Universidad De Granada

Apredizaje Automático

Proyecto Final

Pedro Bonilla Nadal
Antonio Martín Ruiz

18 de junio de 2020

Índice

1. Comprensión del problema a resolver

Este dataset contiene datos de la oficina del censo[1] relativos a censos de 1995, obtenidos del repositorio UCI de bases de datos [2]. Un total de 48842 personas han sido encuestadas para este censo. De estas personas tenemos una serie de variables con información de carácter socioeconómico. En particular:

- Tenemos 6 variables de tipo numérico u ordinal, con valores en rangos distintos.
- 8 variables de tipo categórico.
- Una variable de clase que toma como valores < 50K y > 50k.
- Es un problema de clasificación, ya que no tenemos información para hacer una regresión sobre la ganancia.

El código utilizado para la resolución de la práctica se encuentra en el archivo `main.py`. Del mismo modo, los datos limpiados se encuentran en el archivo `adults.data` y su información relativa procesada en el archivo `adult.names`.

2. División y codificación de los datos

A la hora de obtener los datos en el respositorio se encuentran tres archivos: `adult.data`, `adult.test` y `adult.names`. Estos datos fueron procesados en el año 1996. Nosotros realizamos una operación de formato de los datos para ajustarlos a convenciones más recientes.

- Cambiamos la variable de clase por valores categóricos 0 para < 50K y 1 para > 50K.
- Eliminamos el espaciado después de la coma para facilitar la lectura por parte de librerías actuales.
- Añadimos el flag `@attribute` a la lista de atributos provista en `adult.names`.

Utilizaremos la división train/test que viene establecida por el propio dataset. La proporción de clases es parecida en ambos conjuntos y la cantidad de datos en el training establecido es la suficiente para el aprendizaje. Es por ello que no modificamos esta partición. Además, a la hora de validar la selección de hiperparámetros utilizaremos la técnica cross validation, para la cual dividiremos el conjunto de training en 5 subconjuntos. Se utilizará para esta la función `cross_validate`[4].

Como último detalle de codificación de los datos durante la parte de preprocesado aplicamos la codificación sugerida por scikit-learn mediante `dummy_variables`. Mediante este proceso se transforman las variables categóricas en numéricas dale un valor natural diferente a la variable para cada tipo de valor que tomara anteriormente.

3. Preprocesado

Mostraremos a continuación las siguientes técnicas de preprocesado realizadas. Justificaremos individualmente su uso.

- Normalización de las variables. Realizaremos una normalización de los datos, para evitar que su escala afecte a la relevancia de estos. Para ello usaremos la función `StandardScaler`[3] provista en la librería `sklearn`.
- Valoración del interés de la variables. Para esto vamos a realizar dos técnicas:

- El conjunto de datos presenta 105 columnas. Por ello vamos a intentar reducir aquellas que presenten poca o ningúna información haciendo un análisis por varianza, y eliminando aquellas columnas cuya varianza sea inferior a 0,1. Esto lo realizaremos con el algoritmo `varianceThreshold`[9].

Proceso	Dimensionalidad Resultante
`Dummy_Variables`	105
`varianceThreshold`	57

Tabla 1: Dimensionalidad de la muestra tras cada proceso.

Podemos observar como tenemos una gran cantidad de variables que aportan poca o ninguna información. Observando la muestra atentamente veremos que se debe a variables como `nacionality`, que tras ser codificadas como dummy variables quedan como columnas nacionalidades con muy pocos positivos. Esto es negativo porque da pie a que algunos algoritmos (en particular los basados en árboles) sobreajusten a esta submuestra en particular.

- Además realizamos un estudio `PCA` con el objetivo de avergiguar cuales son las columnas que mejor explican la variabilidad de la base de datos. Para ello ejecutamos un predictor `PCA`, así como seleccionar las variables principales. Este nos devuelve una estimación de la variabilidad que explica cada variable, obteniendo como resultado:

```
[4.49573987e+00 2.94835042e+00 2.37099181e+00
2.01320755e+00 1.76975896e+00 1.75675195e+00
1.68179310e+00 1.42475478e+00 1.38339986e+00
1.32481406e+00 1.28634456e+00 1.24730223e+00
1.19496365e+00 1.17428828e+00 1.15682951e+00
1.13753027e+00 1.12467327e+00 1.11347522e+00
1.09735994e+00 1.07570368e+00 1.07225031e+00
1.06723994e+00 1.03858652e+00 1.02811739e+00
1.02497765e+00 1.01754201e+00 1.01319671e+00
1.00978578e+00 1.00228665e+00 9.92216208e-01
9.86143567e-01 9.81570539e-01 9.65009215e-01
9.52551640e-01 9.38447568e-01 9.16648964e-01
8.97850717e-01 8.77922999e-01 8.31440082e-01
7.68450694e-01 7.55866947e-01 7.14349376e-01
6.89593020e-01 6.14759178e-01 5.64426417e-01
5.22746091e-01 4.35534961e-01 4.01351472e-01
7.00012064e-02 4.35859769e-02 2.11241135e-02
5.56548322e-03 1.52954845e-03 1.03141699e-03
1.72725144e-05 1.17471416e-31 9.16296960e-33]
```

Figura 1: Resultado algoritmo PCA

Como podemos observar tenemos una diferencia significativa con respecto a las dos últimas variables, por lo que decidimos desestimarlas. Con esto obtenemos loas resultados.

Proceso	Dimensionalidad Resultante
`varianceThreshold`	57
`PCA`	55

Tabla 2: Dimensionalidad de la muestra tras cada proceso.

- Polynomial Freatures. Con especial interés en el caso lineal, el uso de variaciones polinómicas de los datos puede ser útil para permitir aproximaciones a clases de funciones nuevas. Probaremos variación cuadrática sobre las variables numéricas cuando realizemos un ajuste lineal. Para random forest y perceptrón multi capa no lo consideramos necesario por la ya conocida complejidad de los modelos, y para SVM utilizaremos variaciones de kernel.

- Valoración de las variables de interés. Para realizar un estudio de las variables de interés. Este ya lo hemos visto mostrado en

- Datos incompletos y valores perdidos. En relación a los valores perdidos encontramos en tres variables de tipo categórico.

Variable	Valores distintos	Valores Perdidos
`workclass`	9	2799
`occupation`	15	2809
`native-country`	42	857

Tabla 3: Representación de valores perdidos.

  Realizamos la sustitución de estos datos mediante la función `replace_lost_categorical_values`. En esta, para cada columna, calculamos su distribución de probabilidad, esto es, sumamos las ocurrencias de cada categoría y dividimos entre el total de valores con valores no perdidos. Obtenemos así la probabilidad de cada categoría. A continuación calculamos las probabilidades acumuladas sumando para categoría su probablidad y la de todas las anteriores. Para cada dato perdido generamos un número aleatorio mediante una distribución uniforme en el intervalo $[0, 1]$. La clase por la que el valor perdido será sustituida será la de cuyo intervalo contenga al valor generado, siendo el intervalo de cada clase el comprendido entre la probabilidad acumulada de la anterior y su probabilidad acumulada (incluyendo en cada uno su extremo inferior, pero no su extremo superior).

- Datos inconsistentes. No encontramos datos inconsistentes.

- Balanceo de clases. Nos encontramos ante una situación con un desbalanceo notable. Sin embargo, dada la cantidad de datos provista por la base de datos creemos que no es necesario realizar modificaciones de los datos ni en el conjunto de train ni en el de test.

4. Métricas

Hemos decidido considerar, en este contexto de clasificación, dos funciones con objetivo de medir el error, ambas halladas en [6].

- *accuracy*. Decidimos utilizar esta medida como principal por su simplicidad y expresividad. Esta métrica nos propocionará una idea general de la bondad de nuestro modelo en un caso general, así como nos permitirá comparar, por ejemplo, con el clasificador modal.

- *f1-score.* Al estar en un modelo de clases desbalanceadas, consideramos que debiamos utilizar además una métrica que penara comportamientos de 'asignación modal'. La medida F_1 se define como[7]

$$F_1 = \frac{2}{\text{recall}^{-1} + \text{precision}^{-1}} = 2 \cdot \frac{\text{precision} \cdot \text{recall}}{\text{precision} + \text{recall}}.$$

  donde precisión es (en términos de clase positiva/negativa) el número de verdaderos positivos entre los positivos escogidos, y recall es el número de positivos escogidos entre los positivos totales. En este caso la medida que mostramos es la media obtenida de las f1-score obtenidas de considerar cada una de las clases como la positiva.

5. Modelos Considerados

5.1. Modelo lineal

Como modelo lineal, dado que estamos en clasificación, podemos escoger de los modelos explicados en clase entre Regresión Logística y PLA. Decidimos utilizar regresión logística por una serie de motivos.

- Al estar en un dataset que proviene de un censo sucede que podemos encontrar una gran cantidad de outliers y datos algo ruidosos. Esto hará que el perceptron sea contraindicado, dado que dificilmente podremos conseguir convergencia de este debido al ruido, y en el caso de que uno de estos elementos ruidosos sea además un outlier podría variar notable (e indebidamente) el plano clasificador.
- Nos resulta interesante el hecho de que podamos obtener la confianza de una predicción y no solo la predicción.

Para la aplicación de la regresión logistica haremos uso de la función de sklearn[12] .

5.2. Random Forest

El principal objetivo que justifica la aplicación de esta técnica generar un modelo complejo que represente con mucha precisión el conjunto de entrenamiento. Esto nos permitirá su comparación con otros modelos más sencillos que produzcan un ajuste menos exacto de los datos. Mediante esta comparación podremos comprobar la incidencia del problema del overfitting en este caso.
Para la aplicación de random forest haremos uso de la función de sklearn[13].

5.3. SVM

Consideramos que la técnica SVM-soft margin puede ser muy útil para este problema debido a que podemos fijar a placer el grado de compromiso entre busqueda del plano óptimo y la tolerancia al ruido, ambas cosas resultando de mucha utilidad a la hora de trabajar con datos potencialmente ruidosos.

Otro motivo para decidir ajustar este modelo, aunque de carácter menos importante, es la curiosidad suscitada por este modelo durante el desarrollo teórico de la asignatura.
Para la aplicación de support vector machine haremos uso de la función de sklearn[14].

5.4. Modelo extra. Perceptrón multi capa

Decidimos desarrollar un modelo extra además de los obligatorios. Elegimos el modelo de perceptrón multi capa. Con este añadimos otro modelo a la comparación, en este caso calculando una función no lineal y con una complejidad alta del modelo. Aunque no nos encontramos ante un problema de una dimensionalidad muy alta nos servirá para comprobar como funcionan modelos más complejos en este problema.
Para la aplicación de MLP haremos uso de la función de sklearn[15] para clasificación mediante este modelo.

6. Técnica de Ajuste para el Modelo Lineal

Para este conjunto utilizaremos dos técinas de ajuste, ambas definidas como variaciones especializadas sobre el algoritmo SGD.:

- `lbfgs`. Utiliza una variación del método Newton visto en el bonus de la práctica 1 que en lugar de la matriz Hessiana utiliza una aproximación a esta. Es eficiente en comparación con el método de newton estandard y se podría definir como una variación del gradiente descendente estocástico. Como desventajas tiene que se podría acercar a puntos críticos que no fuese mínimos. Esta técnica no admite regularización `l1`.

- `liblinear`. Es el ganador ICML 2008 large-scale learning challenge. Lo utilizamos para probar las regularizaciónes dado que funciona tanto con `l1` como con `l2`.

  Liblinear utiliza técnicas de descenso coordinado[8] donde minimizamos la función en torno a cada eje, resolviendo un problema de minimización unidimensional en bucle.

Estas dos técnicas se pueden entender como variaciones del gradiente descendiente estocástico en esecia, dado que surgen de su filosofía de realizar aproximaciones sucesivas al mínimo. Pese a ello se debe comprender que se parecen en poco más que filosofía. Son técnicas mucho más eficientes que el algoritmo de la pseudoinversa, cosa que nos puede resultar interesante, aunque trabajamos con un dataset que no es particularmente grande. Las compararemos y comentaremos sus resultados en la sección de hiperparámetros y selección del modelo, pero elegiremos `lbfgs` dado que para el tamaño de nuestro dataset no es relevante la mejora en eficiencia de `liblinear` y `lbfgs` consigue un resultado ligeramente mejor.

7. Hiperparámetros y selección del modelo

En esta sección estudiaremos la aplicación de las técnicas especificando claramente qué algoritmos se usan en la estimación de los parámetros, los hiperparámetros y el error de generalización.

El primer modelo que presentamos es el modelo lineal. Este fue el primer modelo ajustado y de su ajuste aprovechamos conocimiento para el ajuste de los otro modelos.

7.1. Modelo lineal

Como hiperparámetro vamos a considerar la prueba de las dos distintas técnicas de ajuste, las dos técnicas de regularización, así como la fuerza de regularización. También consideraremos una muestra con variables lineales y otra con variables polinómicas.

- Variables polinómicas. Como primera aproximación al problema estidiamos, con un estimador base, la mejora supuesta al aumentar la dimensionalidad del dataset al incluir variables polinómicas. Con esto pretendemos ver si el modelo lineal es suficiente para

explicar la variabilidad de la muestra. Obtenemos como resultados las siguientes estimaciones[1]:

Grado	Accuracy	F1
1	0.85061	0.78141
2	0.85061	0.78141

Tabla 4: Resultados ajustes con variables polinómicas.

Lo primero que observamos es que, debido a la cantidad y dimensionalidad de los datos sucede que para generar una base de datos con cobinaciones cúbicas creamos un conjunto de 8.30 GB que no somos capaces de alojar en la RAM del procesador. Es por eso que, por las limitaciónes técnicas que poseemos no consideramos más grado[2]. De todos modos esta limitación no nos afecta pues:

- Estaríamos en un caso claro de ,aldición de la dimensionalidad[11] a pesar de haber realizado ya una reducción de dimenisonalidad tanto por `varianceThreshold` como por `PCA`.
- Podemos ver que se mantiene invariable la estimación del error. Esto es debido a que el modelo lineal ya se ajusta perfectamente a la variabilidad de los datos.

Es por esto que decidimos no aplicar esta técnica.

- Regularización. Empezamos considerando que regularización debemos usar. Barajamos como opciones regularización `l1` y regularización `l2`. Durante el desarrollo de la asignatura hemos aprendido que cada una de estas penaliza de forma distinta. En particular:

  - `l1`: penaliza si habia variables que podrían no ser relevantes. Esta es mejor en caso de alta dimensionalidad.
  - `l2`: penaliza que todas estas sean relevantes. Esta es mejor en casos de datos con alta cantidad de outliers.

  Utilizando conocimiento del mundo en el que vivimos es fácil imaginar que en un censo económico hay muchos outliers, que consistirán en la gente más privilegiada. Sin embargo no utilizaremos esta información y usaremos datos puramente analíticos. Podemos considerar dos casos:

  - *Outliers de variables categóricas.* Estos han sido gestionados ya, debido al one-hot-encoding realizado durante la codificación del dataset, así como el filtro de varianza, ha hecho que todas las categorías sin un número significativo de personas incluido sean desestimadas.
  - *Outliers de variables ordinales.* Para el estudio de este examinamos una a una las variables, imprimiendo el gráfico[2].Podemos ver que tenemos una gran cantidad de outliers. En particular tenemos distribuciones de cola larga o *Heavy Tail* como son conocidas en el mundo de la estadística. Estas distribuciones son aquellas en las cuales, pese a estar la mayoría de los valores agregados en un espacio pequeño del dominio.

[1] Estimaciones de error con 5Fold-Cross-Validation.

[2] En el código entregado no adjuntamos esta ejecución para evitar entregar un código que no funcione de principio a fin, lo que provocaría la descalificación de la práctica

Por otro lado podemos ver que después del preprocesado, hemos descartado una gran cantidad de variables (casi la mitad) y hemos estudiado con `PCA` que el resto de variables consideradas aportan información para explicar la variabilidad de la muestra. Es por esto que no consideramos que la regularización `l1` pueda aportar mucho al modelo.

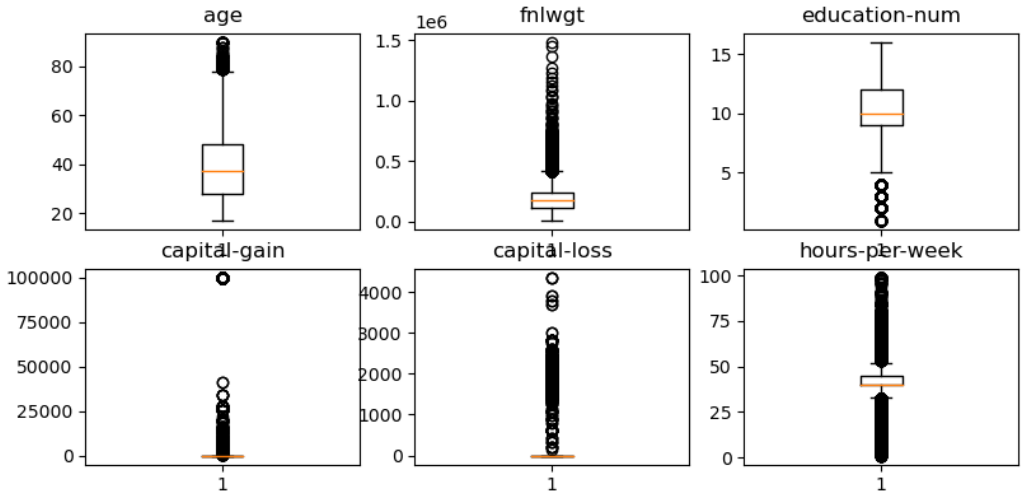

Figura 2: Representación de las variables ordinales.

Debido a esto decidimos usar regularización `l2`.

- Técnica de ajuste. No tenemos ningún argumento en contra o a favor de la técnica de ajuste elegida. Probaremos ambas y eligiremos la que mejor error bondad de ajuste aproxime. Tras las pruebas obtenemos que la mejor técnica el `liblinear`, aunque las diferencias con mínimas.
- Fuerza Regularización. Para seleccionar la fuerza de ajuste realizaremos una representación de esta, con el objetivo de estimar una cantidad óptima. Obtenemos como resultado 7.1. Podemos ver que en valores muy pequeños de va fuerza de regularización, obtenemos un peor resultado, y que se estabiliza a partir de valores mayores. Elegimos como mejor resultado tras realizar una ingeniería de parámetros un poco más exhaustiva, elegimos como valor de la fuerza de regularización 2.

Así obtenemos que el mejor ajuste es:

```
LR( penalty='l2', random_state=0, solver='liblinear', C=1.9)
```

Obtenemos los resultados:

	Accuracy	F1-score
Train	0.85132	0.78230
Test	0.84976	0.77027
Validación	0.85064	0.78150

Tabla 5: Resultados ajustes con variables polinómicas.

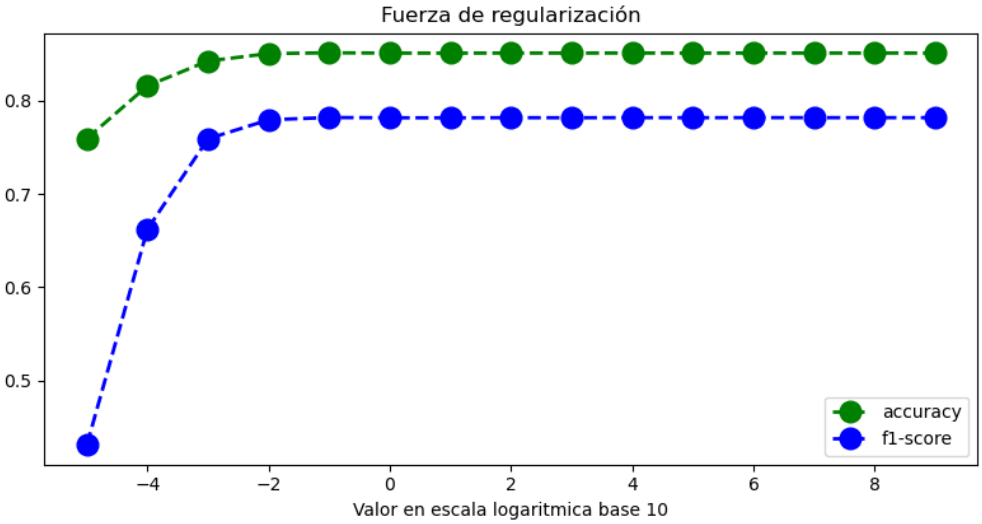

Figura 3: Relación de las métricas de error con la regularización.

7.2. Random forest

A partir del modelo por defecto que proporciona scikit-learn modificaremos tres parámetros para obtener el resultado final

- Número de estimadores. Viene dado por el parámetro `n_estimators`. El número por defecto es 100. Procedemos ampliando este número hasta que dejan de producirse mejoras significativas. En las gráficas vemos cómo evoluciona la puntuación en la predicción dentro de test y en cross validation. Comenzamos con un número pequeño de estimadores y comprobamos que la predicción dentro del test sube hasta llegar a los 100 iteradores, donde ya es prácticamente perfecta. En cross validation observamos que la subida es equivalente hasta los 100 estimadores, pero a partir de aquí la predicción baja. Comprobamos un repunte en 400 estimadores y a partir de este punto podemos considerar que empieza a crecer el overfitting. Obtenemos entonces que el número óptimo es el de 400 estimadores.

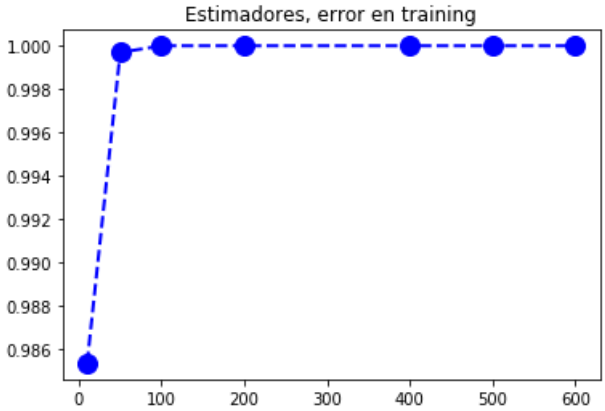

Figura 4: Evolución d ela predicción en train según número de estimadores.

- Criterio de selección. El criterio que se aplica para obtener la calidad de las divisiones. El criterio por defecto es la impureza de Gini, pero también puede seleccionarse la ganancia de información. Puesto que esta segunda opción es la que se ha estudiado en teoría y la que produce mejores resultados, aunque con peores tiempos de ejecución, elegimos

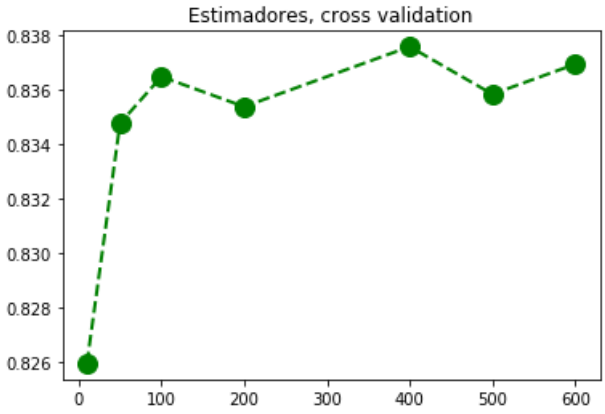

Figura 5: Evolución de la predicción en cross validation según número de estimadores.

este. Realizamos un experimento para esta comparación. Los resultados del mismo se encuentran en la correspondiente tabla.

Criterio	Prueba	Accuracy	F1-score
Gini	Train	0,9999	0,9999
Gini	Validación	0.8390	0,7207
Entropy	Train	0.9999	0.9999
Entropy	Validación	0.8303	0.7241

Tabla 6: Tabla comparativa sobre criterios de selección.

- Profundidad máxima. Es clave para limitar la complejidad del modelo y evitar overfitting. Por defecto la profundidad no está limitada y el árbol se amplía todo lo posible. Elegiremos limitar la profundidad de los árboles ya que obtenemos claras mejoras en clasificación. Procedemos disminuyendo este número hasta que deja de producirse una mejora en la puntuación de cross validation y empieza a empeorar el resultado. En las gráficas correspondientes podemos comprobar cómo la clasificación en training aumenta hasta la profundidad de 40, donde se estanca. En cross validation vemos que la clasificación mejora hasta la profundidad de 10, a partir de la cual empeora. El comportamiento es claro, aumentando la calidad de la predicción conforme aumenta la complejidad del modelo hasta llegar al umbral de la profundidad 40, a partir de la cuál comienza a aumentarse el error por sesgo y por tanto el overfitting. Decidimos por tanto tomar la profundidad 40 como la mejor.

Así obtenemos que el mejor ajuste es:

```
RandomForestClassifier(n_estimators=400, criterion='entropy', max_depth=50,)
```

Obtenemos los resultados:

	Accuracy	F1-score
Train	0,9999	0,9999
Test	0,8307	0,7269
Validación	rellenar	rellenar

Tabla 7: Resultados de Random Forest.

7.3. SVM

Para el ajuste de Support Vector Machine vamos a considerar los parámetros C o fuerza de regularización y kernel. Al no tener clasificación multiclase en este problema no tiene sentido aplicar estudio de varias de las variables. También estudiaremos el parámetro max iter, aunque no lo comentaremos, procuraremos que sea lo suficientemente amplio como para obtener siempre convergencia, pero acotado para asegurar un tiempo de ejecución limitado. En general exigiremos que sea siempre inferior a 15000 iteraciones.

- Kernel. Consideraremos kernel Gaussiano como viene recomendado en el guión así como lineal. No elegimos kernel polinómico debido al estudio ya realizado sobre este tema en Regresión logística.

  Debido al éxito del modelo lineal estamos interesados en saber como se comportará un plano óptimo. Sin embargo, consideramos que la probable no existencia de un plano óptimo, así como la falta de variabilidad en el modelo y los habituales buenos resultados del nucleo gaussiano, hace que este se presente como una mejor alternativa. En efecto:

Nucleo	Accuracy	F1
Lineal	0.775	0.625
Gaussiano	0.852	0.777

Tabla 8: Resultados ajustes con variables polinómicas.

- Fuerza de regularización. Para hacer un ajuste de esta, al igual que en modelo lineal, realizaremos un barrido de valores en escala logarítmica para detectar alguna variación grande, para posteriormente hacer una ingeniería de parámetros más fina. Obtenemos como resultado:

Así obtenemos que el mejor ajuste es:

```
svm.SVC(max_iter=15000, C=2, kernel=kernel)
```

Obtenemos los resultados:

	Accuracy	F1-score
Train	0.85132	0.78230
Test	0.84976	0.77027
Validación	0.85064	0.78150

Tabla 9: Resultados ajuste final regresión logística.

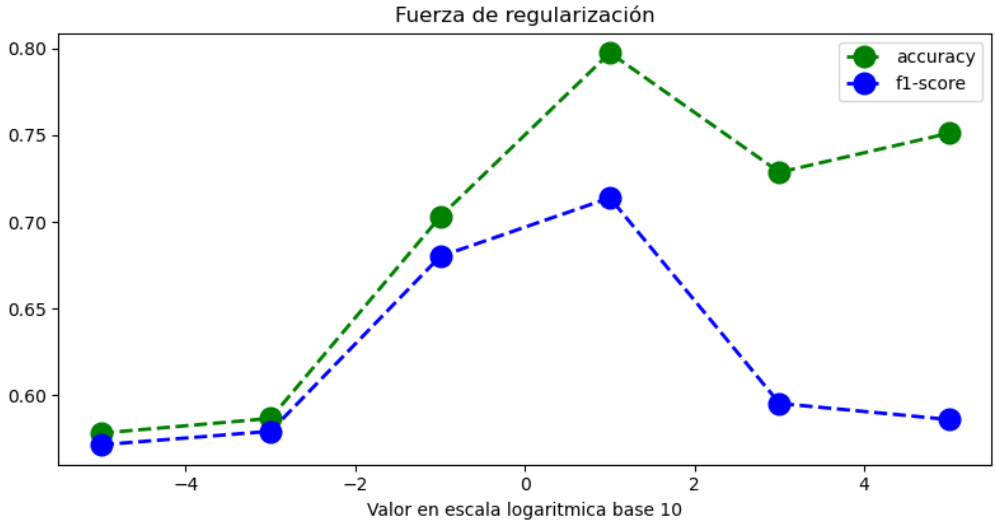

Figura 6: Cambiar este gráfico por le otro cuando esté.

7.4. Perceptrón multi capa

Según lo sugerido en la práctica consideramos el perceptron multi capa con dos capas intermedias. Cada una de ellas podrá tener valores de entre 50 y 100 unidades. Haremos una búsqueda a través de estas opciones para comprobar cuál es la mejor. En las tablas de resultados vemos los experimentos realizados.

	1ª capa 50	1ª capa 60	1ª capa 80	1ª capa 100
2ª capa 50	0,9106	0,9148	0,9256	0,9362
2ª capa 60	0,9115	0,9179	0,9329	0,9403
2ª capa 80	0,9209	0,9283	0,9359	0,9468
2ª capa 100	0,9277	0,9283	0,9409	0,9455

Tabla 10: Resultados en training de perceptrón multi capa.

	1ª capa 50	1ª capa 60	1ª capa 80	1ª capa 100
2ª capa 50	0,8264	0,8219	0,8211	0,8209
2ª capa 60	0,8281	0,8261	0,8198	0,8204
2ª capa 80	0,8242	0,8227	0,8217	0,8217
2ª capa 100	0,8254	0,8213	0,8242	0,8222

Tabla 11: Resultados en cross validation de perceptrón multi capa.

Podemos comprobar que la mejor opción es una de las más sencillas, 50 en la primera capa y 60 en la segunda. Además esta opción no es la que mejor ajusta el conjunto de training. Esto nos lleva a pensar que puede mejorarse el resultado simplificando el modelo. Probamos un perceptrón de dos capas y obtenemos, con 60 unidades en la capa intermedia una precisión en cross validation de 0,8403, superior a la obtenida con dos capas.
Así, los resultados obtenidos con ambos modelos pueden consultarse en las tablas correspondientes. Los modelos finales tomados son:

```
MLPClassifier(hidden_layer_sizes=[50, 60])
```

	Accuracy	F1-score
Train	0.9167	0.8839
Test	0.8230	0.7498
Validación	0.8297	0.7603

Tabla 12: Resultados ajuste MLP de tres capas.

	Accuracy	F1-score
Train	0.8867	0.8394
Test	0.8265	0.7541
Validación	0.8402	0.7772

Tabla 13: Resultados ajuste MLP de dos capas.

```
MLPClassifier(hidden_layer_sizes= 60)
```

8. Error de Generalización

Realizaremos dos trabajos para ganar comprensión del error real E_{out}, tanto según la métrica `accuracy` como la métrica `f1`. Se debe considerar que durante todo el texto se ha comentado en función de medidores de la bondad. Para medir el error debemos considerar `1 - accuracy` y `1 - f1`.

- Realizaremos una prueba de error en test y de ahí sacaremos una cota de E_{out}. Para ello aplicamos la fórmula expuesta en el desarrollo teórico de la asignatura que relaciona los valores en una muestra con la cota de un error 'real' E_{out}. En particular usaremos la formula:

$$E_{out} \leq E_{test} + \sqrt{\frac{1}{2N} \log \frac{2}{\delta}}$$

  Donde $1 - \delta$ es el grado de confianza que buscamos. Al utilizar los datos de test, podemos considerar una cota mucho más ajustada, al no tener que tener en cuenta la complejidad de de clase y la dimensión VC de esta. Consideramos una confianza del 99 %. Realizaremos la estimación del error real según el mejor resultado entrenado de cada modelo, es decir, es que nos ofrece mejor error de validación en el conjunto de Training. Al tener el conjunto de test 16281 muestras podremos conseguir una cota ajustada.

- Realizamos una estimación basa en los resultados de cross validation del conjunto de training. Aprovecharemos para este propósito que el error Cross validation es un estimador insesgado de E_{out}.

Procedamos a ver los resultados

Modelo	Accuracy			F1-score		
	E_{cv}	E_{test}	Cota E_{out}	E_{cv}	E_{test}	Cota E_{out}
Regresión logística	0.1494	0.1502	0.1514	0.2185	0.2297	0.2309
Random Forest	0.1629	0.1693	0.1705	0.2394	0.2704	0.2716
Support Vector Machine	-	-		-	-	
Perceptron Multi Capa	0.1598	0 .1735	0.1747	0.2228	0.2459	0.2471

Tabla 14: Resultados de error.

9. Conclusiones

Valoración de los resultados y justificación (gráficas, métricas de error, análisis de residuos, etc)
que se ha obtenido la mejor de las posibles soluciones con la técnica elegida y la muestra dada. Argumentar en términos de los errores de ajuste y generalización.

Referencias

[1] Página oficial de la Oficina del censo: http://www.census.gov/ftp/pub/DES/www/welcome.html.

[2] Página del Centro para el Apredizaje Automático y Sistemas Inteligentes:http://archive.ics.uci.edu/ml/datasets/Adult.

[3] Función StandardScaler de la librería sklearn: https://scikit-learn.org/stable/modules/generated/sklearn.preprocessing.StandardScaler.html

[4] Función cross_validate de la librería sklearn: https://scikit-learn.org/stable/modules/generated/sklearn.model_selection.cross_validate.html

[5] Artículo donde explica el concepto de dummy variable: https://en.wikipedia.org/wiki/Dummy_variable_(statistics)

[6] Métricas de sklearn https://scikit-learn.org/stable/modules/model_evaluation.html#balanced-accuracy-score

[7] Métrica f1-score https://en.wikipedia.org/wiki/F1_score

[8] Código RL: https://en.wikipedia.org/wiki/Coordinate_descent

[9] Uso del algoritmo varianceThreshold: https://scikit-learn.org/stable/modules/generated/sklearn.feature_selection.VarianceThreshold.html

[10] Técnica de PCA: https://scikit-learn.org/stable/modules/generated/sklearn.decomposition.PCA.html#sklearn.decomposition.PCA

[11] Artículo donde explica el concepto de maldición de la dimensión: https://en.wikipedia.org/wiki/Curse_of_dimensionality

[12] Función de regresión logística: https://scikit-learn.org/stable/modules/generated/sklearn.linear_model.LogisticRegression.html

[13] Función de ramdon forest: https://scikit-learn.org/stable/modules/generated/sklearn.ensemble.RandomForestClassifier.html

[14] Función de support vector machine: https://scikit-learn.org/stable/modules/generated/sklearn.svm.SVC.html

[15] Función de perceptrón multi capa para clasificación: https://scikit-learn.org/stable/modules/generated/sklearn.neural_network.MLPClassifier.html