Pareja: 02
Alba Ramos Pedroviejo
Nicolás Serrano Salas

Práctica 4
Grupo: 2361
INTART

Notebook 1

- **Número de vecinos en k-nn. ¿Por qué siempre debe ser impar cuando hay dos clases?**

  Se han recogido muestras para los 4 inputs tomando diferentes números de vecinos cada vez, tanto modificando los pesos como sin modificarlos:

Nº vecinos	Score Input 1	Score Input 2	Score Input 3	Score Input 4
1	0.93	0.97	1.00	0.97
2	0.92	0.98	1.00	0.96
9	0.96	0.98	1.00	0.98
9 (añadiendo peso)	0.96	0.98	1.00	0.98
10	0.96	0.98	1.00	0.98
11	0.96	0.96	1.00	0.98
11 (añadiendo peso)	0.96	0.98	1.00	0.98
50	0.94	0.82	0.95	0.94
50 (añadiendo peso)	0.94	0.94	0.98	0.98
101	0.95	0.69	0.87	0.59
101 (añadiendo peso)	0.95	0.92	0.95	0.96
201	0.95	0.60	0.81	0.53
201 (añadiendo peso)	0.95	0.91	0.91	0.87

Analizamos primero usando pesos por defecto:

Podemos comprobar cómo a medida que crece el número de vecinos empeoran los resultados para todos los inputs. Esto tiene sentido, ya que este método basa su predicción en los resultados de una muestra de puntos alrededor del que queremos predecir. De este modo, si esa muestra aumenta considerablemente, va a llegar un punto en el que no será representativa del problema, sino que cogerá valores de la clase contraria, que estarán alejados de nuestro punto pero se cogerán debido a que hay que cubrir el número de vecinos solicitado.

En relación con esto, podemos ver cómo para un número de vecinos más bajo mejoran los resultados para la mayoría de inputs (excepto el primero), ya que se toman vecinos cercanos y pocos, de forma que realmente se estime en función de lo que haya alrededor. Sin embargo, al ser escaso este número de vecinos, la predicción podría verse afectada por valores erróneos o "ruido" que haya en la muestra. De este modo, es conveniente aumentar el número de vecinos a un tamaño razonable para hacer una predicción realista, por ejemplo 9. De esta forma, la predicción se está basando en valores cercanos y suficientes para no verse afectada por ruido.

Analizamos ahora añadiendo más peso a los vecinos más cercanos:

Podemos añadir el parámetro weights='distance' para dar más peso a vecinos más cercanos y menos a los vecinos más lejanos dentro de nuestro k. De este modo, observamos una clara mejora a medida que aumenta k en la mayoría de los distintos inputs.

Esto sucede debido a que, cuando se están escogiendo muchos vecinos (k grande), independientemente de que sean más propicios a predecir bien (los más cercanos), se les va a dar el mismo peso a todos, pero si les damos más peso a los más cercanos, la predicción va a ser más precisa. Esto destaca claramente para el caso de K = 101, donde vemos una gran mejora respecto a predecir con pesos uniformes para el input 2 y el input 4.

Sin embargo, también podemos observar que para k pequeño (por ejemplo, k = 9) no se aprecia diferencia al modificar los pesos. Esto también es lógico, puesto que cuando estamos escogiendo un número de vecinos pequeño, estos serán los más cercanos (por la propia definición del modelo), por tanto el hecho de editar los pesos de los mismos no modificará el resultado (ya que los que se eligen ya son los más cercanos).

Finalmente, para k muy grande notamos que esta mejora disminuye (caso k = 200), puesto que si bien se le da más peso a los vecinos más cercanos, el resto seguirá teniendo su aporte (aunque sea menor). De este modo, por el

hecho de usar tantos vecinos, los lejanos tendrán aún así influencia en el resultado.

Cabe destacar, en cualquier caso, que es mejor utilizar un número de vecinos impar cuando tenemos 2 clases, ya que si usamos un número par podría ocurrir que hubiera un empate en la predicción y se tuviera que elegir al azar. Habiendo un número impar siempre vamos a poder desempatar.

- **Profundidad máxima de los árboles de decisión.**

  Se han recogido muestras para los 4 inputs tomando diferentes números de profundidad cada vez:

Profundidad máxima	Score Input 1	Score Input 2	Score Input 3	Score Input 4
1	0.88	0.59	0.79	0.61
2	0.92	0.59	0.88	0.73
3	0.93	0.76	0.88	0.82
4	0.91	0.79	0.89	0.93
5	0.92	0.91	0.95	0.93
10	0.91	0.93	0.95	0.94
25	0.91	0.93	0.95	0.94
50	0.91	0.93	0.95	0.94
100	0.91	0.93	0.95	0.94

  En la tabla anterior podemos observar las pruebas que hemos ejecutado para la resolución de esta pregunta. En ella podemos observar que, a partir de cierta profundidad, por mucho que aumente esta, las predicciones que se van a hacer no van a cambiar, e incluso en algunos casos como en el del Input 1 se puede realizar un sobreajuste y disminuir la precisión de las predicciones. Por lo tanto, remarcar que la profundidad a la que debemos hacer las predicciones es relativa a los tipos de datos que tenemos, pero que a simple vista y teniendo en cuenta todos los inputs que estamos estudiando la

profundidad máxima óptima sería la 5, pues todas las predicciones sobrepasan el 0.9 de score.

- **Número de neuronas en la red neuronal y máximo número de épocas de entrenamiento. Nota: (50,) indica una única capa oculta con 50 neuronas. (50,10,) indica dos capas ocultas con 50 y 10 neuronas respectivamente. (50,10,20,) indica tres capas ocultas con 50, 10 y 20 neuronas respectivamente, etc.**

  Se han recogido muestras para los 4 inputs tomando diferentes números de neuronas y capas cada vez:

Neuronas y capas	Score Input 1	Score Input 2	Score Input 3	Score Input 4
(2,)	0.94	0.99	0.84	0.69
(5,)	0.94	0.99	0.91	0.69
(25,)	0.95	0.99	0.97	0.99
(50,)	0.95	0.99	0.98	0.99
(2,2,)	0.95	0.99	0.86	0.67
(5,5,)	0.95	0.99	0.85	0.96
(50,50,)	0.95	0.99	1.00	0.98
(2,2,2,)	0.48	0.99	0.90	0.78
(5,5,5,)	0.94	0.99	1.00	0.99
(50,50,50,)	0.94	0.99	1.00	0.98

  Observamos que, por muy pocas neuronas que utilicemos, aunque no usemos capas ocultas, ya mejora con creces la precisión para el Input 2 a diferencia de los anteriores métodos probados (k-nn y árboles de decisión).

  Asimismo, observamos que no por añadir más neuronas por capa o más capas mejora la predicción, como podemos comprobar para el caso (50,50,50,) respecto al caso (25,). Incluso podría empeorar, como vemos para el caso del Input 1 con (2,) y (2,2,2,).

  Además, a medida que vamos añadiendo neuronas y capas, el coste de procesamiento va aumentando (se ha notado que empeora sobre todo al realizar la prueba de (50,50,50,)), y si los datos de entrada no tienen tantas

características relevantes, podría ocurrir que haya muchas de estas neuronas no se utilicen apenas.

En definitiva, parece que en este tipo de predicción, la precisión depende mucho del tipo de datos de la entrada, y no se puede generalizar.

Notebook 2

- **¿Cuál es el mejor score que consigues con un k-nn y con qué k (valor de n_neighbours)?**

  Tras realizar pruebas usando *KNeighborsClassifier* usando distinto número de vecinos cada vez (k), observamos los siguientes resultados:

Nº vecinos	Score	Score tras estandarizar
1	0.66 +/- 0.05	0.71 +/- 0.02
5	0.72 +/- 0.02	0.73 +/- 0.02
7	0.74 +/- 0.02	0.74 +/- 0.02
9	0.74 +/- 0.03	0.74 +/- 0.03
11	0.75 +/- 0.03	0.74 +/- 0.02
13	0.76 +/- 0.04	0.76 +/- 0.01
15	0.74 +/- 0.04	0.76 +/- 0.02
25	0.74 +/- 0.05	0.75 +/- 0.02
55	0.73 +/- 0.02	0.76 +/- 0.02

  En vista de estos resultados, el mejor score (sin estandarizar datos) se obtiene para K = 13 vecinos, pues se obtiene 0.76. Sin embargo, la desviación típica de esta predicción es de +/- 0.04, lo cual indica que no es tan precisa como otras predicciones que tenemos. Por ejemplo, para k = 7 vecinos tenemos un score de 0.74 con una desviación típica de +/- 0.02, y para k = 11 vecinos tenemos un score de 0.75 con una desviación típica de +/- 0.03. Si analizamos estos tres resultados podemos comprobar que, si a los tres les restasemos la desviación típica, todos darían 0.72. Por lo tanto, en el caso peor los tres resultados predecirán lo mismo. Sin embargo, en el caso mejor está claro que **k = 13** vecinos es el mejor de los tres. En vista de este análisis, nos quedaremos con este valor de k.

Si probamos a **estandarizar los datos** antes de realizar la estimación (cosa que suele ser útil para K-NN y para redes neuronales), lo que ocurriría sería que todos los atributos se pondrían "a la misma escala", dicho muy burdamente. En este sentido, podemos observar una mejora general para el score obtenido en este modelo si estandarizamos los datos al usar un número de vecinos más elevado (sobre todo se observa una menor desviación estándar, es decir, que es más fiable la precisión). Sin embargo, en relación score-desviación, volvemos a quedarnos con **k = 13** vecinos, si bien el resto de predicciones han mejorado.

- **¿Cuál es el mejor score que consigues con un árbol de decisión y con qué profundidad máxima (valor de max_depth)?**

  Tras realizar pruebas manualmente para *DecisionTreeClassifier* usando distinto número de profundidad cada vez, observamos los siguientes resultados:

Profundidad máxima	Score
1	0.72 +/- 0.03
2	0.74 +/- 0.01
3	0.73 +/- 0.01
4	0.73 +/- 0.03
5	0.75 +/- 0.04
6	0.73 +/- 0.04
7	0.74 +/- 0.03
8	0.71 +/- 0.02
9	0.71 +/- 0.04
10	0.71 +/- 0.04
25	0.71 +/- 0.03
50	0.71 +/- 0.03
100	0.71 +/- 0.05

Podemos observar en la tabla anterior que el mejor score de media lo obtenemos con profundidad 5, 0.75, pero también tiene una desviación típica de +/- 0.04, por lo que opinamos que la mejor profundidad para el análisis de

estos datos es la de 2, puesto que tiene la segunda media más alta 0.74, y una desviación típica de +/- 0.01, por lo que las predicciones a esta profundidad serán más fiables.

- **¿Cuál es el mejor score que consigues con una red neuronal y con qué configuración (valor de hidden_layer_sizes)?**

  Tras realizar pruebas usando *MLPClassifier* usando distinto número de profundidad cada vez, observamos los siguientes resultados:

Neuronas y capas	Score alfa = 0	Score alfa = 0 (estandarizado)	Score alfa = 5	Score alfa = 5 (estandarizado)
(2,)	0.65 +/- 0.00	0.75 +/- 0.05	0.67 +/- 0.03	0.74 +/- 0.05
(5,)	0.64 +/- 0.07	0.76 +/- 0.02	0.69 +/- 0.04	0.77 +/- 0.01
(10,)	0.70 +/- 0.04	0.77 +/- 0.04	0.67 +/- 0.05	0.77 +/- 0.01
(15,)	0.66 +/- 0.08	0.77 +/- 0.04	0.67 +/- 0.02	0.77 +/- 0.01
(25,)	0.67 +/- 0.03	0.77 +/- 0.03	0.71 +/- 0.04	0.77 +/- 0.02
(50,)	0.64 +/- 0.06	0.76 +/- 0.04	0.69 +/- 0.02	0.76 +/- 0.02
(2,2,)	0.65 +/- 0.00	0.75 +/- 0.03	0.62 +/- 0.10	0.69 +/- 0.04
(5,5,)	0.66 +/- 0.03	0.77 +/- 0.02	0.69 +/- 0.03	0.75 +/- 0.02
(10,10,)	0.69 +/- 0.02	0.77 +/- 0.03	0.68 +/- 0.03	
(15,15,)	0.68 +/- 0.03	0.75 +/- 0.03	0.70 +/- 0.02	0.75 +/- 0.01
(25,25,)	0.68 +/- 0.07	0.75 +/- 0.01	0.68 +/- 0.02	
(50,50,)	0.68 +/- 0.03	0.74 +/- 0.02	0.69 +/- 0.02	
(2,2,2,)	0.67 +/- 0.02	0.71 +/- 0.07	0.66 +/- 0.03	0.65 +/- 0.00
(5,5,5,)	0.67 +/- 0.04		0.71 +/- 0.04	
(10,10,10,)	0.67 +/- 0.06	0.76 +/- 0.03	0.72 +/- 0.02	
(15,15,15,)	0.65 +/- 0.02		0.71 +/- 0.02	
(25,25,25,)	0.69 +/- 0.03	0.70 +/- 0.03	0.73 +/- 0.02	
(50,50,50,)	0.65 +/- 0.04		0.72 +/- 0.03	
(2,2,2,2,2,)	0.65 +/- 0.00		0.67 +/- 0.04	

(5,5,5,5,5)	0.69 +/- 0.04		0.71 +/- 0.05	
(10,10,10,10,10,)	0.73 +/- 0.03	0.74 +/- 0.02	0.73 +/- 0.02	
(15,15,15,15,15,)	0.69 +/- 0.03		0.74 +/- 0.02	
(25,25,25,25,25,)	0.71 +/- 0.02		0.76 +/- 0.02	
(50,50,50,50,50,)	0.68 +/- 0.03		0.75 +/- 0.02	
(25,25,25,25,25,25,25,25,25,25)	0.71 +/- 0.04		0.65 +/- 0.00	

***Nota**: en la tabla algunas celdas no se han rellenado porque se ha concluido, según los resultados anteriores, que no va a haber una mejora significativa.*

Primero hemos realizado pruebas tomando **alfa = 0**, lo que significa que puede ocurrir que tengamos muchas entradas de datos y algunas de ellas no aporten información relevante al problema. También podría ocurrir en este caso que, si usamos muchas neuronas en una capa, existan muchas conexiones que no se están usando (y esto se propagaría al ir aumentando la cantidad de capas ocultas). Todo esto incurriría en una penalización en la predicción conocida como penalización L2, que de momento dejaremos pasar.

Para este caso, podemos observar que las predicciones usando la red neuronal no son muy buenas y, además, son bastante aleatorias. No se observa una tendencia de mejora al aumentar el número de neuronas por capa, ni tampoco al aumentar el total de capas ocultas. En este sentido, tras realizar las pruebas recogidas en la tabla podemos ver que el caso **(10,10,10,10,10,)** es el que mejor score presenta (relacionándolo con la desviación, como se ha explicado para apartados anteriores).

Sin embargo, como nos parecían muy aleatorios estos resultados, probamos a **estandarizar los datos** antes de calcular el score y probar con el mismo valor de alfa. En estas pruebas hemos podido comprobar que *no por añadir más capas mejora la predicción*, sino que con una y dos capas ya obtenemos buenos resultados. Observamos también que el resultado con este número de capas es bastante similar cuando vamos añadiendo neuronas, así que nos quedaremos con el mejor resultado (relación score-desviación) proporcionado por el menor número de neuronas y de capas, es decir, nos quedamos con **(5,)**. Podemos concluir que *es necesario estandarizar los datos* a la hora de realizar las estimaciones, y de esta forma vamos a tener un ahorro computacional notable (ya que obtenemos un mejor resultado que en el caso sin estandarizar, y con menos neuronas y capas ocultas).

Para corregir la **penalización L2** podemos **aumentar alfa**. A mayor alfa, mayor número de variables irrelevantes va a intentar eliminar el modelo. Sin embargo, si este valor aumenta desmedidamente, entonces lo que ocurrirá será que se tomarán tan pocas variables como relevantes que la predicción será incorrecta, de forma que se infra-ajuste y sea más aleatoria. Tras realizar pruebas para distintos valores de alfa, hemos observado que **alfa = 5** mejora bastante las predicciones de alfa = 0, como se recoge en la tabla para casi todos los casos. Además, ahora sí parece observarse una tendencia al crecimiento en el score al aumentar tanto capas como neuronas por capa, obteniendo el máximo en el caso **(25,25,25,25,25,)**. Sin embargo, esta conclusión la obtenemos sin haber estandarizado previamente los datos.

Tras **estandarizar los datos** y usar alfa = 5, hemos vuelto a obtener unos resultados mejores para un número bajo de neuronas y capas. Volvemos a obtener el mismo resultado que obtenemos para alfa = 0, **(5,)**.

En definitiva, la conclusión que obtenemos es que conviene utilizar un modelo con un *valor aumentado de alfa* para estos datos, ya que parece que está habiendo variables que no están aportando mucha información sobre el problema y por eso al usar alfa = 0 las predicciones son bastante peores. Pero lo más importante es *estandarizar los datos* antes de realizar las predicciones, ya que este factor sí que influye notablemente en los resultados, y nos permite asimismo comprobar que no por aumentar el *número de neuronas o de capas* mejora el resultado, sino que para dos capas como mucho ya tendremos nuestros mejores resultados.

Notebook 3

Análisis de los datos

Lo primero que hemos hecho ha sido observar qué datos tienen las **columnas** del dataset para comprobar cuáles toman valores binarios y cuáles no (ya que estandarizaremos únicamente las variables que no sean binarias).

A continuación, hemos dibujado los **histogramas** de cada columna (excepto la clase) para ver cuáles tenían largas colas (vemos que, en general, ocurre con todas las variables), lo que indica que hay grandes diferencias entre sus valores y será necesario escalarlo. En este punto hemos descubierto que las dos primeras columnas indican la misma información pero en diferente escala, por tanto podemos eliminar una de ellas debido a que la información es redundante.

Finalmente, comprobamos si hay **valores negativos** en alguna fila, ya que estos indican algún error en la fila (ya que, con las columnas que tenemos, no tiene sentido que algún valor sea negativo). Como encontramos 2 filas en este estado, estas no las tendremos en cuenta.

Preparación y estandarización de datos

Lo primero será **ajustar** las **variables no binarias** cuyos histogramas mostraban largas colas (todas, básicamente). Esto será importante para modelos como regresión logística o KNN, que se ven afectados por las grandes diferencias que, gracias al ajuste, logramos eliminar. Esto se visualiza al pintar de nuevo las gráficas y observar que ya las colas extensas han desaparecido.

A continuación, **eliminamos columnas irrelevantes** para la predicción: la clase, el id del usuario (es un campo no numérico que no aporta información relevante) y “network_age” (que muestra la misma información que “customer tenure in months” pero en otra escala).

Finalmente, **estandarizamos los datos**, pero únicamente las columnas no binarias, que son las 9 primeras.

Prueba de modelos individuales

Hemos probado a entrenar distintos algoritmos para comprobar con cuál obteníamos un mejor score y una menor desviación estándar. Para ello, hemos utilizado **GridSearch** y **cross-validation** de tamaño 10 para variar los parámetros de los modelos y comprobar así cuáles son más adecuados para los datos que estamos usando. A continuación se muestran los resultados obtenidos para los modelos probados:

Gaussiana

Parámetros: por defecto
Score: 0.70 +/- 0.16

Redes neuronales

Variando solamente número de neuronas en una capa entre 1 y 50:
Parámetros: {'hidden_layer_sizes': (8,), 'max_iter': 1000}
Score: 0.71 +/- 0.10

Variando número de capas y neuronas pero también learning rate:
Parámetros: {'alpha': 5, 'hidden_layer_sizes': (5, 10), 'learning_rate_init': 0.01, 'max_iter': 10000}
Score: 0.73 +/- 0.10

Variando número de neuronas por capa, learning rate y alfa:
Parámetros: {'alpha': 1, 'hidden_layer_sizes': (14,), 'learning_rate': 'invscaling', 'learning_rate_init': 1, 'max_iter': 10000}
Score: 0.72 +/- 0.10

Regresión logística

Parámetros: {'C': 0.01, 'max_iter': 1000, 'penalty': 'l2'}
Score: 0.71 +/- 0.11

Knn

Parámetros: {'n_neighbors': 29, 'weights': 'uniform'}
Score: 0.72 +/- 0.11

Árboles de decisión

Parámetros: {'max_depth': 5, 'max_leaf_nodes': 17}
Score: 0.76 +/- 0.10

SVC

Parámetros: {'C': 10, 'gamma': 0.01, 'kernel': 'sigmoid'}
Score: 0.70 +/- 0.13

Combinación de modelos

En este punto, tras observar que los scores obtenidos no eran tan altos como nos esperábamos (y, además, la varianza era bastante elevada en todos ellos), hemos decidido probar combinaciones de modelos siguiendo las indicaciones proporcionadas por los profesores de la asignatura, tanto de teoría como de prácticas. Hemos hecho hincapié en árboles de decisión, puesto que hemos visto que es el que mejor relación score-varianza obtiene. En este sentido, hemos probado varios tipos de clasificadores, cuyo resultado se muestra a continuación. En definitiva, hemos comprobado que combinar un modelo basado en árboles de decisión es lo que mejores resultados parece ofrecer.

Voting

Parámetros:

- BaggingClassifier(base_estimator = DecisionTreeClassifier(max_depth=5, max_leaf_nodes=17),
  n_estimators = 10, random_state=0)
- clf2 = RandomForestClassifier(max_depth=9, n_estimators=500)
- clf3 = DecisionTreeClassifier(max_depth=5, max_leaf_nodes=17)
- clf4 = MLPClassifier(alpha=5, hidden_layer_sizes=(5,10,), learning_rate_init=0.01, max_iter=10000)

Score: 0.76 +/- 0.10

Random forest

Parámetros: {'max_depth': 9, 'n_estimators': 500}
Score: 0.76 +/- 0.10

Aunque parece que este caso no mejora al árbol de decisión individual, es mejor debido a que se trata de la puesta en común de las decisiones de un conjunto de árboles, lo que va a ser más fiable que las decisiones de uno solo.

Bagging

Parámetros:

- Mejor árbol de decisión obtenido en pruebas individuales ({'max_depth': 5, 'max_leaf_nodes': 17})
- 10 estimadores

Score: 0.78 +/- 0.08

Boosting: AdaBoost

Parámetros:

- Mejor árbol de decisión obtenido en pruebas individuales ({'max_depth': 5, 'max_leaf_nodes': 17})
- Mejor random forest ({'max_depth': 9, 'n_estimators': 500})
- 10 estimadores

Score: 0.77 +/- 0.10

Predicciones sobre el dataset de construcción

Primero vamos a verificar la robustez de nuestro modelo usando los datos de entrenamiento. Para ello, tomamos Bagging y árboles de decisión con los mejores parámetros obtenidos en las pruebas anteriores y realizamos las predicciones. Dichas predicciones se encuentran ordenadas siguiendo el mismo orden de filas de la tabla con los datos de construcción, por lo que para hacer la comparativa crearemos una nueva columna con la probabilidad de que se marche el cliente. A continuación comprobaremos cuántos **falsos positivos** (predecimos que se marchan y en realidad se quedan) y **falsos negativos** (predecimos que se quedan y en realidad se marchan) tenemos en la predicción, obteniendo un valor bastante bajo. Además, al comprobar los 100 más probables de la tabla vemos que las probabilidades que muestra son bastante altas (tiene sentido, por ser el conjunto de datos con los que hemos entrenado) pero no incurrimos en sobre-ajuste, ya que estas probabilidades no exceden el 97%. Finalmente, calculando los aciertos, obtenemos un **87%**, por tanto el modelo parece bastante robusto.

Predicciones sobre el dataset de explotación

Como hemos entrenado nuestro modelo teniendo en cuenta la eliminación de filas con valores negativos y la estandarización de datos, tendremos que tener esto en cuenta también para los datos de explotación, por tanto todo ese procedimiento de **preparación y estandarización de datos** inicial que realizamos para los datos de construcción lo haremos también para los de explotación.
A continuación, hemos entrenado nuestros 3 mejores modelos con el objetivo de **reducir la incertidumbre**, realizando la **intersección** de los IDs que tengan en común los tres predictores (el proceso de obtención de probabilidades es análogo al explicado en el apartado anterior, pero usando los datos del conjunto de explotación). Al realizar la intersección, hemos visto que hay 65 IDs en común entre los resultados ofrecidos por los tres modelos, por lo que parece bastante fiable. Finalmente, para completar los IDs restantes tomaremos los resultados ofrecidos por el mejor de los 3 modelos: Bagging. Los resultados de la predicción se encuentran adjuntos en el fichero *predicciones.csv*.