Aprendizaje Automático

Doble Grado en Ingeniería Informática y Matemáticas
Universidad de Granada

Proyecto Final

Pablo Álvarez Cabrera
Luis Antonio Ortega Andrés

18 de junio de 2020

Índice

1. Descripción del problema

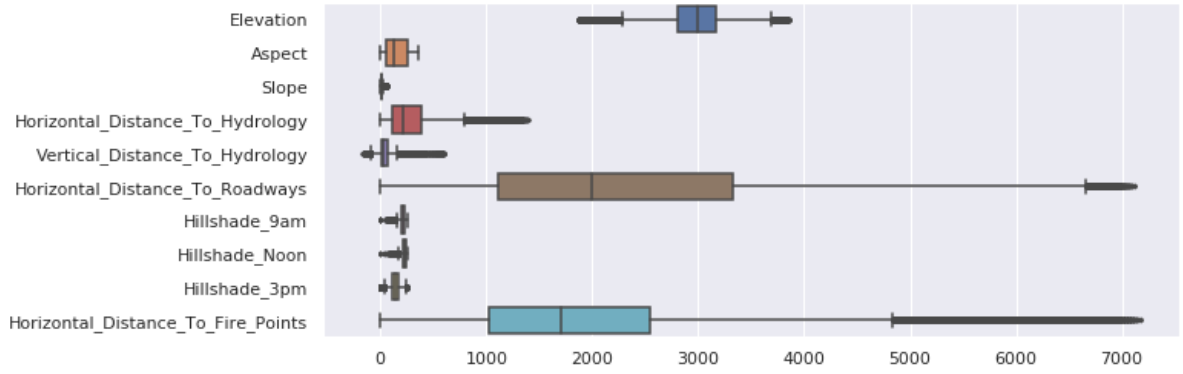
La base de datos que vamos a utilizar en este problema es *Forest Covertype*, la cual se encuentra disponible en `https://archive.ics.uci.edu/ml/datasets/covertype`.

El objetivo es obtener modelos que sean capaces de predecir el tipo de cubierta forestal de una determinada región a partir de sus datos cartográficos. La base de datos proporciona estas observaciones, extraídas de cuatro zonas silvestres situadas en el Bosque Nacional Roosevelt, al norte de Colorado, junto con la cubierta forestal correspondiente. En estas zonas la acción del hombre es mínima, por lo que el tipo de cubierta forestal existente es resultado de procesos naturales.

Estamos, por tanto, frente a un problema de **aprendizaje supervisado**, en concreto de **clasificación multiclase**, donde la variable dependiente es `cover_type`, que toma valores en el rango $\{1, \ldots, 7\}$. Cada uno de los valores codifica un tipo de cubierta forestal distinta.

La base de datos está formada por 581.012 instancias, donde cada una de ellas contiene 12 mediciones cartográficas, que se representan mediante 54 atributos. De ellos, 10 son cuantitativos y 44 son binarios (4 para el zona a la que pertenecen las mediciones y los 40 restantes para el tipo de suelo).

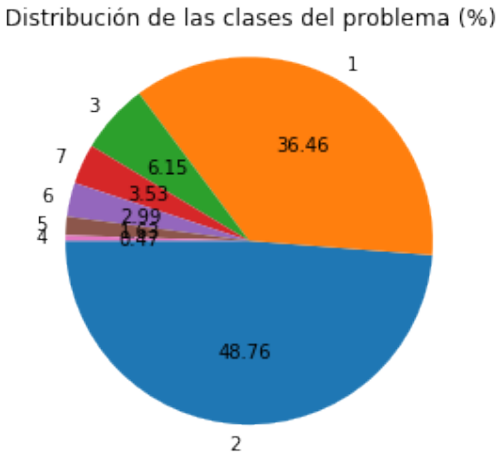
A continuación se muestra un diagrama de caja en el que se pueden observar las distintas distribuciones de los valores en los atributos cuantitativos.



Por tanto, los **elementos del problema** son los siguientes:

- El espacio muestral $\mathcal{X} = \mathbb{R}^{10} \times \{0, 1\}^{44}$
- El conjunto de etiquetas $\mathcal{Y} = \{1, \ldots, 7\}$.
- La función objetivo, $f : \mathcal{X} \to \mathcal{Y}$, que asigna a cada elemento del espacio muestral el tipo de cubierta forestal correspondiente.

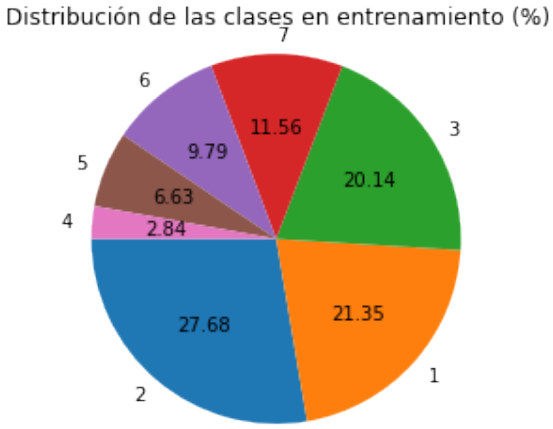
La base de datos no tiene valores perdidos. Por otro lado, las clases del problema están desbalanceadas, como se muestra en la figura inferior. Casi un 50 % de las muestras pertenecen a la clase 2, mientras que las muestras pertenecientes a las clases 4, 5 y 6 suponen menos de un 5 % del total.



2. Particionamiento de los datos

Las 581.012 instancias disponibles se han dividido de la siguientes forma:

- Para el conjunto de **entrenamiento** se ha elegido una muestra aleatoria de 15.500 instancias, de tal forma que la proporción de clases esté algo más equilibrada. En concreto, se han considerado 3310, 4290, 3122, 440, 1028, 1518 y 1792 instancias para cada clase, respectivamente. A continuación se muestra la distribución resultante:

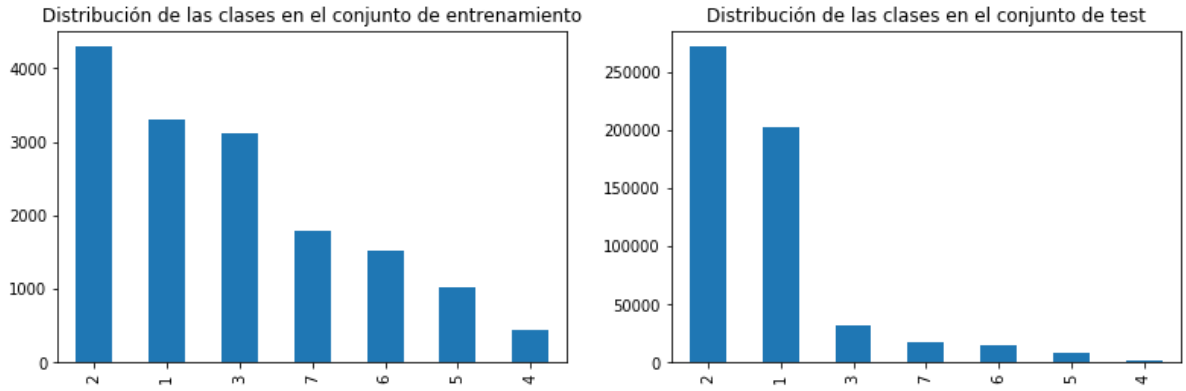


  Si comparamos con lo distribución inicial, se ha reducido la proporción de las clases mayoritarias (1 y 2), que ahora suponen cerca de un 50 % de los datos (y no un 85 %), mientras que el resto de clases conforman el otro 50 % de los datos. De esta forma, se consigue compensar en cierta medida el desbalanceo existente entre las clases sin distorsionar demasiado las proporciones originales.

- Para el conjunto de **validación**, se han escogido otras 15.000 instancias del conjunto restante de datos respetando la distribución de clases, que aunque se encuentra

alterada, es muy similar a la original (el conjunto de entrenamiento supone un 2.6 % de los datos).

- Las instancias restantes se usan como conjunto de **test**, de forma que este y el de validación tienen la misma proporción de clases.



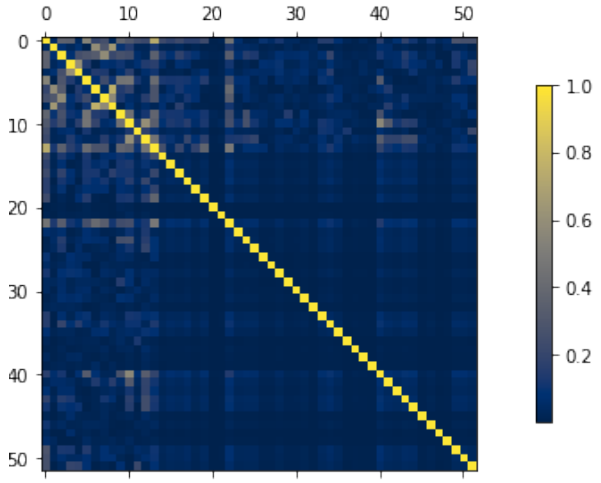
3. Preprocesamiento

A continuación se describe el proceso que se ha seguido para preprocesar los datos.

En primer lugar, se han eliminado los atributos con varianza nula, pues no aportan información al problema. Para ello, se hace uso de la función `VarianceThreshold()` de *scikit-learn*. En este caso, los atributos que se eliminan son `Soil_Type_8` y `Soil_Type_15`, luego pasamos a tener 52 atributos. Así, $\mathcal{X} = \mathbb{R}^{10} \times \{0, 1\}^{42}$.

A continuación, se hace un escalado de los datos, de forma que los valores de cada atributo tengan media 0 y varianza 1, usando la función `StandardScaler()` de *scikit-learn*.

Abajo se muestra la gráfica de correlación de los atributos tras el preprocesamiento.



En vista de la gráfica anterior, y teniendo en cuenta la cantidad de datos disponible y el número de atributos considerados, no se considera necesario usar un método de reducción de la dimensionalidad (PCA, Random Projection, ...).

4. Modelos considerados

A continuación detallamos los modelos que se han considerado para resolver el problema. En primer lugar, probamos con modelos lineales por su sencillez. En vista de que un modelo lineal no es suficiente para este problema, incorporamos dos modelos más complejos: SVM y MLP. Por último, probamos a usar Random Forest, pues pensamos que este tipo de algoritmo es el más adecuado para el problema en cuestión.

4.1. Modelos lineales

Como primera aproximación hemos escogido tres modelos lineales: Perceptrón, Regresión Logística y SVM lineal. Para todos ellos se usa la clase `SGDClassifier` de *scikit-learn*, la cual hace uso del algoritmo de Gradiente Descendiente Estocástico (SGD) para ajustar un modelo lineal.

Como estamos frente a un problema de clasificación multiclase, se usa la técnica de clasificación *One-vs-Rest*. Así, disponemos de un clasificador binario para cada clase,

$$h_i(x) = w_i^\top x + b_i, \quad w_i \in \mathbb{R}^{52},\ b_i \in \mathbb{R} \quad \forall i = 1, \dots, 7$$

de forma que el clasificador i-ésimo permita distinguir entre la clase i-ésima y el resto de clases. Para decidir la clase de una instancia, se elige la del clasificador que haya proporcionado una mayor respuesta, por lo que la hipótesis final será de la forma

$$h(x) = \underset{i=1,\dots,7}{\arg\max}\, h_i(x).$$

Dependiendo de la función de pérdida que se indique en el parámetro `loss` de `SGDClassifier`, obtendremos un tipo de clasificador distinto.

- Para el **Perceptrón** se usa `loss='perceptron'`, que se corresponde con la función de pérdida
$$\mathcal{L}(y_i, f(x_i)) = \text{máx}(0, -y_i f(x_i)))$$
- Para **Regresión Logística** indicamos `loss='log'`, de tal forma que
$$\mathcal{L}(y_i, f(x_i)) = \log(1 + e^{-y_i f(x_i))})$$
- Por último, con `loss='hinge'` obtenemos un **SVM lineal**, pues la función de

pérdida viene dada por

$$\mathcal{L}(y_i, f(x_i)) = \text{máx}(0, 1 - y_i f(x_i)))$$

En todos los casos anteriores usamos el parámetro `class_weight=balanced`, para que los errores en las clases minoritarias se penalicen más que en las mayoritarias, combatiendo de esta forma el desbalanceo que habíamos comentado anteriormente.

Se usa también el parámetro `penalty='l2'` para que el tipo de regularización aplicado sea L_2, dado por

$$\mathcal{R}(w) = \|w\|_2^2$$

Así, para ajustar los parámetros de cada modelo, se minimiza la función de error correspondiente, dada por

$$E(w, b) = \frac{1}{n} \sum_{i=1}^{n} \mathcal{L}(y_i, f(x_i)) + \alpha \mathcal{R}(w)$$

El hiperparámetro α, que controla la cantidad de regularización aplicada, se ajustará usando validación cruzada, como se explica más adelante.

Para el resto de parámetros de `SGDClassifier` se usan los valores por defecto.

4.2. Support Vector Machine (SVM)

Para la implementación de máquinas de vectores de soporte utilizamos la clase `SVC` de *scikit-learn*.

Consideramos 2 tipos de núcleos:

- Polinómico de grado 3, dado por $(\gamma \langle x, x' \rangle)^3$.
- RBF, dado por $\exp(-\gamma \|x - x'\|^2)$

Al usar el parámetro `gamma='scale'`, γ se calcula como 1 / (n_features $\cdot$ Var(X))

De nuevo, se usa el enfoque de clasificación *One-vs-Rest*, así como el parámetro `class_weight='balanced'`, descritos anteriormente.

El resto de valores se mantiene por defecto, mientras que el hiperparámetro que controla la cantidad de regularización (C), se elegirá mediante validación cruzada.

4.3. Perceptrón multicapa (MLP)

En este caso usamos la clase `MLPClassifier` de *scikit-learn*, que implementa un clasificador basado en un perceptrón multicapa.

En concreto, se usa una arquitectura con 3 capas y un número de neuronas por capa en

el rango 50-100, que seleccionaremos mediante validación cruzada más adelante. Como función de activación se usa *ReLU*, y como optimizador *Adam*. La tasa de aprendizaje inicial se fija en 0,001. El hiperparámetro que controla la cantidad de regularización L_2 que se aplica (`alpha`) se ajustará también mediante validación cruzada.

4.4. Random Forest

Random Forest es una técnica de *ensemble* que consiste en ajustar varios árboles de decisión usando distintos subconjuntos de los datos de entrenamiento. Para clasificar una muestra, se usa un sistema de voto por parte de los clasificadores, de tal forma que la clase más elegida sea la que se escoja finalmente. Esto reduce la varianza del modelo y controla el sobreajuste, lo que se traduce en un aumento de la precisión.

Para implementarlo utilizamos la clase `RandomForestClassifier` de *scikit-learn*.

Se van a probar distintos modelos en función del número de árboles considerados: para 100, 200 y 500 árboles se usará el conjunto de entrenamiento completo para ajustar cada árbol. Cuando el número de árboles sea elevado (750 y 1000), se usará un subconjunto de 10000 muestras, para evitar problemas de memoria.

Se usa el parámetro `class_weight='balanced'` para dar más importancia a las clases minoritarias. El resto de parámetros se mantiene por defecto.

5. Métrica del ajuste

Debido al desbalanceo presente entre las distintas clases, se ha decidido utilizar el coeficiente de correlación de Matthews (MCC) para evaluar los modelos. El MCC se considera en general una medida equilibrada que puede utilizarse incluso si las clases son de tamaños muy diferentes [1].

Para problemas de clasificación multiclase, el MCC puede definirse en términos de los elementos presentes en la matriz de confusión C, como

$$MCC = \frac{c \times s - \sum_k p_k \times t_k}{\sqrt{(s^2 - \sum_k p_k^2) \times (s^2 - \sum_k t_k^2)}}$$

donde, en la expresión anterior

- $t_k = \sum_i C_{ik}$ es el número de instancias de la clase k.
- $p_k = \sum_i C_{ki}$ es el número de veces que se predice la clase k.
- $c = \sum_k C_{kk}$ es el número total de instancias clasificadas correctamente.
- $s = \sum_i \sum_j Cij$ es el número total de instancias.

Así, si el MCC vale 1, estamos ante una clasificación perfecta, mientras que el valor

0 equivale a una predicción aleatoria. Pueden darse valores negativos, que indican una clasificación inversa. Por tanto, cuanto más cerca de 1 esté el MCC, más preciso será el modelo considerado.

6. Regularización

Como se ha comentado en el apartado 4, todos los métodos que usamos incorporan regularización, pues es esencial en el campo del aprendizaje automático, ya que la posibilidad de sobreajuste es muy elevada cuando tratamos con tantas variables de entrada.

En este caso hemos utilizado regularización *ridge*, o L_2, que consiste en añadir una penalización cuadrática en los pesos a la función de pérdida. Este tipo de regularización es útil cuando varios atributos están correlados entre sí, y la mayoría son relevantes, luego pensamos que es el más adecuado para nuestro problema. Además, algunos de los modelos considerados solo soportan este tipo de regularización.

$$\mathcal{L}_{reg}(w) = \mathcal{L}(w) + \alpha\|w\|^2.$$

El término de regularización controla la complejidad del modelo resultante, haciendo que los coeficientes aprendidos sean pequeños. La cantidad de regularización aplicada viene determinada por α, de tal forma que valores mayores indiquen una mayor penalización.

La elección del parámetro α, que en términos prácticos regula el efecto de la regularización, será el principal hiperparámetro a ajustar en la selección de modelos, como se verá en el siguiente apartado.

7. Selección de modelo

Para seleccionar el mejor modelo final de entre los que hemos considerado se ha seguido la siguiente estrategia:

1. Partimos de un conjunto de diferentes modelos lineales, SVM, MLP y Random Forest, considerando distintos valores para los hiperparámetros.
2. Sobre cada uno de dichos conjuntos, se ha elegido el *mejor modelo* utilizando validación cruzada con 5 particiones (ver figura 7.1), obteniendo así el mejor modelo lineal, el mejor SVM, el mejor MLP y el mejor Random Forest (de entre los considerados).
3. Entrenamos nuestros 4 candidatos a modelo final sobre el conjunto de entrenamiento completo.
4. El modelo seleccionado finalmente será aquel que obtenga un mayor resultado en el conjunto de validación de entre los 4 seleccionados.

En todo el proceso, se usará como métrica el coeficiente de correlación de Matthews, definido anteriormente.

Para la validación cruzada se han utilizado tanto la clase `cross_val_score` como la clase `Kfold` de `scikit-learn`.

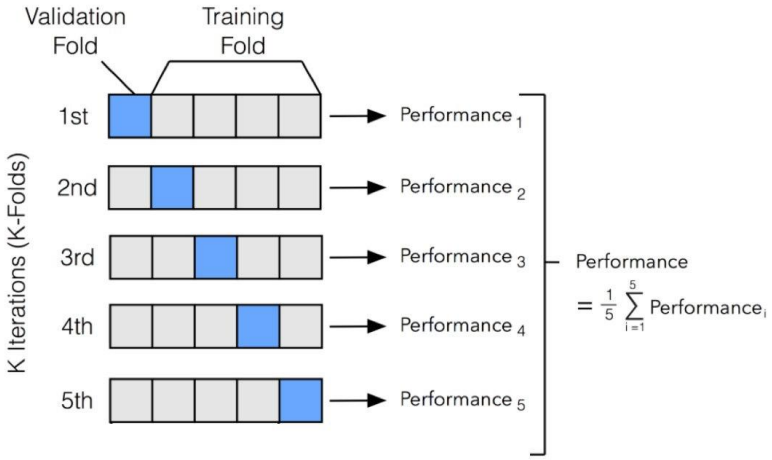


Figura 7.1: Ilustración 5-fold cross-validation

El objeto de la clase `Kfold` nos devolverá los subconjuntos sobre los que realizar validación cruzada (utilizamos la misma semilla en todas las llamadas para generar los mismos subconjuntos en todos los modelos) y el objeto de la clase `cross_val_score` realiza la validación cruzada automáticamente pasándole el objeto `Kfold` como parámetro. En concreto, utilizamos `StratifiedKfold`, que mantiene la proporción de clases en los subconjuntos generados.

7.1. Selección del mejor modelo lineal

Para cada modelo lineal de los explicados en la sección 4, se han probado distintos valores para el parámetro `alpha`, que controla la cantidad de regularización que se aplica. Los resultados obtenidos en validación cruzada con cada uno de estos valores se recogen en la siguientes tablas.

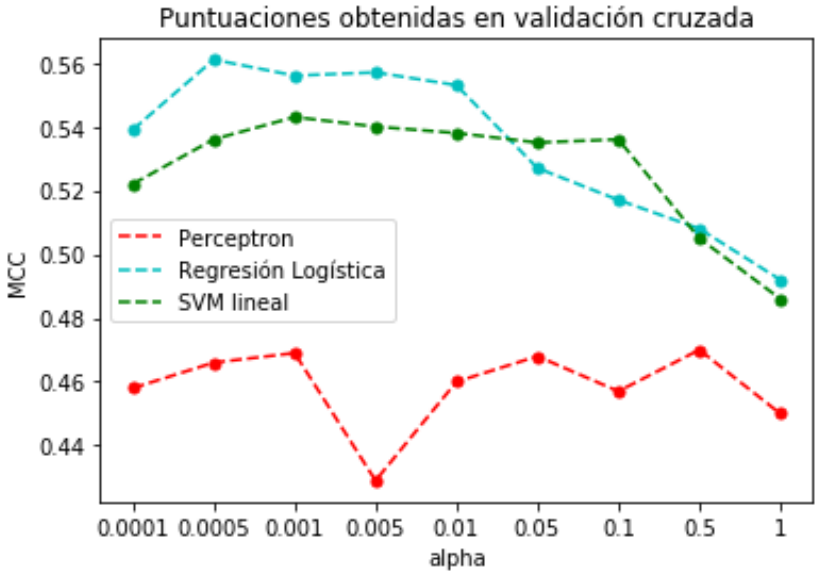
El tiempo de ejecución empleado en evaluar todos los modelos anteriores en validación cruzada ha sido 19.85 segundos, bastante bajo si tenemos en cuenta que se han ajustado 27 modelos en 5 conjuntos de datos distintos.

En la figura inferior se visualizan las puntuaciones anteriores. Se puede ver que el modelo que mejores resultados obtiene es el de Regresión Logística con $\alpha = 0{,}0005$. Aún así, las puntuaciones obtenidas no son muy altas, lo que indica que un modelo lineal no es suficiente para este problema.

Perceptrón	
α	MCC
0.0001	0.458
0.0005	0.466
0.001	0.469
0.005	0.429
0.01	0.460
0.05	0.468
0.1	0.457
0.5	**0.470**
1	0.450

Regresión Logística	
α	MCC
0.0001	0.539
0.0005	**0.561**
0.001	0.556
0.005	0.557
0.01	0.553
0.05	0.527
0.1	0.517
0.5	0.508
1	0.492

SVM lineal	
α	MCC
0.0001	0.522
0.0005	0.536
0.001	**0.543**
0.005	0.540
0.01	0.538
0.05	0.535
0.1	0.536
0.5	0.505
1	0.486



Por tanto, el mejor modelo lineal (que será el que se considere como representante de los modelos lineales) es el siguiente:

```
SGDClassifier(alpha=0.0005, class_weight='balanced', loss='log',
n_jobs=-1, random_state=1997)
```

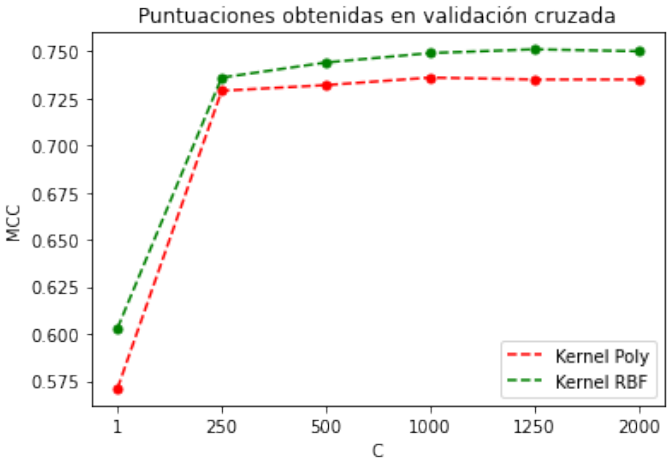
7.2. Selección del mejor SVM

Tanto para el núcleo polinómico (de grado 3) como para el núcleo RBF, se han probado 6 valores distintos del hiperparámetro de regularización `C` (en este caso, el valor de C es inversamente proporcional a la cantidad de regularización aplicada). Estos valores, junto con los resultados obtenidos en validación cruzada, se muestran en la tabla inferior.

En este caso, el tiempo de ejecución para ajustar los 12 SVM considerados en validación

cruzada ha sido de 281.48 segundos, esto es, algo menos de 25 segundos por modelo.

Kernel	C	MCC
Poly	1	0.571
Poly	250	0.729
Poly	500	0.732
Poly	1000	0.736
Poly	1250	0.735
Poly	2000	0.735
RBF	1	0.603
RBF	250	0.736
RBF	500	0.744
RBF	1000	0.749
RBF	**1250**	**0.751**
RBF	2000	0.750



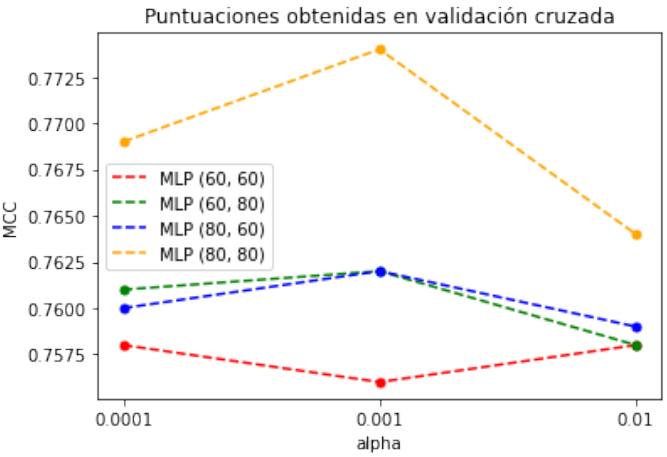
En la gráfica se observa que estos modelos ofrecen puntuaciones más elevadas, superando con creces a las obtenidas con los modelos lineales. El mejor modelo obtenido es aquel que utiliza kernel RBF y un valor del parámetro de `C=1250`:

```
SVC(C=1250, class_weight='balanced')
```

7.3. Selección del mejor perceptrón multicapa

Para esta clase de modelos se han considerado como hiperparámetros tanto el número de neuronas en cada una de las dos capas ocultas como el valor del parámetro de regularización α. En total, se han considerado 12 modelos, obteniendo los siguientes resultados en validación cruzada:

Capa 1	Capa 2	α	MCC
60	60	0.0001	0.758
60	60	0.001	0.756
60	60	0.01	0.758
60	80	0.0001	0.761
60	80	0.001	0.762
60	80	0.01	0.758
80	60	0.0001	0.760
80	60	0.001	0.762
80	60	0.01	0.759
80	80	0.0001	0.769
80	**80**	**0.001**	**0.774**
80	80	0.01	0.764



Los resultados obtenidos con estos modelos también son buenos, pero cabe destacar que el tiempo de ajuste aquí es elevado: se han necesitado unos 18 minutos para ajustar los 12 modelos en validación cruzada, lo que nos da un tiempo por modelo superior al minuto, que ya es algo a tener en cuenta.

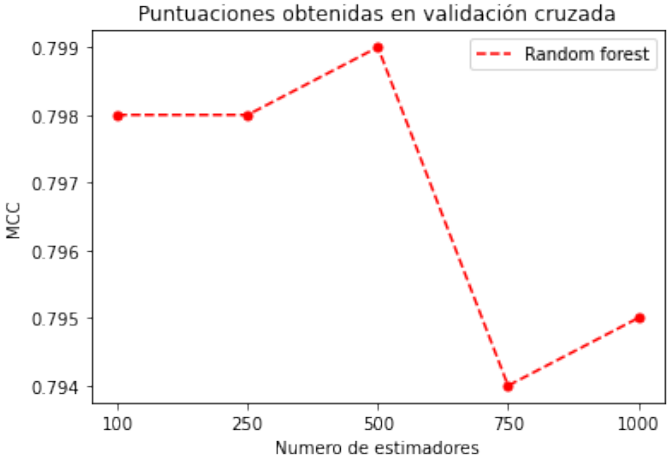
El mejor modelo de esta clase es aquel que tiene 80 neuronas por capa y una constante de regularización de `alpha=0.001`:

```
MLPClassifier(alpha=0.001, hidden_layer_sizes=(80,80), random_state=seed)
```

7.4. Selección del mejor random forest

En este caso se han considerado distintos modelos de *random forest* en función del hiperparámetro que controla el número de estimadores (árboles) a utilizar. Como se indicó antes, para los valores más elevados se utiliza un subconjunto 10.000 elementos para cada estimador en lugar de todo el conjunto de entrenamiento, por eso en la tabla inferior se incluye la columna 'Reducción'.

Redución	Estim.	MCC
No	100	0.798
No	250	0.798
No	**500**	**0.799**
Si	750	0.794
Si	1000	0.795



Podemos ver como estos modelos ofrecen puntuaciones por encima de todos los que se han probado anteriormente. Además, el tiempo de ejecución es inferior al obtenido con SVM y MLP: unos 67 segundos en total, lo que supone unos 13 segundos por modelo.

El mejor modelo de esta clase ha resultado ser aquel que considera 500 estimadores, utilizando el conjunto de entrenamiento en su totalidad para ajustar cada uno de ellos:

```
RandomForest(class_weight='balanced', n_estimators=500, random_state=seed)
```

7.5. Selección del modelo final

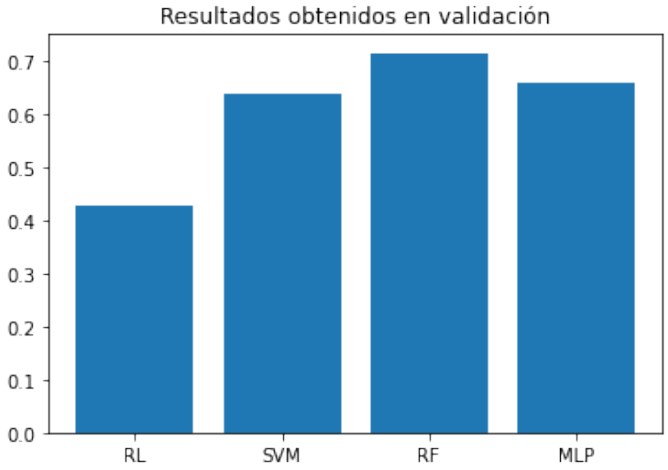
Notemos que un criterio de selección del modelo final podría haber sido tomar aquel con mayor puntuación en la validación cruzada, pues los subconjuntos utilizados son

los mismos para todos los modelos. Sin embargo, consideramos como mejor alternativa entrenar el representante de cada clase en todo el conjunto de entrenamiento y seleccionar como modelo final aquel con una mayor puntuación en el conjunto de validación, que no habíamos usado hasta ahora.

Recordemos que la distribución de clases en el conjunto de validación no ha sido alterada (como en el de entrenamiento), por lo que es similar a la del conjunto de test, luego mas representativa del error que tendremos en este.

Tras ajustar los 4 modelos seleccionados en la etapa anterior en el conjunto de entrenamiento y evaluarlos en el de validación, se han obtenido los siguientes resultados:

Modelo	MCC
Regresión logistica	0.4284
SVM	0.6389
Random forest	**0.7165**
MLP	0.6600



Como podemos ver, aunque la diferencia en las puntuaciones en validación cruzada no era tan notable, al utilizar un conjunto de validación con la distribución de clases original, se aprecian diferencias más notables entre ellos. Podemos confirmar que el mejor modelo de los considerados ha sido el de Random Forest:

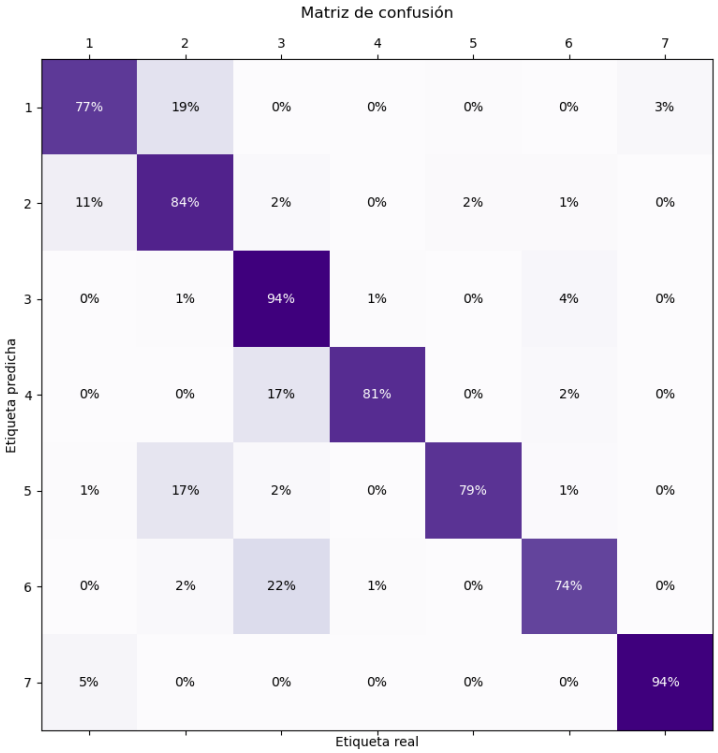
```
RandomForest(class_weight='balanced', n_estimators=500, random_state=seed)
```

8. Resultados obtenidos

Tras seleccionar el mejor modelo, se evalúa en el conjunto de test, formado por 550.512 instancias, en el que podemos analizar con más exactitud la capacidad de generalización del modelo.

El MCC obtenido por el modelo *Random Forest* en el conjunto de test es de **0.71**. Como podemos ver, es bastante parecido al que hemos obtenido en validación, pero bastante inferior al que se había obtenido en validación cruzada.

A continuación se muestra la matriz de confusión obtenida en el conjunto de test:



Para un análisis más profundo, podemos ver los valores obtenidos en diferentes métricas y en cada una de las clases en la siguiente tabla:

class	precision	recall	f1-score	support
1	0.84	0.77	0.81	202999
2	0.85	0.84	0.84	271610
3	0.75	0.94	0.83	31766
4	0.81	0.81	0.81	2246
5	0.53	0.79	0.63	8240
6	0.67	0.74	0.70	15429
7	0.69	0.94	0.80	18222
accuracy			0.82	550512
macro avg	0.73	0.83	0.78	550512
weighted avg	0.83	0.82	0.82	550512

Como podemos ver, el modelo escogido es más que suficiente para el problema en cuestión. Los errores más comunes se dan en clasificar instancias de la clase 3 como pertenecientes a las clases 6 y 4, y en clasificar instancias de la clase 2 como de la clase 5 y 1,

lo que se refleja en la precisión y el recall mostrados en la tabla superior. Podemos comprobar que las clases minoritarias se han aprendido bien, a pesar de la enorme diferencia en el número de instancias (sobre todo las clases 4 y 5). En las clases mayoritarias (1 y 2) también se alcanzan buenas puntuaciones, teniendo en cuenta que se han probado en muchos más casos que la minoritarias.

9. Conclusiones

Podemos concluir indicando que se ha elegido el mejor modelo de entre los considerados, gracias a las técnicas descritas en los apartados anteriores.

En cuanto a los resultados obtenidos, son bastante aceptables si tenemos en cuenta que se ha entrenado con el 2.6 % de las instancias presentes en el conjunto de datos, y se ha alcanzado una precisión superior al 80 %.

Además, en vista de que los resultados obtenidos en cada clase son relativamente parejos, podemos concluir que aunque los conjuntos se encuentren claramente desbalanceados, el modelo ha conseguido aprender de todas las clases con una eficacia similar, logrando así los objetivos que buscábamos en el modelo.

10. Bibliografía

Referencias

[1] Matthews correlation coefficient
https://en.wikipedia.org/wiki/Matthews_correlation_coefficient