UNIVERSIDAD
DE GRANADA

Aprendizaje Automático. Proyecto Final.

Daniel Navarrete Martin,
Ricardo Ruiz Fernández de Alba,

Escuela Técnica de Ingeniería Informática

DECSAI

Universidad de Granada

20 de junio de 2022

Índice general

1

Planteamiento del Problema

Enlace a la carpeta datos de consigna UGR

1.1 | Descripción de los datos

En este problema trabajaremos con el conjunto de datos de Fallos del sistema de presión de aire (APS) en camiones Scania (APS Failure at Scania Trucks Data set en versión original). Este conjunto de datos presenta un problema de clasificación, en el cual tendremos que predecir si un determinado fallo es causado por un elemento del sistema APS o por otro elemento distinto.

Nuestros datos vienen definidos en dos ficheros distintos, uno de entrenamiento, formado por 60000 datos y 171 características, y otro de test, con 16000 datos y las mismas características. La primera de las características es la clase etiqueta. Las características son una combinación de datos numéricos y de contenedores de histograma. *El nombre de las características ha sido anonimizado por derechos de propiedad.* La distribución del atributo clase es la siguiente:

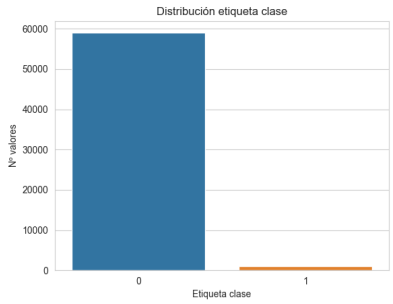

Figura 1.1: Distribución de la etiqueta clase

Observando la Figura 1.1 tenemos 59000 datos pertenecientes a la clase negativa, y solo 1000 pertenecen a la clase positiva. Esto nos muestra que estamos antes un conjunto de datos altamente desbalanceado.

1.2 | Manejo de los datos perdidos

Nuestro conjunto de datos contiene una gran cantidad de datos perdidos. Como podemos ver en la Figura 1.2, donde se representan las 15 características con mayor porcentaje de valores perdidos, hay varias columnas las cuales superan el 50 % de valores perdidos.

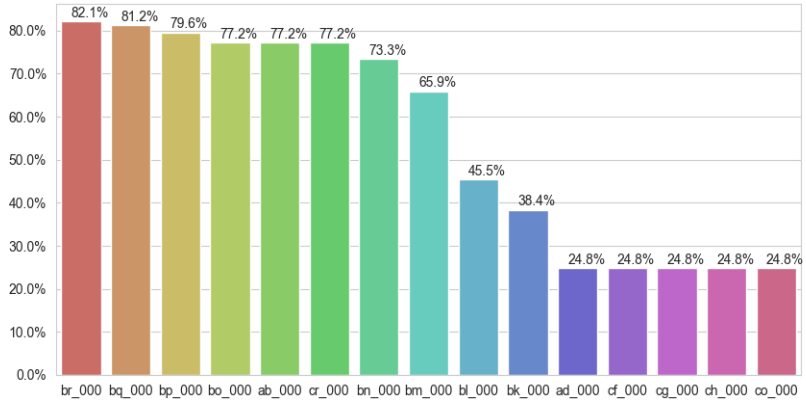

Figura 1.2: Las 15 columnas con mayor porcentaje de valores perdidos

Para resolver este problema, se actuará de la siguiente manera:

- Eliminar las columnas con un porcentaje de valores perdidos superior al 70 % de los datos.
- Eliminar las filas que superen el 15 % de valores perdidos. Eliminar también las filas con valores perdidos cuyas columnas tengan un porcentaje de valores perdidos inferior al 5 %.
- Para las columnas que contengan un porcentaje de valores perdidos superior al 5 % e inferior al 70 %, se imputarán los valores realizando la suma del valor medio de la variable más un valor aleatorio en el intervalo $[-1.5\sigma, 1.5\sigma]$, siendo σ la desviación típica de dicha variable.

1.3 | Preprocesado de los datos

Para llevar a cabo la tarea del preprocesado de los datos, lo primero será calcular la desviación típica de las columnas del dataset, y eliminar aquellas cuyo resultado sea 0. Este caso se da únicamente para la columna *cd_000*, por lo tanto la eliminaremos de nuestro conjunto de datos puesto que no aporta ninguna información para la resolución de nuestro problema.

Lo siguiente será llevar a cabo técnicas de balanceo de los datos, mediante las cuales conseguiremos una mejor capacidad de aprendizaje. Para ello, haremos uso de las siguientes técnicas:

- **SMOTE.** Esta es una técnica de sobremuestreo de datos, mediante la cual se crean nuevos ejemplos sintéticos de la clase minoritaria, en lugar de realizar un sobremuestreo por sustituciones. Para ello, el parámetro de la estrategia de muestreo tomará el valor 0.3. Este parámetro sigue la fórmula 1.1

$$\alpha_{os} = \frac{N_{rm}}{N_M} \tag{1.1}$$

  Donde N_{rm} es el número de muestras en la clase minoritaria después del muestreo, y N_M es el número de muestras en la clase mayoritaria. De esta manera, obtendríamos una distribución de las clases de 54502 para la clase negativa, y 16350 para la clase positiva. Pese a haber aumentado el número de muestras de la clase minoritaria, aún sigue habiendo un claro desbalanceo, por lo que utilizaremos otra técnica más a nuestro modelo.

- **UnderSampling.** Mediante esta técnica, lo que haremos será modificar la distribución de los datos reduciendo el número de casos de la clase mayoritaria. En nuestro problema usaremos **RandomUnderSampler** para llevar a cabo esta tarea. Este elige aleatoriamente los datos que serán reducidos. Del mismo modo que el anterior, el parámetro de la estrategia de muestreo toma el valor 0.5, y sigue la fórmula 1.2

$$\alpha_{us} = \frac{N_m}{N_{rM}} \tag{1.2}$$

  Ahora N_m es el número de muestras de la clase minoritaria, y N_{rM} el número de muestras en la clase mayoritaria. La distribución final de la muestra contiene 32700 muestras de la clase negativa, y 16350 de la clase positiva, resultando en un conjunto de datos mejor balanceado.

Una vez hemos llevado a cabo el balanceo de nuestro conjunto de datos, finalizaremos el preprocesado de nuestros datos llevando a cabo un estandarizado de los mismos, para transformar los datos de forma que todos los predictores estén aproximadamente en la misma escala. Para ello, utilizaremos el **StandardScaler** de Sklearn, que hace uso de la estandarización *Z-score*, $z = \frac{x-\mu}{\sigma}$, la cual divide cada predictor entre su desviación típica después de haber sido centrado, de tal forma que los datos pasan a tener una distribución normal.

1.4 | Métrica de error

Para ayudarnos a estimar el mejor modelo, haremos uso de la métrica de error **Macro-F1** como medida de rendimiento para nuestro problema. Esta tiene en cuenta el valor de la métrica *F1* de cada una de las clases, lo que nos va a servir para mostrarnos el rendimiento de nuestro modelo basándonos en el número de puntos correctamente clasificados para ambas clases, lo cual es de mucha utilidad debido al alto coste de una clasificación errónea en un sistema APS. La métrica sigue la siguiente fórmula:

$$Macro - Precision = \frac{Precision1 + Precision2}{2}$$
$$Macro - Recall = \frac{Recall1 + Recall2}{2}$$
$$Macro - F1 = 2 \cdot \frac{Macro - Precision \cdot Macro - Recall}{Macro - Precision + Macro - Recall}$$

2

Modelos Lineales

2.1 | Regresión Logística

El modelo lineal elegido para llevar a cabo el problema de clasificación es **Regresión Logística.** Este es un modelo lineal de clasificación, donde ahora la función hipótesis de es de la forma

$$h(x) = \theta(w^T X) \tag{2.1}$$

con $\theta(s) = \frac{e^s}{1+e^s} = \frac{1}{1+e^{-s}}$ la llamada **función logística o sigmoide** que toma valores en $[0, 1]$. Estos se interpretan de forma probabilística para un evento binario. A través del método de máxima verosimilitud, obtenemos una medida de error en la muestra:

$$E_{in}(w) = \frac{1}{N} \sum_{n=1}^{N} \ln(1 + e^{y_n w^T x_n}) \tag{2.2}$$

Intuitivamente, el error se acerca más a cero cuanto mayor es cada $y_n w^T x_n$ (positivo). Esto implicaría que $\text{sign}(w^T x_n) = y_n$, es decir, un número mayor de instancias x_n bien clasificadas.

2.1.1 | Parámetros del modelo

A continuación empezaremos a comentar y definir los parámetros que formarán parte de nuestro modelo de Regresión Logística.

- El tipo de regularización usada en este caso es *Lasso (l1)*. En esta, la complejidad C se mide como la media del valor absoluto de los coeficientes del modelo. Ver

ecuación 2.3

$$C = \frac{1}{N} \sum_{j=1}^{N} w_i \tag{2.3}$$

Se usa esta técnica de regularización ya que es la más apropiada para problemas de alta dimensionalidad, y porque ayuda con la sospecha de que algunos de los atributos de entrada sean irrelevantes, fomentando así una solución poco densa, ya que se favorece que algunos de los coeficientes acaben valiendo 0.

- El número máximo de iteraciones que tendrá el modelo para converger será de 1000 iteraciones.
- El algoritmo de optimización usado será *liblinear*.
- El parámetro *random_state* tomará el valor 30, y nos servirá para mezclar los datos.

2.1.2 | Estimación hiperparámetros

Para llevar a cabo la estimación de los hiperparámetros, se han creado distintos modelos, con los posibles valores que iban a tomar los hiperparámetros, y se ha llevado a cabo la técnica de validación cruzada *k-fold 5*, usando la métrica de error vista en el apartado anterior.
Los hiperparámetros que han sido estimados, han sido:

- Tolerancia del modelo. Este es un parámetro muy importante en el desempeño del modelo, puesto que marca el criterio de parada del mismo. Los posibles valores que tomaba eran: 0.001 y 0.0001.
- Inversa de la fuerza de regularización, **C**. Indica la fuerza de la regularización en el modelo. Los valores que serán estimados son: 0.5, 1 y 1.5.

2.1.3 | Resultados de Validación Cruzada

Una vez ya hemos definido nuestros parámetros e hiperparámetros, es momento de analizar los resultados obtenidos por el modelo. Tras la ejecución de la técnica de validación cruzada que vimos anteriormente, obtenemos los siguientes resultados. Ver tabla 2.1

tol \ C	0.5	1	1.5
0.001	0.9651	0.9654	0.9656
0.0001	0.9662	0.9668	0.9669

Cuadro 2.1: Resultados de ejecución de Regresión Logística.

Analizando la tabla anterior, extraemos que nuestro mejor modelo de Regresión Logística lo obtenemos cuando los hiperparámetros toman los valores de 0.0001 para la tolerancia, y de 1.5 para la regularización. Finalmente, nuestro modelo queda de la siguiente forma:

```
LogisticRegression(penalty = l1, C = 1.5, tol = 0.0001, max_iter = 1000,
solver = liblinear, random_state = 30)
```

3

Modelos No lineales

Los métodos de ensemble de árboles son modelos no lineales que combinan múltiples modelos de árboles para conseguir un equilibro entre bias y varianza, mejorando las predicciones de los modelos individuales originales. Hemos utilizado los dos siguientes:

3.1 | Random Forest (RF)

Un modelo Random Forest consiste en el entrenamiento de varios árboles de decisión individuales no correlados, con muestras de entrenamiento generadas mediante bootstrapping. La predicción se obtiene agregando cada una de las predicciones de los árboles individuales.

3.1.1 | Índice Gini

Siendo $\hat{p}_{mk} = \frac{1}{N_m} \sum_{x_i \in R_m} \mathbb{I}[y_i = k]$ con $\mathbb{I}$ función indicadora que devuelve 1 o 0 según la evaluación del predicando.

$$G_{ind} = \sum_{k=1}^{K} \hat{p}_{mk}(1 - \hat{p}_{mk}) \tag{3.1}$$

El valor $\hat{p}_{mk}$ representa la proporción de observaciones en la m-ésima región de la k-ésima clase. El **Índice Gini** (G_{ini}) corresponde el error de entrenamiento en clasificar una observación junto con su probabilidad. Esencialmente es un medida de la varianza por lo que valores bajos de éste indice indican una mejor clasificación.

3.1.2 | Descripción de los hiperparámetros

Para entrenar el modelo se ha utilizado la clase `RandomForestClassifier` de sklearn haciendo uso de Validación cruzada 5-fold.

Esta toma como argumentos:

- `n_estimators`: Número de árboles del modelo a utilizar. Por defecto son 100.
- `criterion`: Usamos el criterio por defecto, que es el Índice gini mencionado anteriormente. En este caso, es equivalente a los criterios de entropía y log_loss pues nuestro problema es de clasificación binaria.
- `max_depth`: Máxima profundidad de cada árbol. Por defecto los nodos se expanden sin limitación.

3.1.3 | Resultados de Validación Cruzada

Debido al alto coste computacional del modelo, se han tomado 3 configuraciones distintas de hiperparámetros variando tanto el número de estimadores como la profundidad máxima.

Tras un experimento inicial con 5 estimadores y profundidad de 75 se ha aumentado cada hiperparámetrs en razón de 5 y 20 respectivamente para comprobar la variación de la Métrica de Error F - Macro en el conjunto de validación.

Iteración	Nº de Estimadores	Profundidad máxima	Puntuación VC
1	5.0	75.0	0.992115
2	10.0	95.0	0.993868
3	15.0	115.0	0.993906

Cuadro 3.1: Resultados de ejecución de Random Forest para 3 configuraciones distintas.

En efecto, se optienen altas puntuaciones con validación cruzada en cada casos siendo el mejor cuando se utilizan 15 estimadores, con profundidad máxima de 115 ($Macro - F1 = 0.9939$).

3.2 | Gradient Boosting Model (GB)

Es un modelo no lineal de ensemble de árboles, pero a diferencia de Random Forest, pertenece a la clase de modelos de Boosting (junto con AdaBoost y Stochastic Gradient Boosting).

Estos modelos ajustan secuencialmente varios modelos sencillos (*weak learners*) de manera que cada modelo aprende de los errores del anterior. En el caso de **Gradient Boosting Trees** (GBT), utilizamos árboles de decisión con pocas ramificaciones como *weak learners*.

Gradient Boosting es una generalización del algoritmo AdaBoost que permite emplear cualquier función de coste mientras sea diferenciable. Su flexibilidad y éxito general están vinculadas a su caracter secuencial: cada modelo ajusta los residuos (errores) de los anteriores.

3.2.1 | Descripción de los hiperparámetros

- `loss`: Función de pérdida a optimizar. Si se utiliza `exponential` se recupera el el modelo AdaBoost. usaremos `log_loss`, la función por defecto:

$$L_{log}(y, p) = -[y \log p + (1 - y) \log(1 - p)] \tag{3.2}$$

- `learning_rate`: La tasa de aprendizaje reduce la contribución de cada árbol. Existe un compromiso entre la tasa de aprendizaje y el número de estimadores. Por defecto es 0.1

- `n_estimators`: El número de etapas de boosting a realizar. El modelo Gradient Boosting es bastante resistente al sobreajuste, luego aumentar este parámetro implica generalmente un mejor rendimiento. Por defecto es 100

- `max_depth`: Profundidad máxima para los weak learners. Por defecto no hay límite.

Para su implementación, se utilizó inicialmente la clase `GradientBoostingClassifier` de sklearn. Sin embargo, no fue posible realizar ajuste en un tiempo razonable pues para este modelo sklearn no implementa paralelización a diferencia de los modelos anteriores (`n_jobs=-1`). Por ello, se ha optado por usar la clase `LGBMClassifier` de la biblioteca lightgbm desarrollada por Microsoft totalmente compatible con sklearn.

3.2.2 | Resultados de Validación Cruzada

Se ha considerado para este modelo un número elevado de estimadores con bajo nivel de profundidad que están en compromiso con una baja tasa de aprendizaje.

Hemos realizado Validación cruzada 5-fold en cada modelo, aumentando el número de estimadores en 100 partiendo de 200 y la profundidad máxima en 1. Esto mantiene bajo el número de ramificaciones

La tasa de aprendizaje se ha dejado constante 0.1 pues valores más bajos implicaba usar mayor número de estimadores con el coste computacional asociado.

i	Nº Estimadores	Profundidad Máxima	Tasa de Aprendizaje	Puntuación VC
1	200.0	2.0	0.1	0.986297
2	300.0	3.0	0.1	0.995124
3	400.0	4.0	0.1	0.996800

Cuadro 3.2: Resultados de ejecución de Gradient Boosting para configuraciones distintas.

En efecto, el mejor resultado se alcanza con el mayor número probado de estimadores (400) y profundidad máxima 4 ($F1 - macro = 0.9968$).

3.3 | Bonus. Support Vector Machine (SVM) y SGD

La Máquina de Vector Soporte es un modelo de clasificación que busca estimar el hiperplano con mayor margen de separación del modelo de datos.

3.3.1 | Función de error

Usaremos la función de pérdida Hinge Loss, que incorpora en su cálculo un margen desde el hiperplano. Aunque las nuevas observaciones sean clasificadas correctamente, pueden implicar una penalización si el margen de la frontera de decisión no es lo suficientemente amplio.

$$L_{Hinge}(y, w) = \text{máx}\{1 - wy, 0\} = |1 - wy|_+ \tag{3.3}$$

Inicialmente, se pensó en utilizar SVM en su versión no lineal con una transformación de kernel, lo cual corresponde la clase `SVC` de sklearn. Sin embargo, esto es inviable para conjuntos de datos con decenas de miles de instancias, como en nuestro caso. Por ello, se ha optado por usar SVM en su versión lineal con `SGDClassifier`.

3.3.2 | Descripción de los hiperparámetros

Los parámetros utilizados han sido:

- `loss`: Función de pérdida a minimizar con SGD. Hemos utilizado `hinge` antes mencionada para SVM lineal.
- `penalty`: Tipo penalización o término de regularización a usar. Por defecto es `l2`, que es estándar para SVM lineal.
- `alpha`: Constante que multiplica el término de regularización. También se utiliza para computar la tasa de aprendizaje cuando toma el valor `optimal`.
- `tol`: Tolerancia para el criterio de parada. Por defecto es 0.001
- `max_iter`: Número máximo de épocas para SGD. Por defecto toma 1000

3.3.3 | Resultados de Validación Cruzada

Se ha comprobado experimentalmente que valores de $\alpha \approx 2.5 \cdot 10^{-3}$ proporcionan los mejores resultados (incrementos de $5 \cdot 10^{-5}$). Además, se aumentado la tolerancia a 0.1 comprobando que se mejora tanto la puntuación como el tiempo de ejecución.

i	Penalización	α	Tolerancia	Nº Max Iteraciones	Puntuación CV
1	l2	0.00250	0.1	1000	0.963310
2	l2	0.00255	0.1	1000	0.963241
3	l2	0.00260	0.1	1000	0.963204

Cuadro 3.3: Resultados de ejecución de SVM para configuraciones distintas.

En efecto, para $\alpha = 2.5 \cdot 10^{-3}$, tolerancia 0.1 y número máximo de iteraciones 1000, se obtiene la mejor puntuación. ($F1 - Macro = 0.96331$).

4

Comparación Modelos

Una vez que ya hemos estimado nuestros cuatro mejores modelos, dos lineales (Regresión Logística y Support Vector Machine con SGD) y dos no lineales (Random Forest y Gradient Boosting), nos encontramos en disposición de comparar estos modelos y seleccionar el que mejor resultados ha obtenido. Para ello, lo primero de todo será enfrentar los modelos en la tabla 4.1.

Modelo	Score
Baseline	0.4
RL	0.9669
RF	0.9939
GB	0.9967
SVM	0.9633

Cuadro 4.1: Comparación de los tres modelos.

Como observamos en la tabla anterior, todos los modelos obtienen resultados bastante superiores al modelo **Baseline**, el cual, como podemos ver en la Figura 4.1, solo clasifica bien los verdaderos negativos.

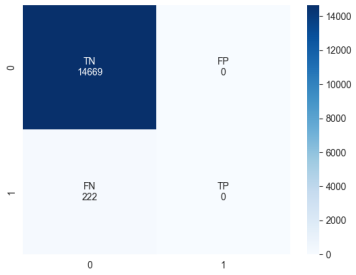

Figura 4.1: Matriz de confusión del modelo Baseline

Pero los mejores resultados, los obtiene el modelo no lineal **Gradient Boosting**, el cual obtiene el mayor valor tras realizar validación cruzada sobre los datos de entrenamiento.

4.0.1 | Estimación E_{out}

Una vez ya hemos elegido nuestro mejor modelo, es el momento de realizar la estimación del valor E_{out} y ver como se comporta el modelo sobre los datos de test, y representaremos la matriz de confusión del modelo.

Gradient Boosting obtiene un valor $E_{out} \approx E_{test} = 0.8966$. Este es un gran resultado, cercano a 1, lo que nos muestra que el modelo es capaz de predecir bien sobre los datos de test, así como lo hacía en los de entrenamiento.

Analizaremos ahora el resultado anterior fijándonos en la matriz de confusión del modelo.

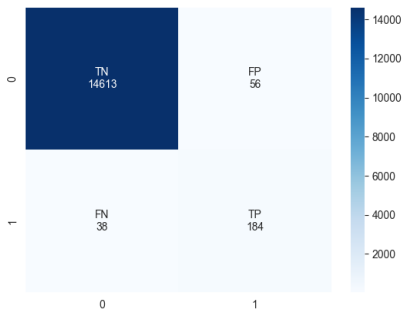

Figura 4.2: Matriz de confusión de Gradient Boosting

Como podemos ver en la Figura 4.2, el modelo clasifica bastante bien los verdaderos negativos, así como los verdaderos positivos, reduciendo casi al máximo tanto los falsos positivos como los falsos negativos. Esto nos indica un gran desempeño del modelo, lo cual es acorde al resultado obtenido con anterioridad.

4.0.2 | Curva de Aprendizaje

Como última parte de nuestro problema, analizaremos la curva de aprendizaje del modelo, representada en la Figura 4.3

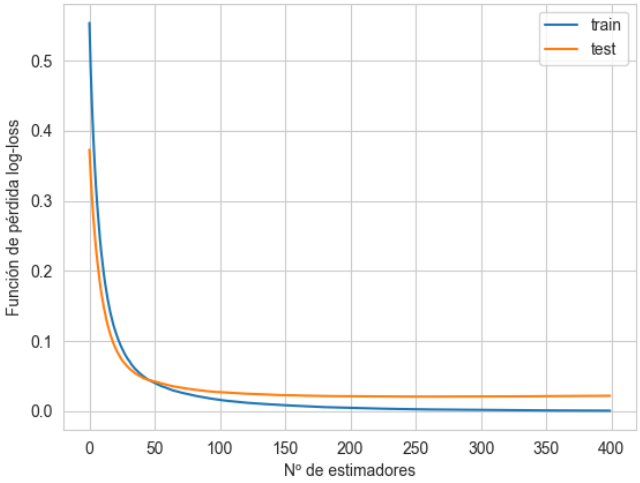

Figura 4.3: Curva de Aprendizaje de Gradient Boosting

Observando la anterior figura, vemos como el error de entrenamiento es mayor que el de test, cuando tenemos pocos estimadores. Esto cambia a partir de 50 estimadores aproximadamente, donde ambas curvas se cortan, y a partir de este punto, los datos de entrenamiento van reduciendo el error hasta que este llega a 0, a medida que el número de estimadores aumenta, mientras que el de test, llega un punto en el que no es capaz de mejorar este error, hasta terminar en los 400 estimadores.