UNIVERSIDAD DE GRANADA

AA

Aprendizaje Automático

Práctica 3: Clasificación

Autores:
Mario Carmona Segovia

Profesor: Pablo Mesejo Santiago

Escuela Técnica Superior de Ingenierías Informática y de Telecomunicación

—

Curso 2020 - 2021

Índice

Índice de figuras

Índice de tablas

1. Analizar y comprender el problema a resolver

El problema de clasificación que vamos a utilizar, esta relacionado con el diagnostico de sistemas de accionamiento electromecánico. El problema consiste en crear un modelo que prediga en que estado se encuentra el sistema en base al comportamiento del sistema, es decir, sin utilizar sensores extras para el diagnóstico. Los datos usados en este problema han sido obtenidos del proyecto de investigación AutASS. Para obtener estos datos se utilizó un sistema de prueba que puede ser utilizado bajo diferentes condiciones de acción, como puede ser la sobrecarga, un incorrecto montaje; lo que deriva ciertos daños y errores en la piezas.

De las pruebas con ese sistema de prueba se extrajeron 11 clases diferentes. Una clase que indica que no hay fallos en el sistema, y otras 10 clases que indican que hay algún tipo de error en el sistema.

1.1. Elementos del problema

1.1.1. Datos de entrada (X)

Los datos de entrada se componen de 58509 ejemplos de condiciones del sistema, teniendo 48 características distintas cada ejemplo. Esta características, como se indica en el paper [1], están relacionadas con las distintas cargas en los cojinetes, con las cargas de par, con las velocidades, etc. Dentro del fichero de datos, los datos de entrada se encuentran en las primeras 48 columnas. Todos los datos son de tipo numérico.

1.1.2. Salida (Y)

La salida del problema es la etiqueta de la clase a la que pertenece el ejemplo. La clase de cada ejemplo de los datos, se especifica en la columna número 49 de los datos. Las clases van desde el 1 hasta el 11.

1.1.3. Función del problema (f: $\mathbf{X} \rightarrow \mathbf{Y}$)

La función del problema consistirá en a partir del valor de la distintas características de las condiciones de un sistema, obtener la clase de estado en que se encuentra el sistema.

1.2. Relación entre las características

Para comprobar si los datos del problema están relacionados, ya sea positivamente o negativamente, voy a calcular la matriz de correlaciones de los atributos de todos los datos de entrada para entrenamiento.

Una correlación es un valor entre -1 y 1, que indica la relación positiva o negativa entre las características. Cuanto más cerca del 0, menos relación tienen las características.

Para calcular la matriz de coeficientes de correlación se hace uso de la función corrcoef de numpy [2]. Esta función calcula la correlación de Pearson de cada par de características, y devuelve los valores como una matriz. Los argumentos más importantes de esta función son:

- x: Matriz de datos
- rowvar: Si es igual a True indica que los atributos son las distintas filas de la matriz de datos, si es igual a False son las distintas columnas de la matriz de datos

De la matriz de correlaciones calculada se puede deducir la siguiente información:

Mínimo	0.000028
Media	0.148

Tabla 1: Datos obtenidos de la matriz de coeficientes de correlación

- Como la mínima correlación no es cero, todas las variables están relacionadas.
- Como la media de correlaciones tiene un valor bajo para la gran cantidad de características que tiene el problema, parece poco probable que sea posible una reducción de la dimensionalidad.

Para tener una mejor idea de la correlación que hay entre las variables, he generado la siguiente gráfica donde es más visible la alta correlación de los valores:

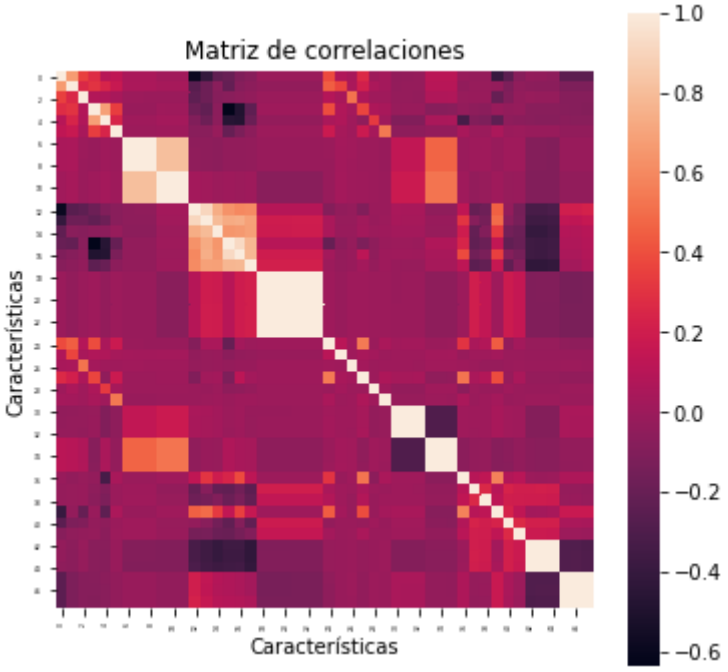

Figura 1: Matriz de correlaciones

En la figura se puede comprobar como hay poca variabilidad en las zonas de color, y hay grandes zonas con poca correlación.

Cuando dos variables tienen mucha correlación quiere decir que se puede eliminar una de ellas, ya que son casi equivalentes.

Dado que se ha obtenido un valor medio de correlación no muy alto pero que no llega a ser nulo, lo más normal sería probar a reducir la dimensionalidad con alguna técnica como PCA y ver si vale la pena.

2. Elección de la clase de funciones

La clase de funciones (H) que se va a usar, es la clase de los polinomios de orden 1; porque es una clase simple de la que se podrá ajustar un buen modelo dada la cantidad de datos de los que disponemos.

La función a ajustar será una combinación lineal de los todas las características con sus respectivos pesos.

$$\hat{y}(w, x) = w_0 + \sum_{n=0}^{N} w_n x_n$$

Siendo N el número total de características y los w's los coeficientes del modelo.

He decidido no realizar transformaciones no lineales porque no he encontrado en la literatura ningún sugerencia que haya dado buenos resultado, ni tengo el suficiente conocimiento sobre el campo del problema como para encontrar alguna relación entre las características.

3. Identificar el conjunto de hipótesis

Dado que el problema a resolver es un problema de clasificación, los algoritmos a usar para ajustar la función son los siguientes:

- Perceptron (PLA)
- Stochastic Gradient Descent (SGD)

Ambos clasificadores son clasificadores binarios, por lo que hace falta usar una técnica como One VS All, para convertir una clasificación multiclase en varias clasificaciones binarias. Cada una de estas clasificaciones binarias va a ser entre una de las clases y el resto de clases juntas. Finalmente se hará la media de las clasificaciones obtenidas con cada clase.

Para realizar esta técnica he usado la función OneVsRestClassifier de la librería scikit learn [3].

El único parámetro interesante es el parámetro estimator, el cuál contendrá el modelo a ajustar, Perceptron ó SGD.

3.1. Perceptron (PLA)

Este clasificador no necesita de ningún input para resolver el problema, digamos que los propios datos conducen a una solución correcta. La solución no es óptima. En este algoritmo se va iterando de uno en uno sobre todo el conjunto de datos. Una iteración no sabe nada de la anterior, se actúa sobre el punto de la iteración sin importar que se deshagan cambios hechos en anteriores iteraciones. El error en este tipo de algoritmo no desciende en forma de monótona conforme pasa el tiempo debido a que las iteraciones no tienen en cuenta los cambios en iteraciones anteriores. Si lo datos son linealmente separables puede llegar a obtener error igual a cero, pero si no se sabe si son linealmente separables se debe indicar un máximo de iteraciones para que termine el algoritmo por si no es linealmente separable.

$$h(x) = sign(w^T x)$$

El clasificador Perceptron es equivalente al algoritmo SGD con función de pérdida del perceptrón, tasa de aprendizaje igual a 1 y constante; y sin penalización.

Los parámetros más importantes de la función del clasificador Perceptron en scikit learn [4] son los siguientes:

- penalty: Tipo de regularización
- alpha: representa la intensidad de la regularización
- max_iter: número máximo de iteraciones

- eta0: tasa de aprendizaje inicial

Esta función de scikit learn usa siempre como función de error el perceptron. La fórmula de la función de error es la siguiente:

$$L(y_i, f(x_i)) = max(0, -y_i f(x_i))$$

3.2. Regresión Logística

El algoritmo de regresión logística es equivalente a utilizar el algoritmo SGD con la función de pérdida sigmoidal (log). La fórmula de la función de error es la siguiente:

$$L(y_i, f(x_i)) = log(1 + exp(-y_i f(x_i)))$$

El algoritmo SGD es una técnica de optimización que lleva mucho tiempo usándose en el machine learning. Como se indica en la gría de scikit learn [5], SGD no es ningún modelo de aprendizaje automático, sólo es una técnica de optimización. Este algoritmo da buenos resultados con conjuntos de datos grandes, como puede ser nuestro problema que tiene más de 10.000 ejemplos.

SGD es una variante del algoritmo de gradiente descendente. Esta variante intenta solucionar el problema de caer en mínimo o máximo locales, intentando separar en varios partes los datos que guían la búsqueda, para que el avance del algoritmo no se vea afectado por unos pocos datos que son malos para el ajuste.

$$h(x) = \sigma(w^T x)$$

Los parámetros más importantes de la función SGDClassifier en scikit learn [6] son los siguientes:

- loss: Función de pérdida a usar
- penalty: Tipo de regularización
- alpha: representa la intensidad de la regularización
- max_iter: número máximo de iteraciones
- learning_rate: forma de calcula la tasa de aprendizaje en cada iteración
- eta0: tasa de aprendizaje inicial

Una desventaja de SGD es que es sensible a la escala de las características, por lo que es imprescindible realizar una normalización para obtener buenos modelos con esta técnica.

4. Separación de los datos en training y test

Dado que disponemos de un conjunto de datos, que no están separados previamente en training y test, debemos realizar un proceso de validación, separando por nuestra cuenta los datos. Para este problema se ha decidido separar un 20 % de los datos para test y el resto para training. Como no hay criterio fijo en la elección del tamaño de ambos conjuntos de datos, me he decidido por la regla general que se sigue, que es dividir entorno al 20 % de los datos para test.

Una vez decidida la proporción de datos para cada parte, toca elegir un método para dividirlos de forma que los datos sigan siendo independientes e idénticamente distribuidos.

Para mantener la proporción que tienen las clases en el conjunto total de datos, en vez de dividir tal cual los datos en dos partes, para mantener esa proporción he decidido dividir los datos de cada clase en un 80 % para training y 20 % para test. Una vez dividida todas las clases, se unen todos los datos de training de cada clase y los datos de test de cada clase.

Una vez realizada la separación en training y test, los datos de test no se deben tocar durante todo el proceso de elección de la hipótesis final, ya que sino se realizaría data snooping, es decir, se está contaminando el ajuste del modelo con los datos de test, por lo que el modelo resultante dará buenos resultados con el test, pero no será un modelo correcto para fuera de la muestra, ya que al sufrir data snooping se ha perdido generalidad.

	Tamaño	**Porcentaje**
X	58509	100 %
X_train	46805	80 %
X_test	11704	20 %

Tabla 2: Separación de los datos de entrada

	Tamaño	**Porcentaje**
Y	58509	100 %
Y_train	46805	80 %
Y_test	11704	20 %

Tabla 3: Separación de los datos de salida

Clase	**Total**	**Train**	**Test**
1	5319	4255	1064
2	5319	4255	1064
3	5319	4255	1064
4	5319	4255	1064
5	5319	4255	1064
6	5319	4255	1064
7	5319	4255	1064
8	5319	4255	1064
9	5319	4255	1064
10	5319	4255	1064
11	5319	4255	1064

Tabla 4: Separación de los datos de cada clase por separado

5. Preprocesado de los datos

Para que el entrenamiento del modelo sea lo más eficaz y eficiente posible, se deben preprocesar los datos de training.

El preprocesamiento de los datos usado consiste en los siguientes pasos:

- Eliminación de los datos sin variabilidad
- Reducción/Aumento de la dimensionalidad
- Normalización de los datos
- Codificación de los datos
- Datos faltantes
- Datos extremos

5.1. Eliminación de los datos sin variabilidad

Se decide eliminar las características sin variabilidad porque una característica sin variabilidad no aporta nada al aprendizaje, por lo que se eliminan esas características para no malgastar recursos. Una característica sin variabilidad es aquella en la que no varía su valor en los distintos ejemplos de los datos de entrada.

En los datos de este problema no hay datos sin variabilidad.

5.2. Reducción/Aumento de la dimensionalidad

Para comprobar la distribución de varianza explicada entre las características he decidido usar PCA. PCA es una técnica para la reducción de dimensionalidad de los datos mediante la descomposición de valores singulares.

Como PCA es una técnica muy sensible a la diferencia de escalas en las características, se deben normalizar los datos antes de usar PCA.

Una vez realizada la técnica PCA sobre los datos, se obtienen las varianzas que explica cada característica.

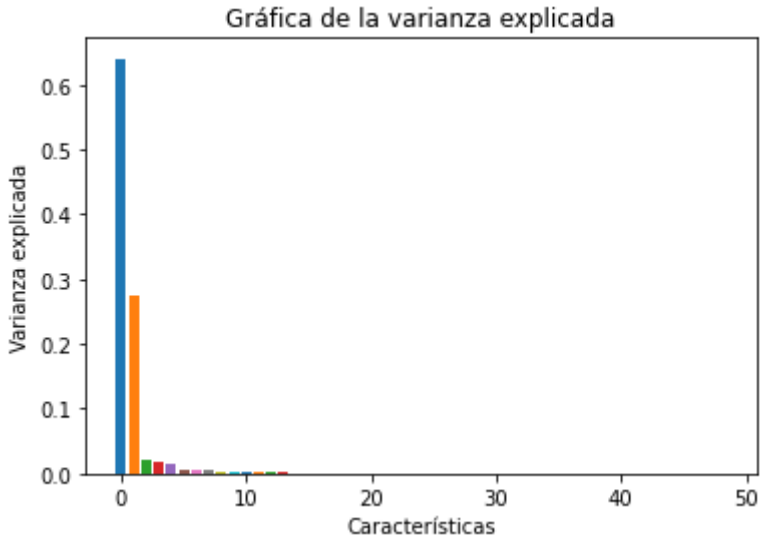

Figura 2: Varianza de cada característica

Como se puede ver en esta figura, la mayoría de las varianza está concentrada en unas pocas variables.

Para comprobar con cuantas variables se podría explicar al menos un 99 % de la varianza he calculado la varianza acumulada de cada característica, y he obtenido la siguiente gráfica:

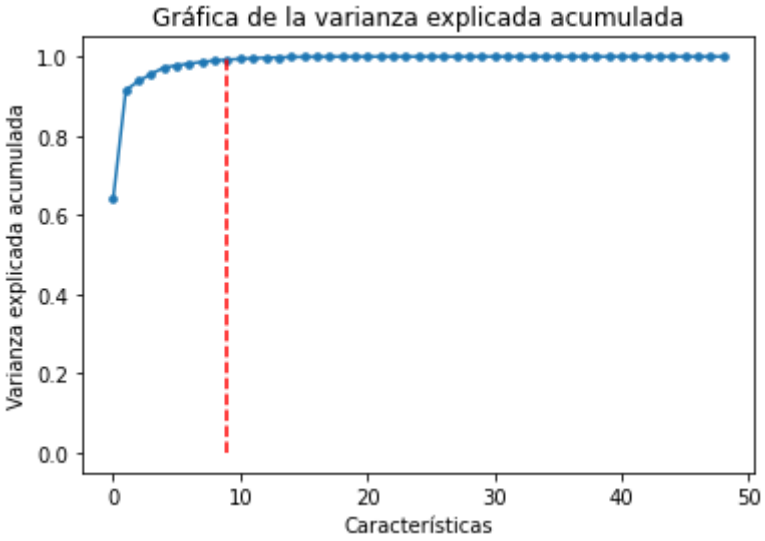

Figura 3: Varianza acumulada de cada característica

En esta figura se indica con una línea roja la primera característica que tiene al menos un 99 % de la varianza explicada. Según la gráfica, con 9 características se puede explicar al menos el 99 %. Por lo que en principio con 9 características se obtendría casi los mismos resultados que

con todas las características.

He decidido no realizar la reducción de dimensiones, debido a que no he obtenido ninguna mejora en la pruebas.

5.3. Normalización de los datos

He decidido normalizar los datos de las distintas características porque ciertas características tienen escalas y rangos muy distintos. Esto se puede comprobar con la siguiente gráfica donde se muestra el mínimo, máximo y media de cada característica.

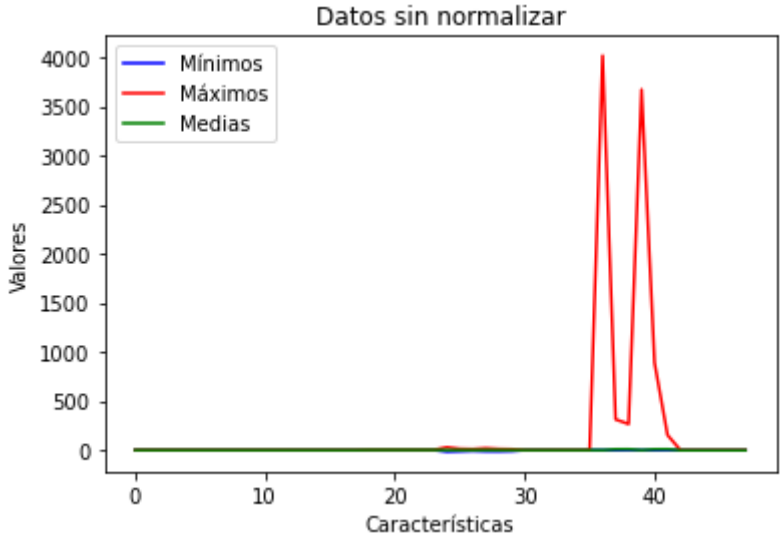

Figura 4: Datos sin normalizar

Para evitar que esa diferencia de escalas influya en el aprendizaje, como le pasa a SGD o a la regularización; más que la propia influencia de la característica en el aprendizaje, se deben normalizar los datos. Se han normalizado los datos en un rango de valores [0,1], para ello se ha utilizado la función MinMaxScaler de la librería scikit learn [7].

Esta función consiste en escalar y trasladar cada ejemplo individualmente al rango que se le ha pasado como parámetro, en mi caso el rango es [0,1].

$$X\_desviEstantar = (X - X\_min)/(X\_max - X\_min)$$
$$X\_escala = X\_desviEstandar * (max\_rango - min\_rango) + min\_rango$$

El principal parámetro de la función es feature_range, que fija el rango de valores que acota el valor de los datos.

Una vez realizada la normalización de los datos de entrenamiento, en la siguiente gráfica se puede ver el resultado de esta.

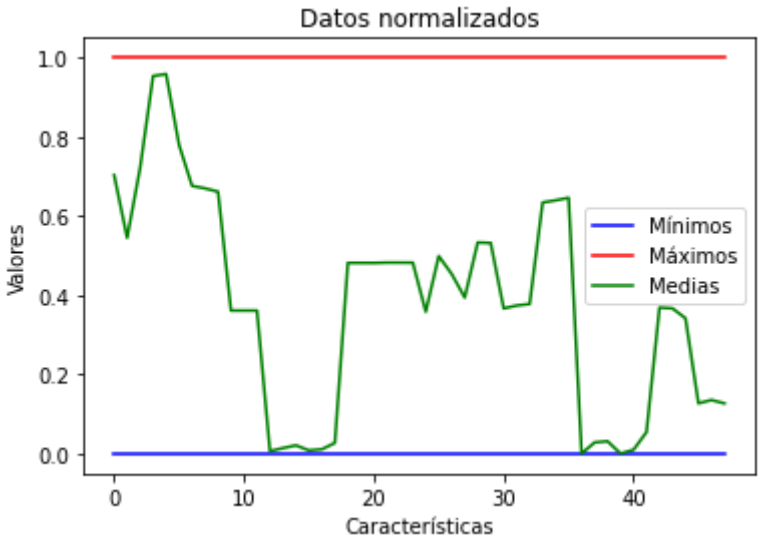

Figura 5: Datos normalizados

Como se puede apreciar en la figura [5], ahora los datos están acotados en el rango [0,1]. Ahora los rangos de las características no afectarán tanto y mejorará el aprendizaje.

5.4. Codificación de los datos

No hace falta codificar los datos, ya que si comprobamos los datos de entrada se puede ver que todas las características tiene un valor numérico, por lo que se dejan tal cuál están los datos.

5.5. Datos faltantes

En este apartado tampoco hace falta procesar los datos, ya que como se puede comprobar en la descripción de los datos en su repositorio UCI [8], se indica que no hay datos faltantes en los datos.

5.6. Datos extremos

La eliminación de los datos extremos es una operación delicada, porque podemos llegar a eliminar datos que no son extremos y que son útiles para conseguir unos modelos de mayor calidad, que tienen en cuenta una mayor cantidad de situaciones.

Para eliminar los outliers he usado la función LocalOutlierFactor de la librería scikit learn. Esta función calcula la densidad local de cada ejemplo, respecto a un cierto número de vecinos. Cuanto mayor se cree que va a ser el número de outliers, se debe aumentar el número de vecinos. En la guía de scikit learn [9] se indica que si no se sabe con cierta certeza el grado de outliers que hay en los datos, un valor adecuado para el número de vecinos es 20, que es el valor por defecto; con este valor se trabajará bien en general. Como yo no tengo claro el grado de outliers,

he utilizado ese valor, y además no he indicado la contaminación. La métrica para la distancia se ha dejado por defecto porque no he encontrado ninguna ventaja en cambiar la métrica.

Los principales argumentos de esta función son:

- n_neighbors: número de vecinos a tener en cuenta en el calculo de la densidad local
- metric: es la métrica con la que se mide la distancia entre el ejemplo y los vecinos (distancia manhattan, distancia euclídea, etc)
- contamination: es el grado de outliers que se espera que haya en los datos

Esta función devuelve un vector de enteros negativos, con un tamaño igual al número de ejemplos; que representa la densidad local del ejemplo. Si el valor es mayor o igual que -1, se considera que el ejemplo es un inlier. Si el valor es menor que -1 se considera que el ejemplo es un outlier.

Para reducir el riesgo de eliminar ejemplos importantes, he decidido eliminar los ejemplos que tenga un valor de densidad menor que el valor del tercer cuartil más 10 veces la distancia intercuartílica.

Con este criterio se han eliminado 42 ejemplos (0.09 %).

6. Justificación de la métrica de error

La función de error usada en los dos tipos de modelos la he indicado en el apartado [3].

Dado que se trata de un problema de clasificación, he decidido coger como métrica para valor las soluciones el accuracy.

El cálculo del accuracy se realiza con la siguiente fórmula:

$$accuracy(y, \hat{y}) = \frac{1}{n_{samples}} \sum_{i=0}^{n_{samples}-1} 1(\hat{y}_i = y_i)$$

Siendo n el número de ejemplos, $\hat{y}_i$ la clase predicha, e y_i la clase verdadera.

En resumen se obtiene el porcentaje de predicciones correctas comparando con las salidas correctas respecto del número total de predicciones. Cuanto mayor es este porcentaje mejor es el ajuste.

7. Justificación de los hiperparámetros

7.1. Modelos

7.1.1. Perceptron (PLA)

En este método hay que fijar los hiperparámetros siguientes:

Tasa de aprendizaje inicial (eta0)

Este parámetro indica el tamaño de los descensos del gradiente. Cuanto mayor sea su valor, mayores son los descensos en cada iteración. Este parámetro es igual a 1, tal y como se indica en la teoría.

Iteraciones máximas

Este parámetro indica el número máximo de iteraciones que realiza la técnica SGD. Para calcular este valor máximo se usa la siguiente fórmula:

$$max\_iter = np.ceil(\frac{10^6}{n}) * 3$$

Siendo n el número de ejemplo del problema.

Esta fórmula la he tomado de la guía de SGD [5] con la única modificación de multiplicar por dos. Hago esta multiplicación porque con la fórmula por defecto que aparece en la guía me daba warnings con algunas hipótesis, por lo que he decidido aumentar el límite máximo un poco. En el caso de este problema es igual a 66.

Penalty

Este parámetro indica que tipo de regularización se va a realizar junto con la técnica SGD. Hay tres opciones: L1, L2 ó Elactic net.

En mi caso sólo he probado con L1 y L2, ya que son regularizaciones más sencillas y que además se han visto en teoría.

Ambas regularizaciones son explicadas en el apartado [7.2].

A parte se debe fijar el alpha para la regularización, pero este hiperparámetro se explica en la sección de regularización [7.2].

7.1.2. Regresión Logística

En este método hay que fijar los hiperparámetros siguientes:

Función de pérdida

Como función de pérdida he elegido la sigmoidal, tal y como se explica en el apartado [3].

Iteraciones máximas

Este parámetro indica el número máximo de iteraciones que realiza la técnica SGD. Para calcular este valor máximo se usa la siguiente fórmula:

$$max\_iter = np.ceil(\frac{10^6}{n}) * 2$$

Siendo n el número de ejemplo del problema.

Esta fórmula la he tomado de la guía de SGD [5] con la única modificación de multiplicar por dos. Hago esta multiplicación porque con la fórmula por defecto que aparece en la guía me daba warnings con algunas hipótesis, por lo que he decidido aumentar el límite máximo un poco. En el caso de este problema es igual a 44.

Penalty

Este parámetro indica que tipo de regularización se va a realizar junto con la técnica SGD. Hay tres opciones: L1, L2 ó Elactic net.

En mi caso sólo he probado con L1 y L2, ya que son regularizaciones más sencillas y que además se han visto en teoría.

Ambas regularizaciones son explicadas en el apartado [7.2].

Tasa de aprendizaje (eta)

Como forma de calcular la tasa de aprendizaje en cada iteración, he elegido que se mantenga constante, por lo que el valor indicado en el eta inicial, se mantendrá durante toda la ejecución del método.

Tasa de aprendizaje inicial (eta0)

Este parámetro indica el tamaño de los descensos del gradiente. Cuanto mayor sea su valor, mayores son los descensos en cada iteración. Para encontrar el valor óptimo he ido haciendo pruebas con distintos valores empezando por 0.01 e ir incrementando en 0.005. Los valores que he escogido para probar con la validación cruzada son 0.015 y 0.02.

A parte se debe fijar el alpha para la regularización, pero este hiperparámetro se explica en la sección de regularización [7.2].

7.2. Regularización

La regularización es un método para evitar realizar overfitting (sobreajuste), es decir, entrenar demasiado con el modelo. Para evitar esta situación, la regularización aplica una penalización a los pesos del modelo. Esta penalización cambia en cada tipo de regularización.

Siempre es bueno realizar regularización sobre un modelo, ya que no afecta en gran medida si no mejora el modelo, y nos proporciona una gran reducción de varianza a cambio de un pequeño incremento en el sesgo.

La regularización L1 o Lasso penaliza la suma del valor absoluto de los coeficientes de regresión. Esta penalización fuerza a que los coeficientes de los predictores tiendan a cero. Cuando un predictor tiene un coeficiente de regresión igual a cero no influye en el modelo, de esta forma esta regularización excluye del modelo a los predictores menos relevantes.

La regularización L2 o Ridge penaliza la suma de los coeficientes elevados al cuadrado. Esta penalización fuerza a reducir de forma proporcional el valor de todos los coeficientes del modelo pero sin que estos lleguen a cero. Que no lleguen a cero los coeficientes es la mayor diferencia con la regularización L1.

La intensidad de ambos tipos de regularización está controlada por el hiperparámetro alpha (λ). Cuando alpha vale cero equivale a no realizar regularización en el modelo. A medida que alpha aumenta, mayor es la penalización y más predictores quedan excluidos.

A priori no se sabe cual será el valor de alpha que nos dará mejores resultados, por ello he ido probando valores comenzando en 1 e ir dividiendo entre 10, con el resto de hiperparámetros fijados como se indica en el apartado de cada algoritmo.

Los valores escogidos para el algoritmo Perceptron son: 1e-09, 1e-10 . Esto se ha decidido comprobando los resultados con distintos valores alpha, los cuáles se pueden ver en la siguiente tabla:

Apha	**Accuracy**
1	0.09
0.001	0.22
1e-05	0.22
1e-07	0.28
1e-09	0.32
1e-10	0.30

Tabla 5: Pruebas con alpha - Perceptron

Los valores escogidos para el algoritmo regresión logística son: 1e-05, 1e-06 . Esto se ha decidido comprobando los resultados con distintos valores alpha, los cuáles se pueden ver en la siguiente tabla:

Apha	**Accuracy**
1	0.09
0.1	0.18
0.001	0.3
0.0001	0.47
1e-05	0.49
1e-06	0.48

Tabla 6: Pruebas con alpha - Regresión Logística

8. Selección de la mejor hipótesis

Para elegir la mejor hipótesis de forma que la elección no sea optimista he usado la técnica de cross-validation. Esta técnica consiste en dividir los datos de entrenamiento en k partes del mismo tamaño, y realizar k veces el ajuste del modelo teniendo todos los fold como datos de training menos uno, y en cada iteración cambia el fold usado como test. En la práctica se realiza el k-fold cross-validation, en vez de usar la técnica Leave One Out, que consiste en dividir todos los datos en grupos de un ejemplo, esta técnica en la práctica es computacionalmente imposible para elegir una hipótesis ya que habría que ajustar tantos modelos como ejemplos hay en los datos.

Las ventajas de usar esta técnica es que nos aseguramos de que no vamos a elegir una hipótesis tomando valores de error optimistas, sino al contrario. Esta técnica nos permite eliminar gran parte de la influencia de los datos usados para entrenar sobre el modelo, para obtener un modelo con más generalidad y cuyo error será tomado de forma pesimista.

De cada hipótesis se obtiene la media del error de cross-validation en cada iteración de la técnica.

La mejor hipótesis será la que obtenga un menor error de cross-validation.

Las distintas hipótesis serán los distintos modelos con todas las combinaciones de hiperparámetros definidos en el apartado [7].

Para realizar k-fold cross-validation se ha usado la función GridSearchCV de la librería scikit learn [10]. Esta función tiene los siguientes parámetros importantes:

- estimator: indica la técnica a usar para ajustar los modelos
- param_grid: lista que indica para los distintos valores de los parámetros del estimador. Los valores de los parámetros que no se indican en esta lista se dejan por defecto.
- cv: número que indica el número de folds a usar en la validación cruzada
- error_score: métrica de error para valorar las hipótesis
- return_train_score: valor booleano que indica si se devuelve o no el error de training de los modelos ajustados en la validación cruzada. Si se pone True, se devuelve este error, pero hay que tener en cuenta que esto llevará un sobrecoste en el tiempo que necesitará la técnica. Si se pone False seguirá devolviendo el mejor modelo sin ningún problema y tardará menos, pero no dispondremos de la información de training.

Como estimador yo he elegido los métodos indicados en el apartado [3].

Como parámetros se han puesto los valores de los hiperparámetros indicados en el apartado [7] para cada método.

Se ha realizado una validación cruzada con 5 folds, por lo que he realizado una 5-fold cross-validation, ya que cuanto mayor sea el número, mayor generalidad se obtiene, pero mayor coste en tiempo conlleva ya que hay que realizar más ajustes; pero como en este problema debemos usar la técnica OneVSAll para convertir el problema de clasificación multiclase en varios problemas

de clasificación binarios, he creído que era más conveniente realizar una 5-fold cross-validation, dado que con 10 folds el tiempo de computo se disparaba. Lo más general es usar 5 o 10 folds.

Como métrica de error para evaluar un modelo se ha usado el accuracy, tal y como se indica en el apartado [6]. El error de validación cruzada se aproxima al E_out de la hipótesis.

Y se indica que no se devuelvan los errores obtenidos para training, para no malgastar recursos en obtener datos que no voy a usar.

Para cada método he obtenido el E_cv de todas las hipótesis de cada método.

Para el algoritmo Perceptron, la mejor hipótesis según el error de validación cruzada es el siguiente:

Penalty	L2
Alpha	1e-09
Eta	1.0
N_iter	27
Max_iter	66
E_cv	0.33

Tabla 7: Mejor hipótesis del algoritmo Perceptron

Para el algoritmo regresión logística, la mejor hipótesis según el error de validación cruzada es el siguiente:

Penalty	L2
Alpha	1e-05
Eta	0.015
N_iter	26
Max_iter	44
E_cv	0.62

Tabla 8: Mejor hipótesis del algoritmo regresión logística

Viendo la mejor hipótesis de ambos métodos, nos quedamos con la mejor hipótesis del algoritmo regresión logística, que es la que mayor accuracy tiene.

9. Cálculo del error de la hipótesis final

Una vez escogida la hipótesis final, ahora debo volver a entrenar el modelo elegido pero con todos los datos de training, sin realizar la validación cruzada.

Antes de obtener las predicciones de los datos de test, se deben preprocesar los datos de entrada de test, pero utilizando los datos de training para ello.

Una vez preprocesados los datos, obtengo las predicciones de los datos de test, y obtengo el accuracy en base a las predicciones y a las salidas verdaderas de test.

El accuracy de test obtenido es 0.48. Este E_test se aproxima al E_out de la hipótesis final.

Para poder comparar este error y ver si es bueno o no, he calculado el accuracy de un clasificador aleatorio.

El accuracy obtenido para este clasificador aleatorio es 0.09, este accuracy que se podría considerar como el mínimo accuracy que se debería obtener.

Dado que el accuracy del modelo ajustado es mucho mejor que el accuracy del clasificador aleatorio considero que es un buen ajuste, aunque no es perfecto, ya que no llega ni al 50 % de aciertos. Tal y como se puede ver en la siguiente matriz de confusión, donde se pueden viasualizar las predicciones correctas e incorrectas:

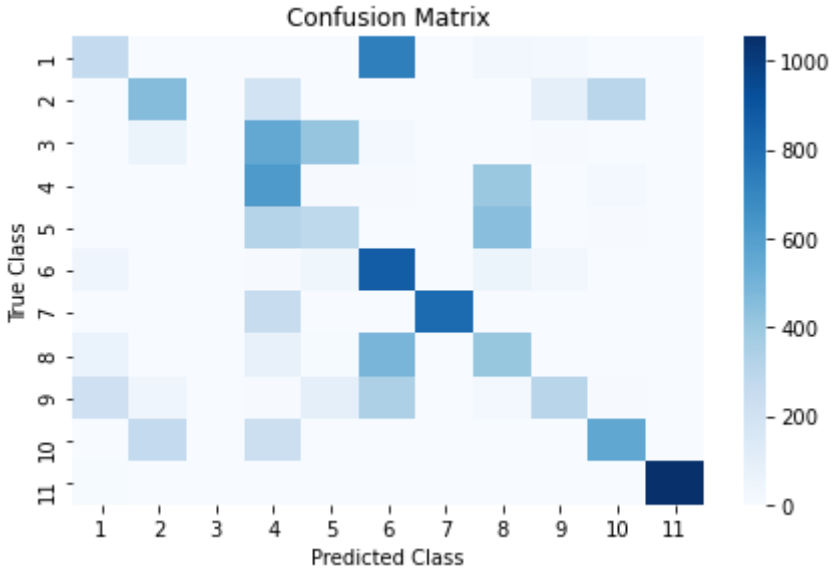

Figura 6: Matriz de confusión

En esta gráfica se puede apreciar como una gran cantidad de puntos no se encuentran en la diagonal principal de la matriz, que representa las predicciones donde se acierta. Además se pude destacar como las predicciones de la clase 11 son en un alto porcentaje correctas y como las predicciones de la clase 4 son en un bajo porcentaje correctas.

Referencias

[1] Tobias Grüner, Falco Böllhoff, Robert Meisetschläger, Alexander Vydrenko, Martyna Bator, Alexander Dicks y Andreas Theissler. "Evaluation of Machine Learning for Sensorless Detection and Classification of Faults in Electromechanical Drive Systems". En: *Procedia Computer Science* 176 (2020). Knowledge-Based and Intelligent Information Engineering Systems: Proceedings of the 24th International Conference KES2020, págs. 1586-1595. ISSN: 1877-0509. DOI: https://doi.org/10.1016/j.procs.2020.09.170. URL: https://www.sciencedirect.com/science/article/pii/S1877050920320706.

[2] *Manual de Numpy: Corrcoef*. URL: https://numpy.org/doc/stable/reference/generated/numpy.corrcoef.html.

[3] *Manual de Scikit-learn: OneVsRestClassifier*. URL: https://scikit-learn.org/stable/modules/generated/sklearn.multiclass.OneVsRestClassifier.html.

[4] *Manual de Scikit-learn: Perceptron*. URL: https://scikit-learn.org/stable/modules/generated/sklearn.linear_model.Perceptron.html.

[5] *Guia de Scikit-learn: SGD*. URL: https://scikit-learn.org/stable/modules/sgd.html#sgd.

[6] *Manual de Scikit-learn: SGD Classifier*. URL: https://scikit-learn.org/stable/modules/generated/sklearn.linear_model.SGDClassifier.html.

[7] *Manual de Scikit-learn: MinMaxScaler*. URL: https://scikit-learn.org/stable/modules/generated/sklearn.preprocessing.MinMaxScaler.html.

[8] *UCI: Sensorless Drive Diagnosis Data Set*. URL: https://archive.ics.uci.edu/ml/datasets/Dataset+for+Sensorless+Drive+Diagnosis.

[9] *Guia de Scikit-learn: Local Outlier Factor*. URL: https://scikit-learn.org/stable/modules/outlier_detection.html.

[10] *Manual de Scikit-learn: GridSearchCV*. URL: https://scikit-learn.org/stable/modules/generated/sklearn.model_selection.GridSearchCV.html.