1. Introducción.

En el presente trabajo práctico se realizaron diversos experimentos de aprendizaje automático. El problema de clasificación a resolver consta de un conjunto de datos con 1 millón de instancias, 28 características y una distribución por clases 90 % y 10 % de clase mayoritaria y minoritaria respectivamente. La distribución de número de instancias entre training y test es de un millón de datos cada una.

Se utilizará como medida de evaluación TPR x TNR en test (el producto de los porcentajes de clasificación en cada clase). El objetivo es maximizar dicha métrica mediante el uso de distintos algoritmos de clasificación y preprocesamiento.

Como el problema a resolver consta de tantos datos no podemos hacer el uso de algoritmos tradicionales. Haremos uso de la filosofía map reduce y trabajaremos en entornos de datos distribuidos. Más concretamente usaremos Spark y el servidor Hadoop de la UGR. Se resolverá el problema utilizando la biblioteca MLLib y los algoritmos disponibles en Spark Packages.

2. Experimento 1.

En nuestro primer contacto con el problema, usamos un árbol de decisión implementado en mlib de Spark. Sin preprocesamiento, los resultados como era de esperar no fueron los mejores.

2.1. Decision Trees.

El árbol de decisión es un algoritmo gredy que realiza una partición binaria recursiva del espacio de características. El árbol predice la misma etiqueta para cada partición más inferior (hoja). Cada partición se elige seleccionando la mejor división de un conjunto de divisiones posibles, para maximizar la ganancia de información en un nodo de árbol. En otras palabras, la división elegida en cada nodo de árbol se elige del conjunto $\text{ArgMax}_s IG(D, s)$ donde $IG(D, s)$ es la ganancia de información cuando una división s se aplica a un conjunto de datos D.

La impurity del nodo es una medida de la homogeneidad de las etiquetas en el nodo. La implementación actual de mlib proporciona dos medidas de impurity para la clasificación (impurity de Gini y entropía) y una medida de impurity para la regresión (varianza).

Impurity	Problema	Fórmula	Descripción
Gini	Clasificación	$\sum_{i=1}^{C} f_i(1 - f_i)$	f_i es la frecuencia de la etiqueta i en un nodo y C es el número de etiquetas únicas.
Entropía	Clasificación	$\sum_{i=1}^{C} -f_i log(f_i)$	f_i es la frecuencia de la etiqueta i en un nodo y C es el número de etiquetas únicas.
Varianza	Regresión	$\frac{1}{N}\sum_{i=1}^{N}(y_i - \mu)^2$	y_i es la etiqueta para una instancia, N es el número de instancias y μ es la media

La ganancia de información es la diferencia entre la impureza del nodo primario y la suma ponderada de las dos impurezas del nodo secundario. Suponiendo que una división s divide el conjunto de datos D de tamaño N en dos conjuntos de datos D_{left} y D_{right} de tamaños N_{left} y N_{right}, respectivamente, la ganancia de información es:

$$IG(D, s) = Impurity(D) - \frac{N_{left}}{N} Impurity(D_{left}) - \frac{N_{right}}{N} Impurity(D_{right}).$$

2.1.1. Candidatos para hacer la partición:

Para variables continuas se calcula un conjunto aproximado de candidatos divididos mediante la realización de un cálculo cuantil sobre un sampleo de los datos. Las divisiones ordenadas crean "binsz el número máximo de dichos bins se puede especificar utilizando el parámetro maxBins.

En variables categóricas, la clasificación binaria (0/1), puede reducir el número de candidatos divididos a M 1 ordenando los valores categóricos de las características por la etiqueta promedio.

2.1.2. Regla de Parada:

La construcción del árbol recursivo se detiene en un nodo cuando se cumple una de las siguientes condiciones:

- La profundidad del nodo es igual al parámetro de entrenamiento maxDepth.
- Ningún candidato dividido conduce a una ganancia de información mayor que minInfoGain.
- Ningún candidato dividido produce nodos secundarios, cada uno de los cuales tiene al menos instancias de entrenamiento minInstancesPerNode.

2.1.3. Escalabilidad:

La computación se escala aproximadamente linealmente en el número de instancias de entrenamiento, en el número de características y en el parámetro maxBins. La comunicación escala aproximadamente linealmente en el número de características y en maxBins. El algoritmo implementado lee datos dispersos y densos. Sin embargo, no está optimizado para entradas dispersas.

2.2. Resultados:

Para nuestro conjunto de datos, Higgs obtuvimos los siguientes resultados:

Accuracy	0.47104274147391845
TPR * TNR	0.3983475060150286
TPR	0.47095271647096976
TNR	0.8458333333333333

Como podemos observar hay problema de clasificación para la clase minoritaria.

3. Experimento 2.

Como obervamos en el experimento 1 había un desbalanceo de clases que influía notariamente en la clasificación. Para lidiar con ello igualaremos las clases mediantes un algoritmo de preprocesamiento, ROS. Además usaremos un modelo de árboles de decisión basado en ensembles, Randon Forest.

3.1. ROS, RUS.

Las técnicas de remuestreo se pueden clasificar en tres grupos o familias:

- Métodos de undersampling, que crean un subconjunto del conjunto de datos original al eliminar instancias (generalmente instancias de clase mayoritaria).
- Métodos de oversampling, que crean un superconjunto del conjunto de datos original replicando algunas instancias o creando nuevas instancias a partir de las existentes.
- Métodos híbridos, que combinan ambos enfoques de muestreo.

Su procedimiento de trabajo es muy simple: están dedicados a eliminar aleatoriamente ejemplos de la clase mayoritaria o replicar ejemplos de la clase minoritaria. Este proceso se lleva a cabo solo en el conjunto de capacitación con el objetivo de reequilibrar la distribución de datos al 50 %. En el primer caso, es decir, el muestreo aleatorio, el principal inconveniente es que puede descartar datos potencialmente útiles, que podrían ser importantes para el proceso de aprendizaje. Para el sobremuestreo aleatorio, varios autores coinciden en que este método puede aumentar la probabilidad de que se produzca un sobreajuste, ya que hace copias exactas de las instancias existentes.

3.2. Random Forest.

Los Random Forest son conjuntos de árboles de decisión. Los RF son uno de los modelos de aprendizaje automático más exitosos para la clasificación y la regresión. Combinan muchos árboles de decisión para reducir el riesgo de sobreajuste. Los RF entrenan un conjunto de árboles de decisión por separado, por lo que la capacitación se puede realizar en paralelo. El algoritmo inyecta aleatoriedad en el proceso de capacitación para que cada árbol de decisión sea un poco diferente. La combinación de las predicciones de cada árbol reduce la varianza de las predicciones, mejorando el rendimiento en los datos de prueba.

La aleatoriedad inyectada en el proceso de capacitación incluye:

- Submuestreo del conjunto de datos original en cada iteración para obtener un conjunto de entrenamiento diferente (bootstrapping).
- Considera diferentes subconjuntos aleatorios de características para dividir en cada nodo del árbol.

Además de estas aleatorizaciones, el entrenamiento del árbol de decisión se realiza de la misma manera que para los árboles de decisión individuales. Para hacer una predicción en una nueva instancia, un RF debe agregar las predicciones de su conjunto de árboles de decisión. Esta agregación se realiza de manera diferente para la clasificación y la regresión.

- Clasificación: voto mayoritario. La predicción de cada árbol se cuenta como un voto para una clase. Se predice que la etiqueta será la clase que reciba más votos.
- Regresión: promedio. Cada árbol predice un valor real. Se predice que la etiqueta es el promedio de las predicciones de árbol.

3.3. Resultados:

En esta ocasión, los resultados tras balancear las clases con ROS, random oversampling, y usar un random forest como clasificador obtuvimos:

Accuracy	0.6727889822334111
TPR * TNR	0.45305419925488544
TPR	0.6388405393487897
TNR	0.7091819810256751

Integrar técnicas de preprocesamiento aumentó mucho el acierto de clasificación para la clase minoritaria, no se pudo probar a hacer el mismo experimento con un RF pero sin preprocesamiento ya que algunos árboles que conforman el RF obtenían subconjuntos de datos con una sola clase debido al alto desbalanceo de las clases.

4. Experimento 3.

Con motivo de comparar el oversampling y el undersampling, en este experimento se usará RUS y el mismo clasificador RF.

4.0.1. Resultados:

Accuracy	0.6711389988433106
TPR * TNR	0.4487843258589566
TPR	0.6473630985383195
TNR	0.693249780335435

Los resultados son levemente peores, observamos que aunque el acierto para la clase minoritaria aumenta, también disminuye para la de la mayoritaria.

5. Experimento 4.

En este experimento haremos un balanceo de clases mediante ROS y un filtrado de ruido, HME-BD Noise Filter, de esa forma reduciremos instancias redundantes para la clasificación. También usaremos otro clasificador basado en ensembles, PCARD.

5.1. HME-BD Noise Filter.

HME-BD se basa en un esquema de partición de conjunto de datos. Realiza un k-fold de los datos de entrada, dividiendo los datos en k particiones. La partición de test es un 1kth de las k particiones, y el train es el resto de las k-1 particiones. Se aprende un random forest profundo (un bosque aleatorio con árboles profundos) en cada fold, utilizando la partición de train como entrada. Una vez el proceso de aprendizaje ha finalizado, cada uno de los k modelos aprendidos predicen la partición de test correspondiente de cada fold de esa manera, los modelos predecirán los datos que no vieron mientras fueron aprendidos. El último paso es eliminar las instancias ruidosas. Esto se hace mediante una comparación de las etiquetas del conjunto test original con las predichas. Si las etiquetas son diferentes, la instancia se considera como ruidosa y eliminada. Finalmente, todas las particiones filtradas se unen para componer un conjunto de datos limpio de ruido.

5.2. Resultados:

Los resultos obtenido fueron los siguientes:

Accuracy	0.6893489
TPR * TNR	0.47565014157236457
TPR	0.6957347703762888
TNR	0.6836659052056702

El uso de un filtrado de ruido nos devuelve un conjunto de datos de mayor calidad y por tanto conseguimos mejorar un poco nuestrá métrica TPR * TNR.

6. Experimento 5.

Usaremos randomforest como clasificador y en esta ocasión usaremos otro filtro de ruido, NCNEdit BD. Añadiremos ROS para lidiar con el desbalanceo de las clases.

6.1. NCNEdit BD.

Este algoritmo utiliza la regla de clasificación knn con la estimación del error de dejar uno. Descarta instancias si está mal clasificado utilizando kNCN. En la NCN regla de clasificación, el entorno no está solo definido por la proximidad de prototipos a un determinado ejemplo, sino también por su distribución simétrica de su alrededor.

6.2. Resultados.

Accuracy	0.6865431797351221
TPR * TNR	0.46996213437525736
TPR	0.6609354605082797
TNR	0.7110560144765754

Al parecer este filtrado de ruido funciona peor que el anterior para este problema. Se ha aumentado el error de clasificación de la clase minoritaria.

7. Experimento 6.

Usaremos RNG filter y compararemos los resultados con los experimentos anteriores. El clasificador nuevamente será un random forest.

7.1. RNG BD.

Este filtro de ruido calcula la proximidad del grafo de los datos. Todos los vecinos del grafo de cada instancia dan un voto para su clase. Si la etiqueta difiere de la etiqueta original, la instancia se considera ruido y se elimina.

7.2. Resultados.

Accuracy	0.6985126544320499
TPR * TNR	0.48679265519212783
TPR	0.6678314656384303
TNR	0.7289154228855721

Hasta ahora este filtro de ruido es el que ha ayudado más en nuestro problema de clasificación.

8. Experimento 7.

Finalmente además del filtrado de ruido, el oversampling, también añadiremos un algoritmo de selección de instancias.

8.1. FCNN_MR

Este algoritmo es uno de los más extendido y ampliamente utilizado en la reducción de datos. Es un algoritmo independiente del orden, basado en la regla NN, para encontrar un subconjunto del conjunto de datos de entrenamiento. Tiene una complejidad cuadrática del tiempo en el peor de los casos. Demostró escalar bien en conjuntos de datos grandes y multidimensionales.

8.2. Resultados.

Accuracy	0.5068370986920333
TPR * TNR	0.46227039267904196
TPR	0.4904903875809602
TNR	0.9424657534246575

La selección de instancias con este algoritmo parece que eliminó instancias de clase minoritaria con lo cual se vio afectado de forma muy negativa en la clasificación.