suficientes para poder olvidar información en caso de que se produzca un cambio de

concepto. Para ello, cada uno de los nodos posee un ID monotónicamente incremental que permite al algoritmo validar las decisiones previamente tomadas y realizar el olvido de información inútil en los cambios de concepto. Cuando un ejemplo antiguo es eliminado de la ventana, se recorre el árbol para decrementar las estadísticas por cada uno de los ID de los nodos por los que ha pasado.

HATT

Otro algoritmo que implementa una solución más eficiente a la presentada por Hoeffding tree es el **Hoeffding Anytime Tree** [4], el cual asegura conseguir mejores resultados explotando la información conforme le llega al modelo en lugar de esperar a traspasar el Hoeffding bound y, realizando correciones sobre estas decisiones de elección de atributo para la división siempre que sea necesario. Por tanto, en cuanto se detecte una división útil en cualquiera de las hojas del árbol, se realizará de forma inmediata y esta, será reemplazada tan pronto como otra alternativa mejor sea identificada.

Además de esto, algunos experimentos muestran que el algoritmo, aún no habiendo sido tratado para ello, muestra algunas características de tratamiento de concept drift.

Option trees

Con el objetivo de introducir a Hoeffding tree mayor estabilidad y romper la barrera que posse VFDT para mirar hacia adelante, surje Option trees[13].
Consiste en una estructura que representa a múltiples árboles en lugar de a uno solo como hace VFDT. Una instancia nueva puede bajar por varios caminos del árbol contribuyendo de diferentes maneras en diferentes opciones. La clase de un ejemplo de test se determina por un comité(mayoría de voto o pesos) hecho por las predicciones de todas los nodos hoja alcanzados. El concepto es crear múltiples opciones pero de la misma forma que lo hace Hoeffding tree.
Esta nueva representación de un árbol difiere únicamente de la representación hecha para VFDT en que contienen unos nodos llamados **nodos opción** que se encargarán de evaluar a los ejemplos que pasen por ellos mediante múltiples tests para determinar por qué ramas dejarle pasar.
En este nuevo método, un ejemplo que entra al árbol puede influir en varios nodos hoja distinto, mientras que en el Hoeffding tree, un solo ejemplo tan solo puede influir en un solo nodo hoja.

3.3. Árboles de decisión monotónicos

Conociendo las restricciones monotónicas descritas en el anterior capítulo, hemos de ser capaces de extrapolar dichos conocimientos a los árboles de decisión para poder resolver problemas de este estilo.

Recordamos que, siendo X e Y valores de atributos y C_x y C_y las clases asignadas a X e Y respectivamente, se cumple una **relación de monotonía** entre el par atributo-clase (X,C_x) y el par (Y,C_y) si y solo si (X,C_x) domina a (Y,C_y), viceversa o son exactamente iguales (tanto en valores de sus atributos como en clase asignada).

Hacemos uso de **este conocimiento sobre un árbol de decisión** de la siguiente manera:
Siendo (P,C_p) y (Q,C_Q) dos caminos distintos del mismo árbol de decisión (donde P y Q son atributos y C_p y C_q son nodos respuesta), estos son monotónicos entre sí, si cumplen las mismas reglas de monotonía descritas justo en el párrafo anterior (relación de dominancia).[12]

3.3.1. Problema:

Un conjunto de datos donde todos sus ejemplos guardan una relación de monotonía entre sí, no garantiza que genere un árbol de decisión monotónico a través de algoritmos teóricos TDIDT(top-down induction decision tree) que usan la entropía como selector de atributos.

Crear un árbol de decisión que cumpla las restricciones de monotonía estudiadas y que al mismo tiempo vele por la minimización de error (la precisión) no es tarea sencilla.

3.3.2. ¿Cómo creamos un árbol de decisión monotónico?

Aún no siendo una tarea simple, existen **métodos de creación de estos árboles** como los que describimos a continuación.

Uno de estos métodos es el **basado en matriz**. Teniendo ya construido un árbol con k ramas, construimos una matriz simétric M de tamaño k x k donde el valor m_{ij} es un 1 en caso de que la rama de la fila i no guarde una relación de monotonía con la rama de la columna j y un 0 en caso contrario. Cada columna (y fila) está asociada con un contador que hace referencia a la suma de los unos que contiene, de esta forma sabemos con qué cantidad de ramas no guarda la relación de monotonía. Comenzamos eliminando (podando) aquellas ramas del árbol que tienen un contador mayor de no-monotonía y actualizando los contadores del resto de ramas hasta llegar a obtener una matriz llena de ceros o hasta llegar a una matriz 1 x 1, en cuyo

caso M sería una matriz de no-monotonicidad.

Otro método más rápido a la hora de ejecutar es tomar inicialmente de forma aleatoria una rama del árbol y declararla como monotónica. A partir de aquí ir tomando siempre de forma aleatoria cada una de las demás ramas del árbol, compararlas con las ramas ya declaradas monotónicas y, o bien descartarlas (en caso de no guardar relación de monotonía con las ya declaradas monotónicas) o bien introducirlas en el conjunto de ramas declaradas monotónicas.

3.3.3. Método de evaluación de monotonicidad y precisión

Describimos en este apartado una métrica para que tenga en cuenta tanto el error como las restricciónes monotónicas.

Primeramente definimos una medida de no-monotonicidad para los árboles de decisión:
El **índice de no-monotonicidad** nos dice el ratio entre el número real de pares de ramas no monotónicas de un árbol de decisión y el número máximo de pares que podrían no haber sido monotónicos con respecto a otras en el mismo árbol.

Para conseguir este índice hacemos uso de la mátriz M creada en el apartado anterior y denotamos W como la suma de todas las entradas de la matriz M, es decir:

$$W = \sum_{i=1}^{k} \sum_{j=1}^{k} m_{ij}$$

Figura 3.9: Definición de W [12]

Sabemos que, como mucho, (k^2 - k) entradas de M pueden ser etiquetadas como no monotónicas, por tanto el índice de no-monotonicidad queda así:

$$I_{a_1,a_2,\ldots,a_v} = \frac{W_{a_1,a_2,\ldots,a_v}}{k^2_{a_1,a_2,\ldots,a_v} - k_{a_1,a_2,\ldots,a_v}}$$

Figura 3.10: Definición índice de no-monotonicidad [12]

Previo cálculo del índice que nos dirá como de bueno es un árbol tanto en precisión como en aguardar las restricciones monotónicas, calculamos el **order-ambiguity-score**, que se define en términos del índice previamente

calculado tal como sigue:

$$A_{a_1,a_2,\ldots,a_v} = \begin{cases} 0 & \text{if } I_{a_1,a_2,\ldots,a_v} = 0 \\ -(\log_2 I_{a_1,a_2,\ldots,a_v})^{-1} & \text{otherwise} \end{cases}$$

Figura 3.11: Definición de order-ambiguity-score [12]

Finalmente, añadimos esta medida de no-monotonicidad calculada a nuestra medida de precisión E-score de la siguiente manera:

$$T_{a_1,a_2,\ldots,a_v} = E_{a_1,a_2,\ldots,a_v} + A_{a_1,a_2,\ldots,a_v}$$

Figura 3.12: Definición de total-ambiguity-score [12]

Una vez tenemos creada la métrica, ya sabemos que, a menor valor, mejor será el resultado, ya que significará que poseemos un menor fallo en la predicción y una mayor conservación de las relaciones de montonía dentro del árbol.

Capítulo 4

Propuesta

abcsssssssssssssssssssssssss

Capítulo 5

Software desarrollado y uso

aqui softwareee

Capítulo 6

Experimentos

6.1. Framework

6.2. Resultados

6.3. Análisis

Capítulo 7

Conclusiones y trabajo futuro

Finalmente, una vez explicado el desarrollo completo del proyecto, es el momento de mirar hacia atrás para cerciorarme de las asignaturas del grado cursadas sin las que no hubiera sido capaz de desenvolverme de manera tan efectiva para la realización del proyecto, así como darme cuenta de los conocimientos que he adquirido durante esta etapa de mi carrera y, por último, realizar una valoración de todo ello.

Bibliografía

[1] Computer society digital library. http://doi.ieeecomputersociety.org/10.1109/TKDE.2012.204.

[2] Decision tree basics. https://bookdown.org/content/2031/arboles-de-decision-parte-i.html#que-son-los-arboles-de-decision.

[3] Decision tree terminology. https://www.researchgate.net/figure/a-describes-the-components-of-a-decision-tree-the-Nodes-represent-the-possible_fig2_303773171.

[4] Definición hoeffding bound. https://arxiv.org/pdf/1802.08780.pdf.

[5] Definición hoeffding tree. http://weka.sourceforge.net/doc.dev/weka/classifiers/trees/HoeffdingTree.html.

[6] Explicación cvfdt. https://pdfs.semanticscholar.org/a1a4/97e7e6845bfb81ecb5fa02fc80a47001afec.pdf.

[7] Fórmula cer. https://towardsdatascience.com/everything-you-need-to-know-about-decision-trees-8fcd68ecaa71.

[8] Fórmula rss. https://www.quantstart.com/articles/Beginners-Guide-to-Decision-Trees-for-Supervised-Machine-Learning.

[9] Fórmulas gini y entropía. https://www.quantstart.com/articles/Beginners-Guide-to-Decision-Trees-for-Supervised-Machine-Learning.

[10] Imagen ciclo cvfdt. https://studylib.net/doc/9462259/cvfdt-algorithm.

[11] A survey on learning from data streams: current and future trends. .

[12] Ben-David. Monotonicity maintenance in information-theoretic machine learning algorithms, 1995.

[13] Geoffrey Holmes Bernhard Pfahringer and Richard Kirkby. New options for hoeffding trees, 2007.

[14] R. Lior and M.O. Z. Data mining with decision trees: Theory and applications, 2007.