Doble Grado en Ingeniería Informática y Matemáticas

APRENDIZAJE AUTOMÁTICO

Práctica 3: Programación

UNIVERSIDAD DE GRANADA

Alberto Estepa Fernández - *albertoestep@correo.ugr.es*

Mayo de 2020

Índice

1. Optical Recognition of Handwritten Digits Data Set

1.1. Comprensión del problema

El objetivo del problema es clasificar imágenes de los distintos dígitos pertenecientes al sistema de numeración arábico. El dataset proporcionado consta de un conjunto de imágenes (debidamente codificadas) de dígitos manuscritos y queremos precedir para nuevas imágenes de que dígito se trata.

Nos han proporcionado 2 ficheros (*optdigits.tes*, que contiene los datos de 'test' y *optdigits.tra* para los datos de 'train'). Son ficheros que tienen el mismo formato que un archivo CSV (comma-separated values), es decir, en las que las columnas se separan por comas y las filas por saltos de línea.

Los dígitos son mapas de bits de 32x32 (donde un píxel es blanco o negro). Se dividen en bloques no superpuestos de 4x4 y se cuenta el número de píxeles negros en cada bloque. Esto genera una matriz de entrada de 8x8 donde cada elemento es un número entero en el rango entre 0 y 16. Esto reduce la dimensionalidad y da invariancia a pequeñas distorsiones.

La etiqueta de cada dato es un número entero entre 0 y 9 inclusive, que indica el dígito con el que se representa.

Estamos por tanto afrontando un problema de aprendizaje supervisado donde las características X es la matriz de entrada 8x8 mencionada anteriormente ($X = \{0, \ldots, 16\}^{8x8}$) y la etiqueta y es un número entre 0 y 9 que indica el dígito con el que se representa ($y = \{0, \ldots, 9\}$). La función de clasificación f es aquella que hace corresponder cada matriz de características X con su correspondiente etiqueta y.

1.2. Selección de las clase/s de funciones a usar.

Como hemos visto, tenemos un total de 64 características a analizar. Usaremos combinaciones lineales de éstas características ya que podríamos sobrecargar el modelo con funciones de grado superior o no lineales, además intentamos evitar el sobreajuste usando estas combinaciones lineales, aunque podemos pecar de 'underfitting' o subajuste. Todo ésto lo analizaremos cuando tengamos los resultados del modelo. Por otro lado al ser un problema de clasificación multinomial, las probabilidades se obtienen con la función softmax [11], es decir

$$P[y = j|x] = \frac{\exp(w_j^T x)}{\sum_{i=0}^{9} \exp(w_i^T x)}$$

donde $j = \{0, \ldots, 9\}$. Así cada $h \in \mathcal{H}$ se expresa como:

$$h(x) = \{j : P[y = j|x] = \max_i P[y = i|x]\}$$

1.3. Fijamos conjuntos de training y test.

El dataset proporcionado está dividido en dos ficheros como hemos explicado: *optdigits.tes* para los datos de 'test' y *optdigits.tra* para los datos de 'train'. Si contamos el número de instancias de entrenamiento, tenemos 3823 muestras, y número de instancias de test es de 1797. Así pues el porcentaje de datos de entrenamiento con respecto al total es del 68.02491103202847 %, y el porcentaje de datos de test con respecto al total es del 31.97508896797153 %. De esta forma tenemos particiones bastante óptimas siguiendo las recomendaciones dadas. Usaremos la técnica cross-validation para escoger los hiperparámetros del clasificador.

1.4. Preprocesado de los datos.

Como hemos comentado los datos proporcionados estaban preprocesados anteriormente y no era esperable que encontrásemos valores perdidos u 'outliers'. Aún así hemos comprobado que no tengamos valores perdidos (el código de ésta comprobación se puede encontrar en el fichero *clasificacion.py*) y nuestra intuición era cierta:

```
Valores perdidos en el dataset de entrenamiento: 0
Valores perdidos en el dataset de test: 0
```

El tema del tratamiento de 'outliers' es más dificultoso pues haría falta un análisis minucioso de la conversión de las matrices 32x32 de 1's y 0's originales a las matrices finales de 8x8 donde los valores oscilan entre 0 y 16 (que recordemos que indican el número de píxeles negros del bloque 4x4 al recorrer la imagen original). Así creemos conveniente que para tratar los posibles 'outliers' debemos comprobar que los valores de las características sean valores enteros entre 0 y 16 y que las etiquetas sean valores enteros entre 0 y 9. Ésto es lo que hemos hecho (se puede consultar el código en el fichero *clasificacion.py*).

```
Todos los valores de entrenamiento son enteros: Sí
Todos los valores de test son enteros: Sí
Todos los valores de las características de entrenamiento pertenecen
al intervalo: [0,16]
Todos los valores de las características de test pertenecen al
intervalo: [0,16]
```

```
Todos los valores de las etiquetas de entrenamiento pertenecen al
intervalo: [0,9]
Todos los valores de las características de test pertenecen al
intervalo: [0,9]
```

Como sabemos una de las principales razones del 'overfitting' o sobreajuste, es la existencia de redundancia en los datos. Esto hace que el modelo sea demasiado complejo con respecto a los datos de entrenamiento facilitados, y por tanto incapaz de hacer generalizaciones correctas en datos no procesados aún. Para solucionar ésto vamos a aplicar la técnica de Análisis de Componentes Principales (PCA - Principal Component Analysis) para reducir la dimensionalidad del dataset y así simplificar la modelización posterior. Al aplicar esta técnica se asume que los datos de trabajado tienen una distribución gaussiana o normal así que nosotros vamos a aplicar a los datos una transformación de normalización de forma que su media sea igual a 0, y su varianza=1. Para ello, usaremos la transformación *Standardscaler* de Sklearn. Aplicamos el algoritmo PCA de Sklearn indicándole que seleccione el número de componentes de modo que la cantidad de varianza que deba explicarse sea mayor a 0.98 %. Hemos de decir que hemos decidido usar PCA una vez que hemos comprobado que utilizándolo obteníamos mejores resultados que sin usarlo y, debido a que las características son bastante abstractas, PCA es el mejor método de reducción de la dimensionalidad para nuestro problema. [6]

Analizando el problema con cierta perspectiva, es probable que haya zonas de las imágenes que no aporten gran información en nuestro problema, como las esquinas, por lo que es lógico que al aplicar PCA con un porcentaje de la varianza explicada menor que 1 pero cercano a éste, se reduzca la dimensionalidad del problema. Ésto rebajará el coste computacional y como hemos dicho anteriormente, podrá rebajar el 'overfitting' de nuestro modelo, permitiendo una mayor generalización. Así pues comprobamos que hemos reducido la dimensionalidad del problema:

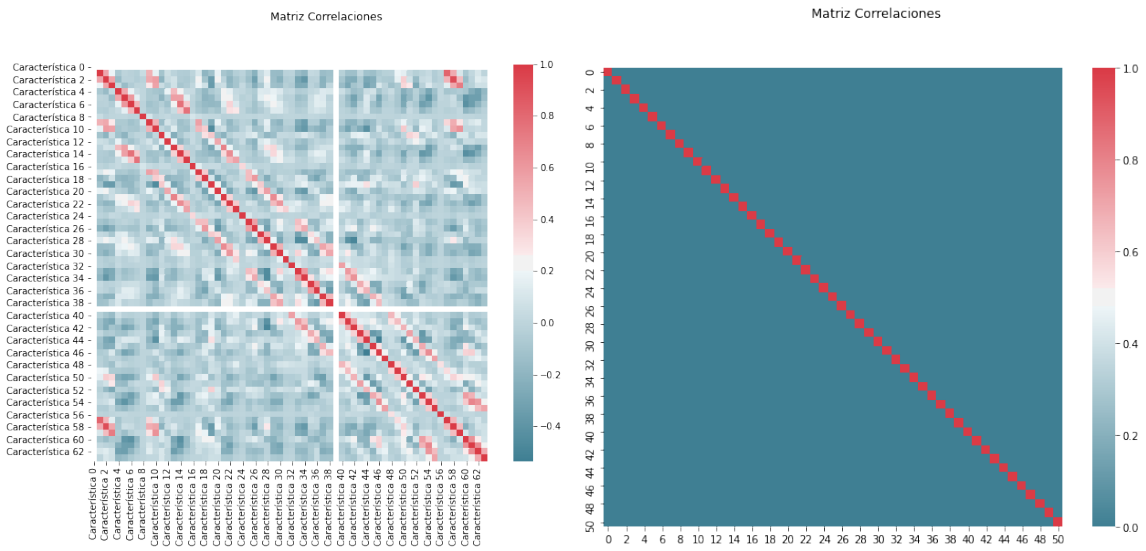
```
El número de características actuales es de: 51
```

Por otro lado haremos uso de la matriz de correlaciones entre los distintos elementos. Sabemos que un valor de correlación alto y positivo entre cada par de elementos indica que los elementos miden la misma destreza o característica. Si los elementos no están altamente correlacionados, entonces los elementos pudieran medir diferentes características o no estar claramente definidos.

Adjuntamos imágenes de la matriz de correlación antes de procesar los datos y después de éste paso:

No creemos conveniente eliminar instancias del dataset, ya que no parece demasiado grande y podemos afrontar su coste computacional, además que todas las clases tienen una cantidad bastante similar de intancias que las representan:

```
Dataset de entrenamiento:
```



(a) Matriz de correlación de los datos originales (b) Matriz de correlación de los datos procesados

Figura 1: Matrices de correlación

```
 376 instancias del dígito 0
389 instancias del dígito 1
380 instancias del dígito 2
389 instancias del dígito 3
387 instancias del dígito 4
376 instancias del dígito 5
377 instancias del dígito 6
387 instancias del dígito 7
380 instancias del dígito 8
382 instancias del dígito 9

Dataset de test:

 178 instancias del dígito 0
182 instancias del dígito 1
177 instancias del dígito 2
183 instancias del dígito 3
181 instancias del dígito 4
182 instancias del dígito 5
181 instancias del dígito 6
179 instancias del dígito 7
174 instancias del dígito 8
180 instancias del dígito 9
```

Así pues tenemos una información uniforme de todas las clases y no creemos necesarios nuevos datos para completar información o eliminar datos para hacer más uniforme la distribución de los datos.

Además los datos están correctamente discretizados, por lo que no consideramos conveniente transformar los datos de ninguna otra forma. Damos así por acabado el preprocesado de los datos.

1.5. Métrica de error a usar.

Dado que tenemos entre manos un problema de clasificación, vamos a usar la matriz de confusión como elemento de visualización del error y alguna de sus correspondientes métricas que nos proporciona. Recordemos que la matriz de confusión es una tabla que describe el rendimiento de un modelo supervisado de Machine Learning. La matriz de confusión en sí misma no es una medida de métrica como tal, pero casi todas las métricas se basan en la matriz de confusión y los números dentro de ella.

Nuestro problema no tiene unas grandes restricciones sobre si queremos que clasifique siempre correctamente una cierta clase siendo más flexibles con otras (no debe que ser estrictamente preciso con una clase y no con las demás), por lo que la precisión y el recall no nos interesan especialmente, aunque podían complementar perfectamente nuestra métrica de error.

Dado que nuestros datos están distribuidos equitativamente sobre cada una de las clases, usaremos la métrica 'accuracy' como métrica de error, que es el porcentaje total de elementos clasificados correctamente. Dicha métrica va a resultar significativa por lo anterior descrito: al estar los datos equitativamente sobre cada una de las clases no existirá la posibilidad de escoger siempre la clase más frecuente como elemento a predecir y que la métrica tenga un buen valor.

Cuando obtengamos los resultados del entrenamiento y la predicción analizaremos este tema con más detalle.

1.6. Discusión de la técnica de ajuste elegida.

Anticipándonos a los acontecimientos, aunque mi idea original era utilizar una regresión logística he decidido probar como ampliación otro algoritmo para entrenar nuestro problema y discutir los resultados: una regresión logística y un lineal support-vector machine (SVM lineal). Ésto se explicará con detalle en la sección 1.8. A continuación escogeremos la técnica de ajuste para uno y otro modelo, ya que el modelo influye en la técnica de ajuste elegida:

Por un lado, para el método de regresión logística, buscando información sobre que técnica de ajuste utilizar, encontré el método de optimización L-BFGS que es análogo al Método de Newton, pero que elimina uno de sus principales inconvenientes: el coste computacional del cálculo de la matriz Hessiana. En éste método la matriz Hessiana se aproxima usando actualizaciones especificadas por evaluaciones de gradiente (o evaluaciones de gradiente aproximadas). En otras palabras, el método usa una estimación de la matriz Hessiana inversa. Además que almacena solo unos pocos vectores que representan la aproximación implícitamente, optimizando el espacio en memoria. Después de realizar varias pruebas y comprobando que obtengo mejores resultados para nuestro problema con él, usaremos dicha técnica de ajuste. [7]

Por otro lado, para el método SVM, he decidido usar la función de pérdida 'Hinge' es decir, $max\{0, 1 - y\omega^T x\}$ que es la estándar para dicho método. Ésta es la mejor para dicho método ya que tiene en cuenta tanto que el método clasifique correctamente como el espacio entre la frontera de clasificación y los vectores de soporte (a mayor espacio, mejor resultados tendrá). La función de Hinge tiene la propiedad de ser convexa, pero no es derivable en 0, lo que hace que un algoritmo de optimización basado en gradiente no se pueda usar directamente. [8]

Así usaremos el algoritmo de optimización que utiliza el modelo LinearSVM de la libreía Sklearn, que es el descenso coordinado. Es un algoritmo de optimización que minimiza sucesivamente a lo largo de las direcciones de coordenadas para encontrar el mínimo de una función. En cada iteración, el algoritmo determina una coordenada o un bloque de coordenadas a través de una regla de selección de coordenadas, luego minimiza de manera exacta o inexacta sobre el hiperplano de coordenadas correspondiente mientras fija todas las demás coordenadas o bloques de coordenadas. Se puede realizar una búsqueda de línea a lo largo de la dirección de coordenadas en la iteración actual para determinar el tamaño de paso apropiado. El descenso coordinado es aplicable tanto en contextos diferenciables como libres de derivadas, lo cual es perfecto para nuestro problema. [9]

1.7. Discusión de la necesidad de regularización.

Como hemos comprobado, el dataset tiene una alta dimensionalidad, es decir, gran cantidad de características, lo que puede provocar sobreajuste si el modelo es complejo. Para evitar dicha complejidad es necesario hacer uso de la regularización: un método que penaliza la complejidad del modelo, introduciendo un término en la función de coste tal que si antes de regularizar la función de coste era J, ahora resultará que es $J + \alpha C$, donde α indica cómo de importante es para nosotros que el modelo sea simple en relación a cómo de importante es su rendimiento y C es la medida de complejidad del modelo, y la escogeremos posteriormente mediante validación cruzada ya que será un hiperparámetro del modelo. Esto provoca que el

modelo sea más simple y pueda generalizar mejor. [10]

Como medida de complejidad del modelo, las más importantes son:

- Regularización Lasso (L1): Es la media del valor absoluto de los coeficientes del modelo, es decir:

$$\frac{1}{N+1}\sum_{j=0}^{N}|\omega_j|$$

  y será efectiva cuando algunas características sean irrelevantes. Al usar la regularización L1 favorecemos que algunos de los coeficientes acaben valiendo 0, es decir nos puede ayudar a hacer la selección de características. Lasso funciona mejor cuando los atributos no están muy correlados entre ellos.

- Regularización Ridge (L2): Es la media del cuadrado de los coeficientes del modelo, es decir:

$$\frac{1}{2(N+1)}\sum_{j=0}^{N}\omega_j^2$$

  y será efectiva cuando algunas característica estén correladas entre ellas. Al usar la regularización L2 los coeficientes acaban siendo más pequeños lo que minimiza el efecto de la correlación entre los atributos de entrada y hace que el modelo generalice mejor. Ridge funciona mejor cuando la mayoría de los atributos son relevantes.

De esta forma y situándonos en nuestro problema, hemos visto que todos los atributos son relevantes pues es información local de la imagen y eliminarla provocaría extraer menos información de la imagen dada. Creemos así que la regularización que mejor resultados nos puede proporcionar con respecto a nuestro problema es la regularización Ridge o L2.

1.8. Identificación de los modelos a usar.

Como anteriormente hemos comentado, vamos a usar una regresión logística para entrenar los datos y poder predecir los resultados, y como complemento y ampliación lo compararemos con un lineal support-vector machine (SVM lineal).

Para la regresión logística, queremos que permita separar más de dos clases. Para ello tenemos dos posibilidades [4]:

- One vs Rest: Realiza tantos problemas de clasificación binaria como posibles clases haya (en nuestro caso 10, tantas como dígitos diferentes). Cada uno de ellos aplica una regresión logística binaria entre una clase contra el resto de ellas, diferenciandose cada problema en la clase escogida para aplicar la regresión.

- Multinomial: Realiza un problema de clasificación simultaneamente para todas las clases. A la hora de predecir la clase de una muestra, el modelo calcula un 'score' para cada clase y hace pasar el vector de 'scores' por la función softmax (o exponencial normalizada), que estima las probabilidades de cada clase y devuelve la clase correspondiente a la máxima probabilidad estimada.

Escogemos el modelo de Regresión Logística Multinomial ya que es más fácilmente interpretable en un entorno probabilístico y además, aunque no podemos afirmarlo con seguridad, es lógico pensar que las clases son mutuamente excluyentes (recordemos que la entrada son matrices 1's y 0's, que indican el trazo de un dígito manuscrito, a las que se ha pasado un filtro que cuenta los números de 1's de la imagen en el filtro, por lo que visualmente entendemos que son diferentes entre sí). Ésta intuición de que las clases sean mútuamente excluyentes decanta la balanza a elegir éste modelo de Regresión Logística que sabemos que tiene mejor efectividad frente al otro si las clases son mutuamente excluyentes.

Por otro lado, y de la misma forma que en la regresión logística, tenemos varias posibilidades para los SVMs [4]:

- One vs Rest: Realiza tantos problemas de clasificación binaria como posibles clases haya (en nuestro caso 10, tantas como dígitos diferentes). Cada uno de ellos aplica un SVM lineal entre una clase contra el resto de ellas, diferenciandose cada problema en la clase escogida para aplicar la regresión.

- Crammer-Singer: propone un método SVM multiclase que arroja el problema de clasificación multiclase en un solo problema de clasificación simultáneo, en lugar de descomponerse en múltiples problemas de clasificación binaria. [1]

De forma análoga, escogeremos el modelo de SVM basado en Crammer-Singer como modelo de clasificación de los datos.

A continuación realizaremos una estimación de los hiperparámetros y seleccionaremos el mejor modelo resultante definiendo todas sus características.

1.9. Estimación de hiperparámetros y selección del mejor modelo.

Vamos a usar la validación cruzada para ajustar los hiperparámetros de los modelos. Sabemos que, al aplicar la validación cruzada, el conjunto de datos de entrenamiento se divide en grupos de igual tamaño. Una vez realizada la partición se procede a entrenar el modelo una vez por cada uno de los grupos. Utilizando todos los grupos menos el de la iteración para entrenar y este para validar los resultados. Una diferencia importante con la validación fuera de muestra es que en esta ocasión se entrena y valida con todos los datos, cambiando el conjunto utilizado para la validación en cada iteración. Así es posible identificar si los modelos son inestables

o estables, es decir, si el resultado depende de los datos utilizados o no. En caso de que los resultados dependan de los datos utilizados, el modelo posiblemente estará siendo siendo sobreajustado. Indicando que el modelo empleado dispone de demasiados grados de libertad.

Como hemos dicho usaremos el clasificador *LogisticRegression* de la librería Sklearn [4]. Los parámetros del clasificador serán:

- *penalty='l2'*: Regularización Ridge (L2). Explicado en la sección 1.7.
- *multi_class='multinomial'*: Indicamos que la regresión logística es multinomial. Explicado en la sección 1.8.
- *max_iter = 1000*: Número máximo de iteraciones del algoritmo de optimización. Se ha probado que es el número correcto de iteraciones necesarias para que el algoritmo de optimización converja.
- *solver = 'lbfgs'*: Algoritmo a utilizar en el problema de optimización (el algoritmo es L-BFGS por defecto). Explicado en la sección 1.6.
- Los demás valores se han dejado por defecto. Se pueden consultar en la salida del código, ya que hemos usado la función *get_params* del clasificador y permitimos visualizar su salida.

Por otra parte, para el modelo de ampliación a los pedidos, usaremos el clasificador *LinearSVC* de la librería Sklearn [4]. Los parámetros del clasificador serán:

- *penalty='l2'*: Regularización Ridge (L2). Explicado en la sección 1.7.
- *multi_class='crammer_singer'*: Indicamos que el SVM multiclase es el propuesto por Crammer-Singer. Explicado en la sección 1.8.
- *loss = 'hinge'*: Función de pérdida Hinge. Explicado en la sección 1.6.
- Los demás valores se han dejado por defecto. Se pueden consultar en la salida del código, ya que hemos usado la función *get_params* del clasificador y permitimos visualizar su salida.

Realizaremos validación cruzada en una primera instancia y como parte oficial de la práctica, sobre el modelo de regresión logística para estimar el mejor valor del hiperparámetro C, que como hemos dicho en la sección 1.7 mide la influencia de la complejidad del modelo. Dividiremos en 5 conjuntos para la validación. Ésto es porque reserva cerca del 15 % del total de las instancias para validar, un valor apropiado para el conjunto de validación teniendo en cuenta el tamaño del conjunto de entrenamiento.

El mejor clasificador, basandonos en la métrica 'accuracy' para los datos de entrenamiento es la regresión logística con $C = 0.1$, obteniendo un error de validación

de 0.03583410327481784.

Anotamos que si quisieramos hacer una estimación de cuál es el mejor modelo entre la regresión logística y el SVMc lineal que hemos planteado durante las secciones posteriores, deberíamos haber realizado una validación cruzada con la misma separación de los datos para los conjuntos de validación y aplicarla a los dos modelos. Posteriormente (en la sección 1.11) se discutirá ésto.

1.10. Estimación por validación cruzada del error del modelo.

Una vez realizada la validación cruzada entrenando los distintos modelos con cada hiperparámetro, hemos obtenido distintos errores de validación y hemos escogido el mejor de acuerdo a nuestra métrica ('accuracy') para poder así elegir el modelo resultante (todo ésto lo hemos hecho mediante la función $GridSearchCV$ de Sklearn). Así entrenado, con los datos de entrenamiento, éste modelo elegido, vamos a predecir los resultados de los datos de test y a compararlos con su correspondiente salida.

Podemos medir el error del clasificador en los datos de entrenamiento (E_{in}) (aunque ésto no es pedido en la práctica) y estimamos el error del clasificador en los datos de test (E_{test}) que nos servirá para estimar E_{out}, obteniendo:

$$E_{in} : 0.020664399686110335$$
$$E_{test} : 0.0506399554813578$$

Así, por la desigualdad de Hoeffding, tenemos con probabilidad $1 - \delta$ (transparencias de teoria, diapositiva 22 Tema 3), una cota del error:

$$E_{out} \leq E_{test} + \sqrt{\frac{1}{2N_{test}} \log \frac{2|H|}{\delta}}$$

Aplicando ésta fórmula, sabiendo que $|H| = 1$, ya que hemos estimado la función anteriormente y escogido una concreta, tenemos que con probabilidad de un 95 %:

$$E_{out} : 0.08267742421603491$$

Como vemos obtenemos un acierto de fuera de la muestra de entrenamiento de más de 0.91 sobre 1, lo cual es un resultado bastante bueno y quiere decir que nos clasifica más de 9 de cada 10 muestras correctamente, que, recordando la complejidad de las muestras (imágenes de números escritos a mano), y que visualmente pueden ser confusas de diferenciar entre sí, lo consideramos un resultado bastante aceptable.

A pesar de ésto comprobamos que el error de entrenamiento con respecto al de test es considerablemente menor. Ésto puede ser porque haya un sobreajuste del modelo a éstos datos de entrenamiento.

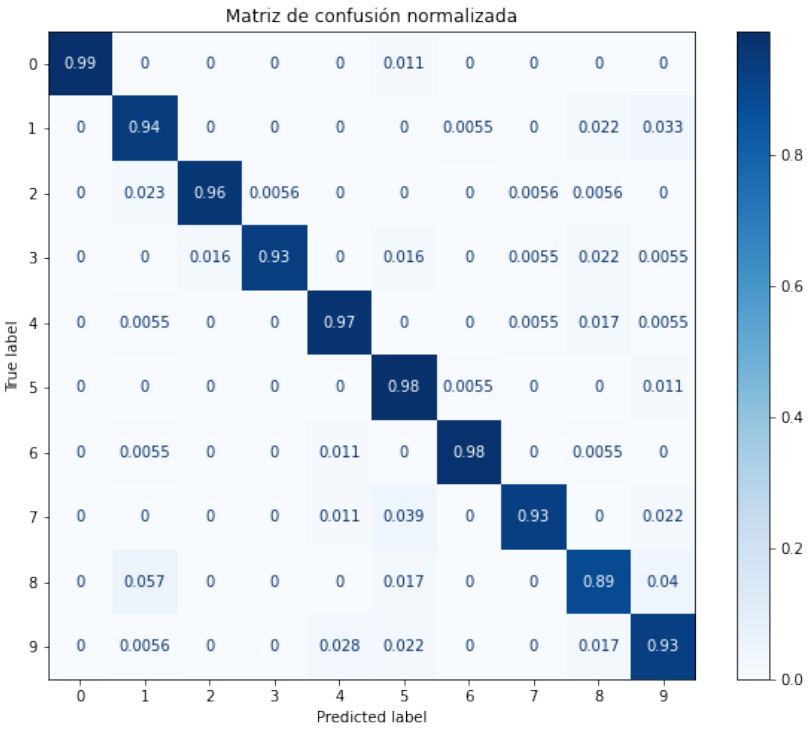


Figura 2: Matriz de confusión normalizada

Utilizamos la matriz de confusión para ver donde nuestro modelo ha tenido errores en la clasificación, pero para nuestro caso hemos normalizado los datos de la matriz entre 0 y 1 para poder visualizar cada casilla de forma general.

Como podemos comprobar, tenemos más errores en clasificar correctamente muestras correspondientes a la clase del dígito 8, lo que visualmente es lógico pues comparte muchos trazos con los otros dígitos. Otra clase que nos da más problemas que lo habitual (que la media) es la clase del dígito 7, que es bastante confundida con la clase del dígito 5, cuya causa puede ser la reducción de dimensionalidad a la que sometimos los datos, eliminando alguna característica que permitía distinguir con mayor claridad éstas dos clases (pero como hemos discutido hemos obtenido otras ventajas al realizar dicha reducción de la dimensionalidad, por lo que aceptamos tener dicha pérdida de exactitud en éstas dos clases).

Hay clases sin embargo que éstas casi correctamente clasificadas en su totalidad, como son la clase del dígito 0, la clase del dígito 5 o la clase del dígito 6.

1.11. Modelo propuesto para la empresa

Hasta ahora hemos estudiado una clase de modelos para este problema. Podríamos haber cogido varias clases (ahora es donde entra en juego el SVM lineal, por ejemplo). Si hubiesemos realizado una validación cruzada con la misma separación de los datos para los conjuntos de validación y la hubiesemos aplicado a los dos modelos, hubiesemos obtenido el mejor modelo resultante. Sin embargo al haber entrenado ya los datos con nuestro modelo de regresión logística ya no podemos establecer ésta comparación en igualdad de condiciones.

Si entrenamos el SVM lineal haciendo validación cruzada sobre los datos de entrenamiento para obtener los hiperparámetros obtenemos un error de validación de 0.03766622181158685, es decir, muy cercano al que obtuvimos con el de regresión logística. Pero como hemos dicho, éste dato no permite compararlos y decir cual es el mejor modelo, porque no se han entrenado en igualdad de condiciones.

Sin embargo si podemos decir que ambos modelos representan de forma adecuada a los datos de nuestro problema y la calidad de los dos modelos es bastante buena por lo explicado en la sección anterior.

No obstante nuestro modelo no es el que proporciona el mejor error. Por ejemplo, en la documentación proporcionada se argumenta que si implementamos un K-NN usando la distancia Euclídea como métrica obtenemos resultados incluso mejores. Sin embargo ésta clase de modelos no es lineal, la complejidad es muy superior y el modelo es más difícil de entrenar llegando a tener mayor riesgo de sobreajuste.

Así la decisión final de un modelo propuesto para la empresa dependerá de las restricciones que nos ponga la empresa:

- Si la empresa premia la calidad de la clasificación, es decir necesita que la calidad de la clasificación sea lo mejor posible, para detallar el mejor algoritmo necesitaríamos entrenar modelos no lineales como K-NN, y decidir en función de ésto.
- Si la empresa premia el equilibrio de la sencillez del modelo con respecto a la calidad de la clasificación, el modelo de SVM lineal implementado sería muy buena opción.

2. Communities and Crime Data Set

2.1. Comprensión del problema

Tenemos en nuestras manos un resumen de datos de las comunidades y delitos de EE.UU.: los datos combinan datos socioeconómicos del Censo de EE.UU. de 1990, datos de la encuesta LEMAS de EE.UU. de 1990 y datos de delitos de la UCR del FBI de 1995. Concretamente el conjunto de datos recopila información de diferentes comunidades en los Estados Unidos sobre varios factores que pueden influir mucho en algunos delitos comunes, como robos, asesinatos o violaciones. Concretamente:

- state: estado de EE.UU. (por número).
- county: código numérico para condado - no predictivo, y muchos valores faltantes (numérico).
- community: código numérico para la comunidad: no predictivo y muchos valores faltantes (numérico)
- communityname: nombre de comunidad - no predictivo - solo para información (string)
- fold: número de fold para validación cruzada no aleatoria de 10 fold, potencialmente útil para depuración, pruebas emparejadas - no predictivo (numérico)
- population: población por comunidad: (numérico - decimal)
- householdsize: personas promedio por hogar (numérico - decimal)
- racepctblack: porcentaje de la población que es afroamericana (numérico - decimal)
- racePctWhite: porcentaje de población que es caucásica (numérico - decimal)
- racePctAsian: porcentaje de la población que es de ascendencia asiática (numérico - decimal)
- racePctHisp: porcentaje de la población que es de herencia hispana (numérico - decimal)
- agePct12t21: porcentaje de la población que tiene entre 12 y 21 años de edad (numérico - decimal)
- agePct12t29: porcentaje de la población que tiene entre 12 y 29 años (numérico - decimal)
- agePct16t24: porcentaje de la población que tiene entre 16 y 24 años (numérico - decimal)

- agePct65up: porcentaje de la población que tiene 65 años o más (numérico - decimal)
- numbUrban: número de personas que viven en áreas clasificadas como urbanas (numéricas - decimales)
- pctUrban: porcentaje de personas que viven en áreas clasificadas como urbanas (numérico - decimal)
- medIncome: ingreso medio del hogar (numérico - decimal)
- pctWWage: porcentaje de hogares con ingresos salariales en 1989 (numérico - decimal)
- pctWFarmSelf: porcentaje de hogares con ingresos agrícolas o por cuenta propia en 1989 (numérico - decimal)
- pctWInvInc: porcentaje de hogares con ingresos de inversión / renta en 1989 (numérico - decimal)
- pctWSocSec: porcentaje de hogares con ingresos de seguridad social en 1989 (numérico - decimal)
- pctWPubAsst: porcentaje de hogares con ingresos de asistencia pública en 1989 (numérico - decimal)
- pctWRetire: porcentaje de hogares con ingresos de jubilación en 1989 (numérico - decimal)
- medFamInc: ingreso familiar medio (difiere del ingreso familiar para hogares no familiares) (numérico - decimal)
- perCapInc: ingreso per cápita (numérico - decimal)
- whitePerCap: ingreso per cápita para caucásicos (numérico - decimal)
- blackPerCap: ingreso per cápita para los afroamericanos (numérico - decimal)
- indianPerCap: ingreso per cápita para los nativos americanos (numérico - decimal)
- AsianPerCap: ingreso per cápita para personas con herencia asiática (numérico - decimal)
- OtherPerCap: ingreso per cápita para personas con otra procedencia que no sea caucásica, afroamericana, nativa americana o asiática (numérico - decimal)
- HispPerCap: ingreso per cápita para personas con herencia hispana (numérico - decimal)
- NumUnderPov: número de personas bajo el nivel de pobreza (numérico - decimal)

- PctPopUnderPov: porcentaje de personas bajo el nivel de pobreza (numérico - decimal)
- PctLess9thGrade: porcentaje de personas de 25 años o más con educación inferior al noveno grado (numérico - decimal)
- PctNotHSGrad: porcentaje de personas de 25 años y más que no son graduados de secundaria (numérico - decimal)
- PctBSorMore: porcentaje de personas de 25 años o más con una licenciatura o educación superior (numérico - decimal)
- PctUnemployed: porcentaje de personas de 16 años y más, en el mundo laboral y desempleados (numérico - decimal)
- PctEmploy: porcentaje de personas de 16 años y más que tienen empleo (numérico - decimal)
- PctEmplManu: porcentaje de personas de 16 años y más que trabajan en la fabricación (numérico - decimal)
- PctEmplProfServ: porcentaje de personas de 16 años y más que tienen empleos en servicios profesionales (numérico - decimal)
- PctOccupManu: porcentaje de personas de 16 años y más que trabajan en la manufactura (numérico - decimal)
- PctOccupMgmtProf: porcentaje de personas de 16 años y más que tienen empleo en ocupaciones gerenciales o profesionales (numérico - decimal)
- MalePctDivorce: porcentaje de hombres divorciados (numérico - decimal)
- MalePctNevMarr: porcentaje de hombres que nunca se han casado (numérico - decimal)
- FemalePctDiv: porcentaje de mujeres divorciadas (numérico - decimal)
- TotalPctDiv: porcentaje de población divorciada (numérico - decimal)
- PersPerFam: número medio de personas por familia (numérico - decimal)
- PctFam2Par: porcentaje de familias (con hijos) encabezadas por dos padres (numérico - decimal)
- PctKids2Par: porcentaje de niños en viviendas familiares con dos padres (numérico - decimal)
- PctYoungKids2Par: porcentaje de niños de 4 años y menores con padres divorciados (numérico - decimal)
- PctTeen2Par: porcentaje de niños de 12-17 años con padres divorciados (numérico - decimal)

- PctWorkMomYoungKids: porcentaje de madres de niños de 6 años o menos en el mundo laboral (numérico - decimal)
- PctWorkMom: porcentaje de madres de niños menores de 18 años en la el mundo laboral (numérico - decimal)
- NumIlleg: número de hijos nacidos entre no casados (numérico - decimal)
- PctIlleg: porcentaje de niños nacidos entre no casados (numérico - decimal)
- PctImmigRecent: porcentaje de inmigrantes que inmigraron en los últimos 3 años (numérico - decimal)
- PctImmigRec5: porcentaje de inmigrantes que inmigraron en los últimos 5 años (numérico - decimal)
- PctImmigRec8: porcentaje de inmigrantes que inmigraron en los últimos 8 años (numérico - decimal)
- PctImmigRec10: porcentaje de inmigrantes que inmigraron en los últimos 10 años (numérico - decimal)
- PctRecentImmig: porcentaje de la población que ha inmigrado en los últimos 3 años (numérico - decimal)
- PctRecImmig5: porcentaje de la población que ha inmigrado en los últimos 5 años (numérico - decimal)
- PctRecImmig8: porcentaje de la población que ha inmigrado en los últimos 8 años (numérico - decimal)
- PctRecImmig10: porcentaje de la población que ha inmigrado en los últimos 10 años (numérico - decimal)
- PctSpeakEnglOnly: porcentaje de personas que solo hablan inglés (numérico - decimal)
- PctNotSpeakEnglWell: porcentaje de personas que no hablan bien inglés (numérico - decimal)
- PctLargHouseFam: porcentaje de hogares familiares que son grandes (6 o más) (numérico - decimal)
- PctLargHouseOccup: porcentaje de todos los hogares ocupados que son grandes (6 o más personas) (numérico - decimal)
- PersPerOccupHous: personas promedio por hogar (numérico - decimal)
- PersPerOwnOccHous: personas promedio por hogar ocupado por sus propietarios (numérico - decimal)

- PersPerRentOccHous: personas promedio por hogar de alquiler (numérico - decimal)
- PctPersOwnOccup: porcentaje de personas en hogares ocupados por sus propietarios (numérico - decimal)
- PctPersDenseHous: porcentaje de personas en viviendas densas (más de 1 persona por habitación) (numérico - decimal)
- PctHousLess3BR: porcentaje de unidades de vivienda con menos de 3 dormitorios (numérico - decimal)
- MedNumBR: número medio de habitaciones (numérico - decimal)
- HousVacant: número de hogares vacíos (numérico - decimal)
- PctHousOccup: porcentaje de viviendas ocupadas (numérico - decimal)
- PctHousOwnOcc: porcentaje de hogares ocupados por el propietario (numérico - decimal)
- PctVacantBoarded: porcentaje de viviendas vacías que están tapiadas (numérico - decimal)
- PctVacMore6Mos: porcentaje de viviendas vacías que han estado vacías por más de 6 meses (numérico - decimal)
- MedYrHousBuilt: Número medio de viviendas construidas en un año (numérico - decimal)
- PctHousNoPhone: porcentaje de unidades de vivienda ocupadas sin teléfono (en 1990, ¡ésto era raro!) (Numérico - decimal)
- PctWOFullPlumb: porcentaje de viviendas sin instalaciones de fontanería completas (numérico - decimal)
- OwnOccLowQuart: vivienda ocupada por el propietario - valor del cuartil inferior (numérico - decimal)
- OwnOccMedVal: vivienda ocupada por el propietario - valor de la mediana (numérico - decimal)
- OwnOccHiQuart: vivienda ocupada por el propietario - valor del cuartil superior (numérico - decimal)
- RentLowQ: vivienda de alquiler - valor del cuartil inferior (numérico - decimal)
- RentMedian: vivienda de alquiler - valor de la mediana (variable del censo H32B del archivo STF1A) (numérico - decimal)
- RentHighQ: vivienda de alquiler - valor del cuartil superior (numérico - decimal)

- MedRent: Valor medio de la renta bruta (variable del censo H43A del archivo STF3A - incluye las empresas de servicios públicos) (numérico - decimal)
- MedRentPctHousInc: renta bruta media en porcentaje del ingreso familiar (numérico - decimal)
- MedOwnCostPctInc: el costo promedio de los propietarios en porcentaje del ingreso familiar - para los propietarios con una hipoteca (numérico - decimal)
- MedOwnCostPctIncNoMtg: el costo promedio de los propietarios en porcentaje del ingreso familiar - para los propietarios sin hipoteca (numérico - decimal)
- NumInShelters: número de personas en refugios para personas sin hogar (numérico - decimal)
- NumStreet: número de personas sin hogar contadas en la calle (numérico - decimal)
- PctForeignBorn: porcentaje de personas extranjeras (numérico - decimal)
- PctBornSameState: porcentaje de personas locales que viven actualmente (numérico - decimal)
- PctSameHouse85: porcentaje de personas que viven en la misma casa que en 1985 (5 años antes) (numérico - decimal)
- PctSameCity85: porcentaje de personas que viven en la misma ciudad que en 1985 (5 años antes) (numérico - decimal)
- PctSameState85: porcentaje de personas que viven en el mismo estado que en 1985 (5 años antes) (numérico - decimal)
- LemasSwornFT: número de policías a tiempo completo (numérico - decimal)
- LemasSwFTPerPop: policías a tiempo completo por cada 100K de población (numérico - decimal)
- LemasSwFTFieldOps: número de policías a tiempo completo en operaciones de campo (en la calle en lugar de administrativos, etc.) (numérico - decimal)
- LemasSwFTFieldPerPop: oficiales de policía a tiempo completo en operaciones de campo (en la calle en lugar de administrativos, etc.) por cada 100 mil habitantes (numérico - decimal)
- LemasTotalReq: solicitudes totales de policía (numérico - decimal)
- LemasTotReqPerPop: solicitudes totales de policía por población de 100K (numérico - decimal)
- PolicReqPerOffic: solicitudes totales de policía para oficial de policía (numérico - decimal)

- PolicPerPop: oficiales de policía por cada 100K de población (numérico - decimal)
- RacialMatchCommPol: una medida de la coincidencia racial entre la comunidad y la fuerza policial. Los valores altos indican que las proporciones en la comunidad y la fuerza policial son similares (numérico - decimal)
- PctPolicWhite: porcentaje de policías que son caucásicos (numérico - decimal)
- PctPolicBlack: porcentaje de policías que son afroamericanos (numérico - decimal)
- PctPolicHisp: porcentaje de policías que son hispanos (numéricos - decimales)
- PctPolicAsian: porcentaje de policías que son asiáticos (numéricos - decimales)
- PctPolicMinor: porcentaje de policías que son minoría de cualquier tipo (numérico - decimal)
- OfficAssgnDrugUnits: número de oficiales asignados a unidades especiales de drogas (numérico - decimal)
- NumKindsDrugsSeiz: número de diferentes tipos de drogas incautadas (numérico - decimal)
- PolicAveOTWorked: promedio de horas extras policiales trabajadas (numérico - decimal)
- LandArea: área de tierra en millas cuadradas (numérico - decimal)
- PopDens: densidad de población en personas por milla cuadrada (numérico - decimal)
- PctUsePubTrans: porcentaje de personas que usan el transporte público para viajar (numérico - decimal)
- PolicCars: número de coches de policía (numérico - decimal)
- PolicOperBudg: presupuesto operativo de la policía (numérico - decimal)
- LemasPctPolicOnPatr: porcentaje de policías a tiempo completo en patrulla (numérico - decimal)
- LemasGangUnitDeploy: unidad de pandillas desplegada (numérica - decimal - pero realmente ordinal - 0 significa NO, 1 significa SÍ, 0.5 significa Tiempo parcial)
- LemasPctOfficDrugUn: porcentaje de oficiales asignados a unidades de drogas (numérico - decimal)
- PolicBudgPerPop: presupuesto operativo de la policía por población (numérico - decimal)

- ViolentCrimesPerPop: número total de crímenes violentos por cada 100 mil habitantes (numérico - decimal) atributo Objetivo (por predecir)

Así podemos ver que tenemos 128 atributos o características en el dataset y 1994 instancias de éstos.

El objetivo es predecir el atributo *ViolentCrimesPerPop*. Estamos por tanto afrontando un problema de aprendizaje supervisado donde las características, X, son los 127 atributos explicados anteriormente sin el atributo objetivo ($X = [0,1]^{122} x Z$, donde Z se corresponde con los 5 atributos no predictivos del dataset que ya hemos comentado y analizaremos con detalle posteriormente; y las demás, son las 122 características restantes, sin contar el atributo objetivo, que están normalizadas entre 0 y 1), y por tanto, la etiqueta y es el atribujo objetivo 'ViolentCrimesPerPop' que como hemos dicho explica el número total de crímenes violentos por cada 100 mil habitantes y está normalizada entre 0 y 1 ($y = [0,1]$). Así, la función de clasificación f es aquella que hace corresponder las características de cualquier instancia posible, del dataset o no, con su correspondiente etiqueta y.

Veamos una visualización de la variable objetivo por cada éstado:

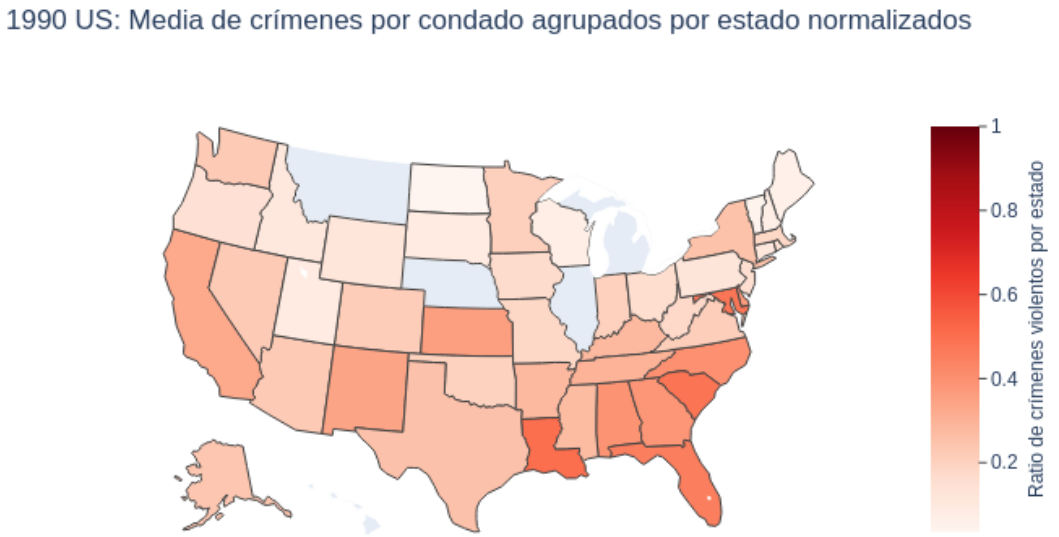


Figura 3: Visualización de la variable *ViolentCrimesPerPop* por estado.

Para obtener dicho gráfico hemos hecho un estudio de la codificación de cada estado y hemos obtenido la media de cada instancia del dataset por condados agrupados por estados, datos que éstan normalizados entre el máximo de dicho valor y el menor (en éste caso 0, ya que hay información nula de algunos éstados). Aclaramos que en nuestro dataset se considera estado el distrito federal de Washington D.C..

Observamos que no tenemos información de 5 éstados.

Ampliando sobre el distrito de Washington D.C., que recordemos que es la capital de EE.UU., observamos que es aquí donde tenemos una mayor incidencia de crímenes violentos por habitante en todo el país.

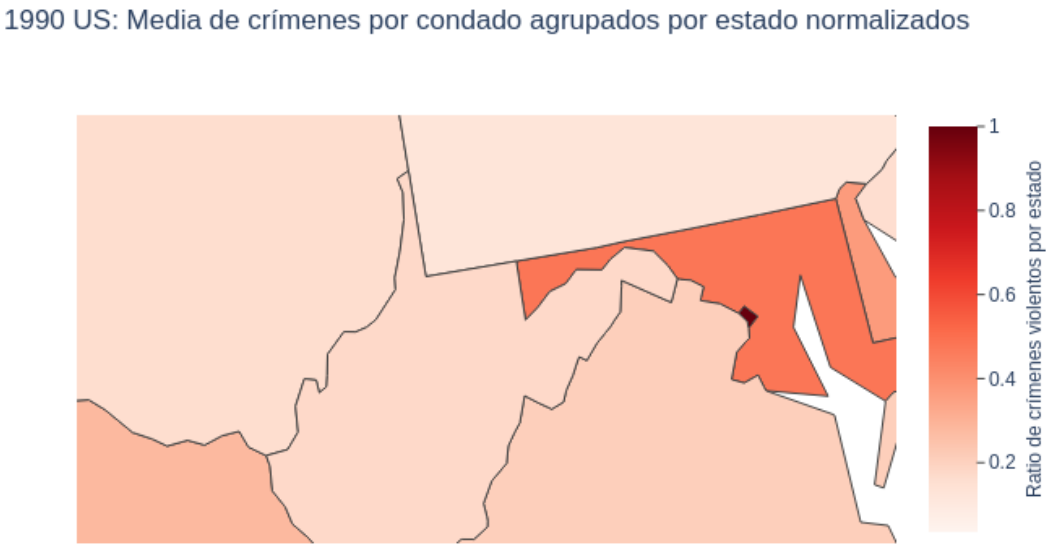


Figura 4: Visualización de la variable *ViolentCrimesPerPop* por estado, ampliado a Washington D.C..

Éste gráfico está obtenido mediante la libreria *plotly.graph_objects* de Python.

Por otro lado, hemos comprobado que los valores perdidos se interpretan como '?' en nuestro dataset, por lo que tras tratar éste tema y cambiar el tipo de las columnas en *Python*, hemos comprobado que tenemos 127 atributos de tipo numérico y 1 de tipo categórico: *communityname*, que es meramente informativa y no aporta información para predicir la variable objetivo.

Por último concluimos y aclaramos que estamos ante un problema de regresión, pues nuestro objetivo es predecir un valor real de nuestra variable 'ViolentCrimesPerPop' para cada instancia que tratemos, es decir, el resultado es un valor numérico, dentro de un conjunto infinito de posibles resultados. Ésto es lo que caracteriza a los problemas de regresión con respecto a los de clasificación.

Información extraída del fichero *communities.names* proporcionado junto al dataset.

2.2. Selección de las clase/s de funciones a usar.

Como hemos visto, tenemos un total de 127 características a analizar (aunque eliminaremos algunas posteriormente, el número seguirá siendo elevado). Creo necesario usar, como en el problema anterior, combinaciones lineales de éstas características como clase de funciones a usar, ya que podríamos sobrecargar el modelo con clases de funciones de grado superior o no lineales. Además intentamos evitar el sobreajuste usando estas combinaciones lineales, aunque podemos pecar de 'underfitting' o subajuste. Todo ésto lo analizaremos cuando tengamos los resultados del modelo. Así pues el modelo será de la forma:

$$\mathcal{H} = w_0 + \sum_{i=1}^{N} w_i x_i \ donde \ w_i \in \mathbb{R}.$$

2.3. Fijamos conjuntos de training y test.

Hemos separado el dataset en dos conjuntos: el conjunto de test (formado por el 25 % de las instancias) y el conjunto de train (formado por el 75 % de las instancias). Así de las 1994 muestras totales que tenemos en el dataset, para el conjunto de entreanamiento hemos escogido 1495, y para el conjunto de test 499. Así pues el porcentaje de datos de entrenamiento con respecto al total es del 74.97492477432297 %, y el porcentaje de datos de test con respecto al total es del 25.025075225677032 %. De esta forma tenemos particiones bastante óptimas siguiendo las recomendaciones dadas. Usaremos la técnica cross-validation para escoger los hiperparámetros del clasificador.

A partir de aquí solo trataremos con los datos del conjunto de train hasta que entrenemos nuestro modelo y usaremos los datos del conjunto de test para estimar el error de dicho modelo.

2.4. Preprocesado de los datos.

Una vez cargados los datos, el primer paso que hemos dado, como hemos comentado, ha sido sustituir los '?' que indicaban valores perdidos, por NaN ("Not a Number") que son valores vacíos no computables para Python. Una vez hecho ésto podemos estudiar de forma cómoda los valores perdidos.

Hemos creado una función denominada $valores\_perdidos(dataframe)$ que hace un recuento de los valores perdidos de cada columna e indica el porcentaje de valores perdidos de la columna con respecto al total, y por último ordena las columnas por número de valores perdidos de forma descendente. El resultado de aplicarlo a nuestro

conjunto de datos es que tenemos 25 columnas o características con valores perdidos. Sin embargo la cantidad de información faltante no es uniforme entre las columnas:

	Total	Porcentaje
LemasSwFTFieldOps	1675	0.84002
PctPolicAsian	1675	0.84002
PolicBudgPerPop	1675	0.84002
LemasSwornFT	1675	0.84002
LemasSwFTPerPop	1675	0.84002
LemasSwFTFieldPerPop	1675	0.84002
LemasTotalReq	1675	0.84002
LemasTotReqPerPop	1675	0.84002
PolicReqPerOffic	1675	0.84002
PolicPerPop	1675	0.84002
PctPolicWhite	1675	0.84002
PctPolicBlack	1675	0.84002
PctPolicHisp	1675	0.84002

(a) Parte 1

	Total	Porcentaje
RacialMatchCommPol	1675	0.840020
PctPolicMinor	1675	0.840020
PolicCars	1675	0.840020
LemasGangUnitDeploy	1675	0.840020
OfficAssgnDrugUnits	1675	0.840020
PolicOperBudg	1675	0.840020
LemasPctPolicOnPatr	1675	0.840020
PolicAveOTWorked	1675	0.840020
NumKindsDrugsSeiz	1675	0.840020
community	1177	0.590271
county	1174	0.588766
OtherPerCap	1	0.000502
AsianPerCap	0	0.000000

(b) Parte 2

Figura 5: Valores perdidos por columnas

Por un lado vemos que tenemos 22 columnas con mas de un 80 % de sus valores vacíos, todos referente a atributos sobre la policía. Ante ésto, lo mejor es eliminar dichas columnas ya que es imposible estimar dichos valores a partir de los valores no faltantes y tener carácterísticas sin apenas valores no aporta información al modelo.

Por otro lado, tenemos mas del 50 % de valores perdidos para los atributos 'community' y 'county' que como hemos dicho anteriormente son atributos informativos de localización de las instancias del dataset, con lo que podemos prescindir de ellas, ya que tenemos el atributo 'state' para referirnos a la localización y el atributo 'communityname' para concretar la localización dentro de cada estado. Así pues, los atributos 'community' y 'county' son redundantes para nuestro objetivo y pueden ser eliminados, al igual que las otras 22 columnas anteriores.

Por último nos falta tratar el único valor perdido para el atributo 'OtherPerCap' que recordemos es el ingreso per cápita para personas con otra procedencia que no sea caucásica, afroamericana, nativa americana o asiática. Hemos considerado rellenar dicho valor con la media de los otros valores de dicho atributo, sin embargo

extraeremos más información del dataset y lo rellenaremos con la media de los valores de dicho atributo del mismo estado al que pertenece, para hacer más concreto y fiable el valor proporcionado a dicha instancia.

De esta forma podemos dar por tratados los valores perdidos. Nos hemos quedado con 103 atributos y hemos mantenido el número de instancias.

También vamos a eliminar los atributos restantes que no nos van a servir para la predicción: 'state', que es un idenficador del número de estado; 'communityname', que es una cadena con el nombre de la población y 'fold' que es un atributo no predictivo para hacer la validación cruzada. Eliminamos éste último puesto que la validación cruzada la haremos sin fijarnos en éste atributo.

Por otro lado, para seguir haciendo una buena selección de variables nos basaremos en la matriz de correlación.

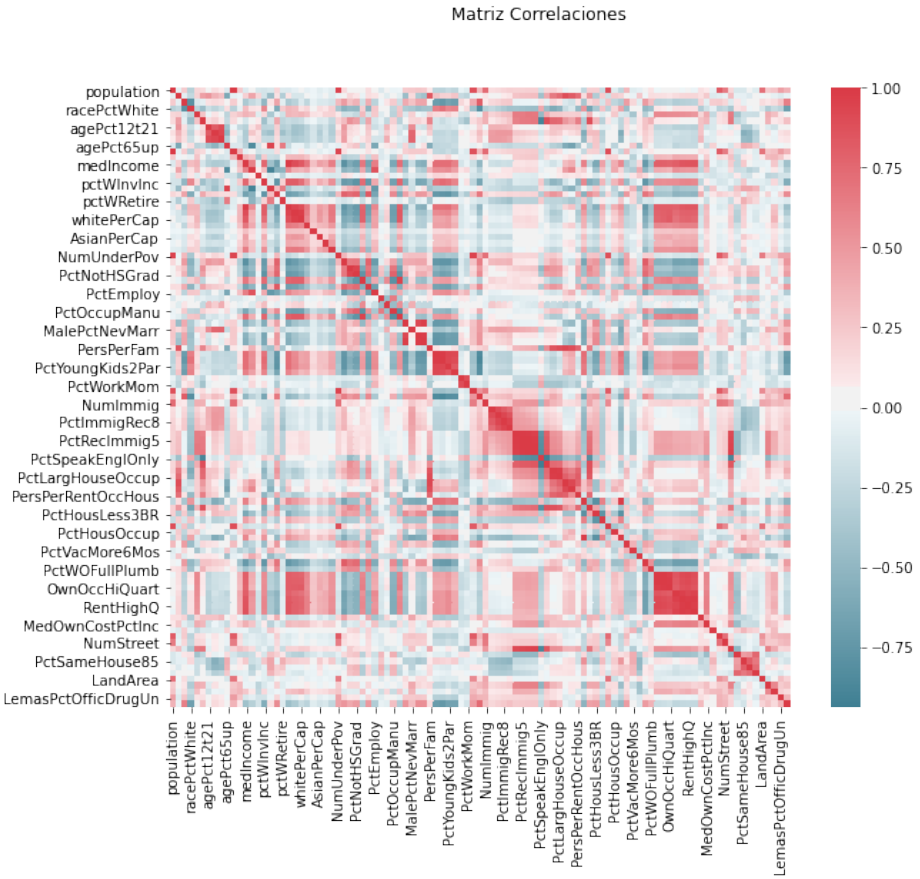


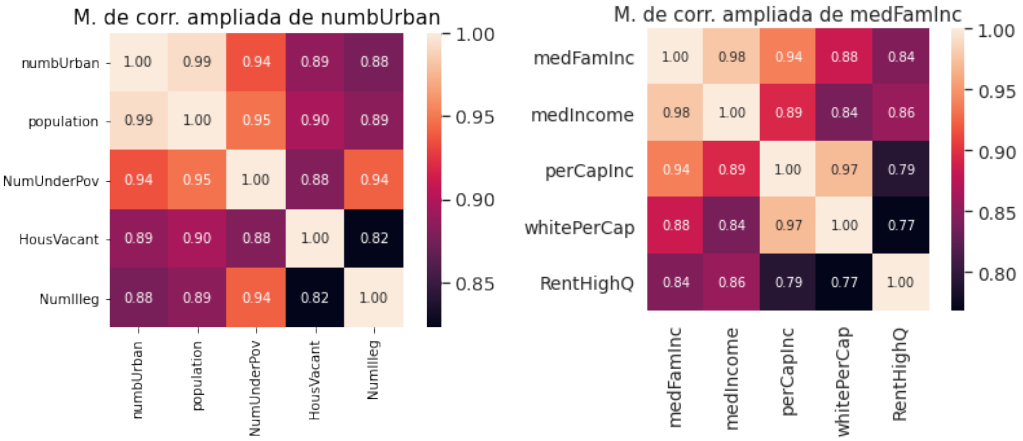
Figura 6: Matriz de correlación

Usaremos el coeficiente de Pearson para calcular dichas correlaciones: si este coeficiente es igual a 1 o -1 (o cercano a estos valores) significa que una variable es fruto de una transformación lineal de la otra. Teniendo una relación directa al tratarse de 1 (cuando una variable aumenta, la otra también), mientras que existirá una relación

inversa al tratarse de -1 (cuando una variable aumenta la otra disminuye). Mientras que, si el coeficiente es igual a cero (o cercano a este valor) no existe relación lineal, aunque puede existir algún otro tipo de relación no lineal. No nos servirá de nada tener variables multicolineales porque básicamente ofrecen la misma información. Vamos a estudiar aquellas variables que tienen un coeficiente de pearson en valor absoluto de más de 0.95 (muy correladas entre sí). En el código de éste apartado hemos incluído una pregunta para indicar si se quiere o no visualizar las siguientes gráficas, ya que puede ser engorroso para el lector.

Por un lado tenemos que *numbUrban* (número de personas que viven en áreas clasificadas como urbanas) y *population* (población por comunidad) tienen una correlación de 0.99 sobre 1, por lo que eliminaremos una de ellas (nos quedaremos con *population*). 7a

De igual forma tenemos que *medFamInc* (ingreso familiar medio) y *medIncome* (ingreso medio del hogar) tienen una correlación de 0.98 sobre 1, por lo que eliminaremos una de ellas (nos quedaremos con *medIncome*). 7b

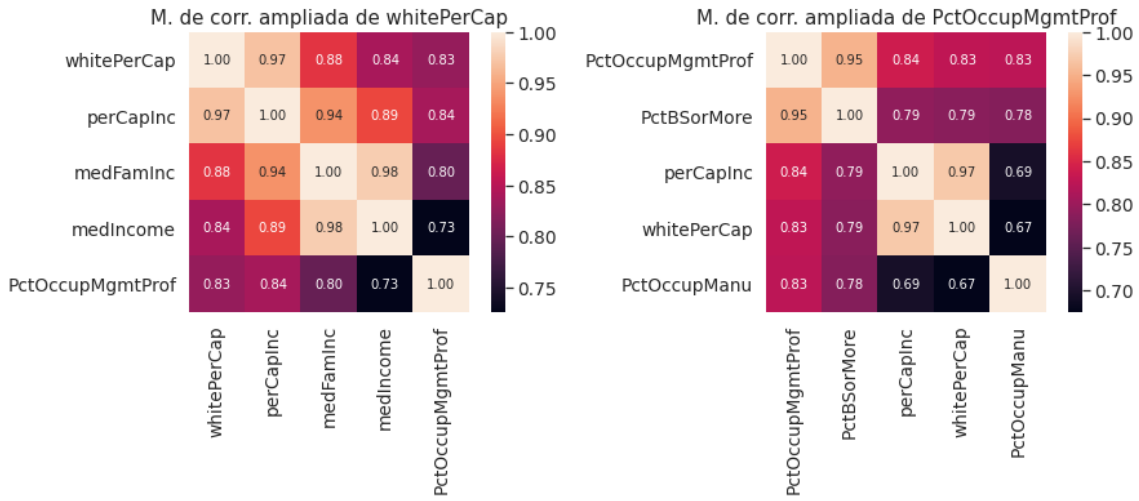


(a) numbUrban (b) medFamInc

Figura 7: Correlaciones ampliadas de cada atributo con similudes de correlación, parte 1

Análogamente tenemos que *whitePerCap* (ingreso per cápita para caucásicos) y *perCapInc* (ingreso per cápita) tienen una correlación de 0.97 sobre 1, por lo que eliminaremos una de ellas (nos quedaremos con *perCapInc*). 8a

De la misma forma tenemos que *PctOccupMgmtProf* (porcentaje de personas de 16 años y más que tienen empleo en ocupaciones gerenciales o profesionales) y *PctBSorMore* (porcentaje de personas de 25 años o más con una licenciatura o educación superior) tienen una correlación de 0.95 sobre 1, por lo que eliminaremos una de ellas (nos quedaremos con *PctBSorMore*). 8b



(a) whitePerCap (b) PctOccupMgmtProf

Figura 8: Correlaciones ampliadas de cada atributo con similudes de correlación, parte 2

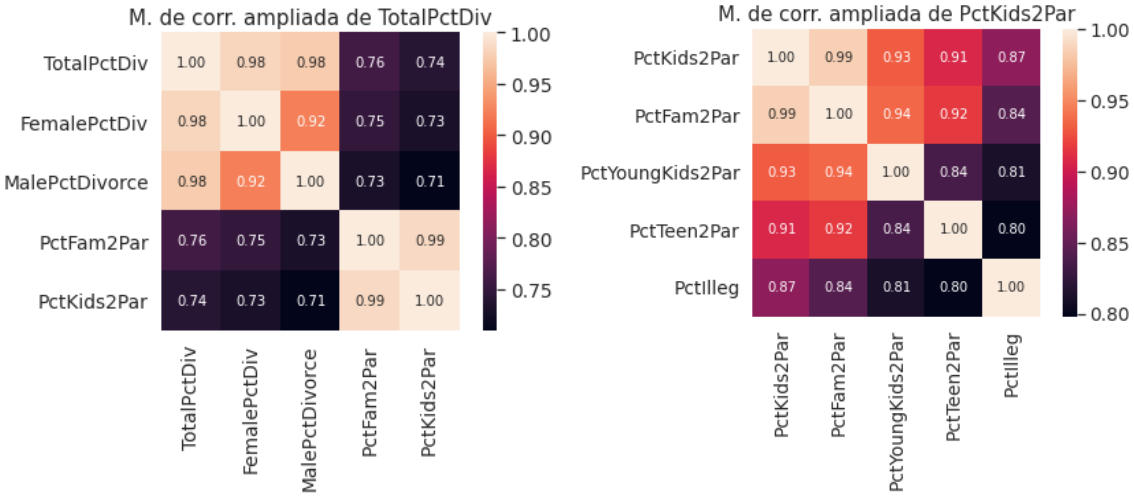
También vemos que tenemos que *TotalPctDiv* (porcentaje de población divorciada) y *FemalePctDiv* (porcentaje de mujeres divorciadas) tienen una correlación de 0.98 sobre 1, pero *TotalPctDiv* también tiene una correlación de 0.98 sobre 1 con *MalePctDivorce* y entre *MalePctDivorce* y *FemalePctDiv* existe una correlación de 0.92 sobre 1 por lo que eliminaremos *TotalPctDiv* ya que se pueden explicar con las otras dos variables. 9a

Se puede ver también que *PctKids2Par* (porcentaje de niños en viviendas familiares con dos padres) y *PctFam2Par* (porcentaje de familias (con hijos) encabezadas por dos padres) tienen una correlación de 0.99 sobre 1 por lo que eliminaremos *PctFam2Par* ya que se puede explicar con la otra variable. 9b

Tratamos ahora un conjunto de variables complicadas: *PctRecImmig8*, *PctRecImmig5*, *PctRecImmig10*, *PctRecentImmig* y *PctForeignBorn* que recordemos que son el porcentaje de personas que inmigraron en los últimos 8 años, en los últimos 5, en los últimos 10, en los últimos 3 y el porcentaje de personas extranjeras en total, respectivamente. Vemos que existe una correlación muy alta entre todos los atributos por lo que hemos decidido quedarnos con *PctRecentImmig* y *PctForeignBorn* que son las que tienen una correlación menor entre sí y explican de forma correcta las otras tres características. 10a

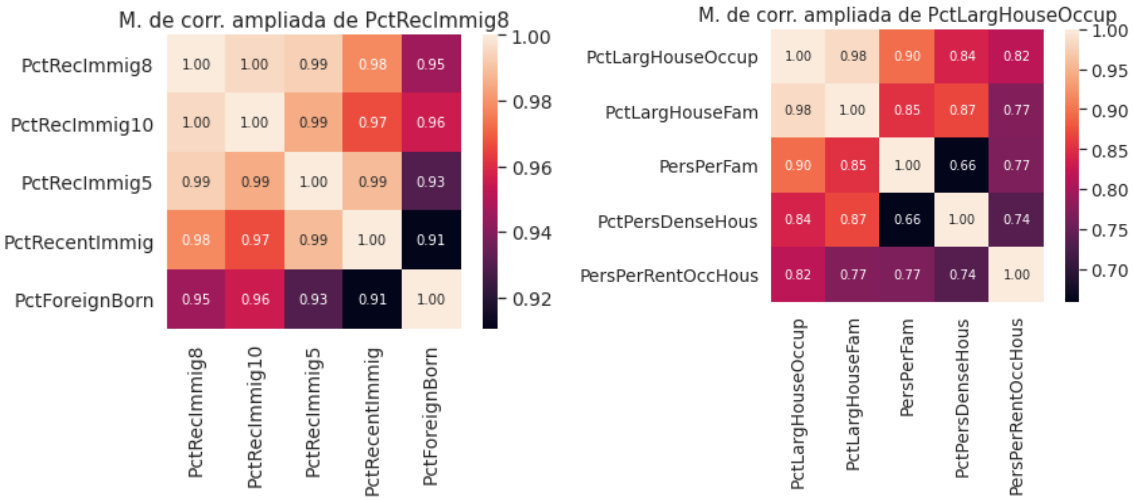
Por otro lado tenemos las variables *PctLargHouseOccup* y *PctLargHouseFam* (porcentaje de todos los hogares ocupados que son grandes (6 o más personas) y el porcentaje de hogares familiares que son grandes (6 o más)) con una correlación de 0.98 sobre 1, por lo que nos quedaremos solo con *PctLargHouseOccup*. 10b

Nos encontramos ahora con las variables *PctHousOwnOcc* y *PctPersOwnOccup*



(a) TotalPctDiv (b) PctKids2Par

Figura 9: Correlaciones ampliadas de cada atributo con similudes de correlación, parte 3



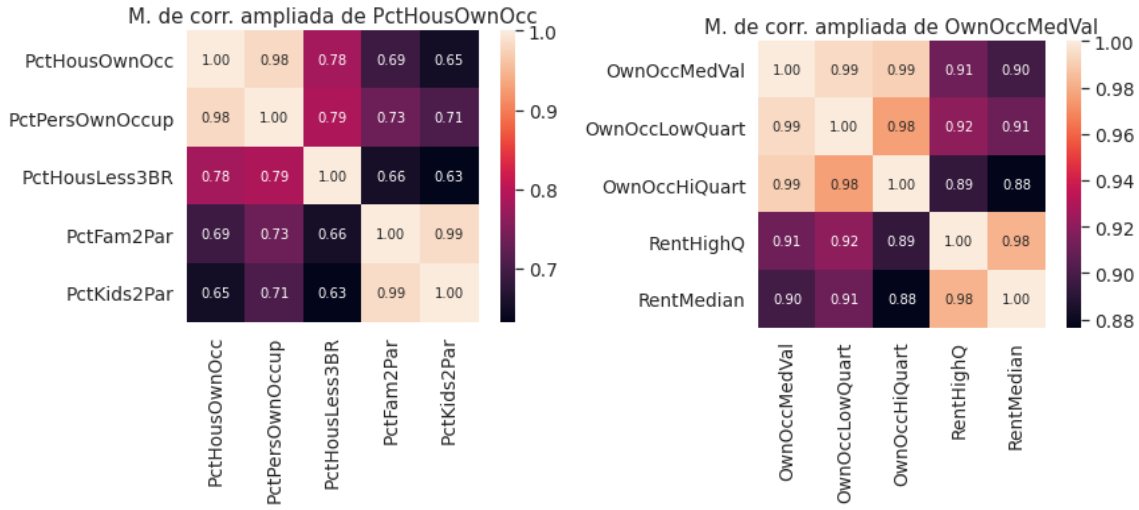
(a) PctRecImmigX (b) PctLargHouseOccup

Figura 10: Correlaciones ampliadas de cada atributo con similudes de correlación, parte 4

(el porcentaje de hogares ocupados por el propietario y el porcentaje de personas en hogares ocupados por sus propietarios, respectivamente) con una correlación de 0.98 sobre 1, por lo que nos quedaremos solo con *PctHousOwnOcc*. 11a

Por otro lado tenemos las características *OwnOccMedVal*, *OwnOccLowQuart* y *OwnOccHiQuart* (viviendas ocupada por el propietario - valor de la mediana, viviendas ocupada por el propietario - valor del cuartil inferior y viviendas ocupada por el propietario - valor del cuartil superior, respectivamente), con una correlación

muy elevada entre cada una de las tres varibales, con lo que nos quedamos con el valor mediano (*OwnOccMedVal*). 11b



(a) PctHousOwnOcc (b) OwnOccMedVal

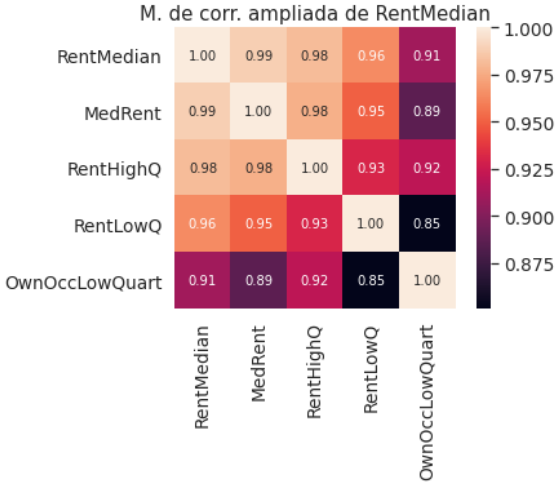
Figura 11: Correlaciones ampliadas de cada atributo con similudes de correlación, parte 5

Por último nos encontramos con el grupo de variables formado por *RentMedian*, *MedRent*, *RentHighQ* y *RentLowQ* (vivienda de alquiler - valor de la mediana, valor medio de la renta bruta - incluye las empresas de servicios públicos), vivienda de alquiler - valor del cuartil superior y vivienda de alquiler - valor del cuartil inferior). Nos quedamos con *MedRent* y con *RentLowQ* ya que son las características que tienen una correlación menor entre sí y explican de forma correcta los otros dos atributos. 12a

Así hemos completado una reducción de la dimensionalidad de forma supervisada (al contrario que hicimos con el problema de clasificación, en el que usamos un método no supervisado (PCA)). Tenemos así 86 atributos que forman nuestro dataset, de los cuales 1 es el atributo objetivo.

Por otro lado, hemos comprobado que la mayoria de los atributos están escalados entre 0 y 1, salvo algunos que no llegan a éstar correctamente escalados, como hemos comprobado. Éstas las hemos escalado entre 0 y 1 mediante la función $MinMaxScaler$ de Sklearn.

De ésta forma, tenemos los datos debidamente escalados, sin valores perdidos, todos los atributos son numéricos y estamos preparados para el entrenamiento del modelo.



(a) RentMedian

Figura 12: Correlaciones ampliadas de cada atributo con similudes de correlación, parte 6

2.5. Métrica de error a usar.

Dado que tenemos entre manos un problema de regresión, vamos a usar el coeficiente de determinación, o R^2, como métrica de error.

$$R^2 = 1 - \frac{\frac{1}{N}\sum_{i=1}^{N}(y_i - \hat{y}_i)^2}{\frac{1}{N}\sum_{i=1}^{N}(y_i - \bar{y}_i)^2}$$

donde y_i el resultado real esperado, $\hat{y}_i$ la predicción del modelo y $\bar{y}_i$ es la media de las y_i [5]

.

El coeficiente de determinación R^2 está estrechamente relacionada con la métrica MSE (error cuadrático medio), pero tiene la ventaja de estar libre de escala, es decir, no importa si los valores de salida son muy grandes o muy pequeños, R^2 siempre estará entre $-\infty$ y 1.

Recordamos que el error cuadrático medio (MSE) calcula, básicamente, para cada punto, la diferencia al cuadrado entre las predicciones y el objetivo y luego promedia esos valores. A mayor valor de ésta métrica, peor es el modelo. Se calcula tal que:

$$MSE = \frac{1}{N}\sum_{i=1}^{N}(y_i - \hat{y}_i)^2$$

con y_i el resultado real esperado y $\hat{y}_i$ la predicción del modelo. [5]

Debemos conocer de R^2 que el mejor valor posible es 1 (el modelo predeciría siempre el valor correcto para los datos proporcionados), pero éste valor puede ser negativo si el modelo es peor que un modelo constante que siempre predice el va-

lor esperado de la variable objetivo sin tener en cuenta las características de entrada.

Cuando obtengamos los resultados del entrenamiento y la predicción analizaremos este tema con más detalle.

2.6. Discusión de la técnica de ajuste elegida.

De nuevo anticipándonos a los acontecimientos, vamos a probar dos algoritmos lineales para entrenar nuestro problema y según los resultados elegir uno u otro: dos tipos de regresión lineal (una con regularización Ridge y que utiliza una técnica de optimización llamada gradiente conjugado; y otra, también con regularización Ridge, pero que utiliza el descenso de gradiente estocástico en lugar del gradiente conjugado porque, a menudo, el descenso de gradiente estocástico es más eficiente para datos a gran escala, de alta dimensión y dispersos); además, por otro lado, como ampliación a la práctica, los compararemos con un lineal support-vector machine (SVM lineal). Ésto se explicará con detalle en la sección 2.8. Así pues, usaremos las clases que nos proporciona la librería de Sklearn para los algoritmos comentados anteriormente. A continuación escogeremos la técnica de ajuste para uno y otro modelo, ya que el modelo influye en la técnica de ajuste elegida:

- *Ridge*: Como hemos indicado, la técnica de ajuste que utiliza es método conocido como **gradiente conjugado**. Viene motivado por el deseo de acelerar la convergencia, obtenida con el Descenso del Gradiente, y a la vez evitar los requisitos de computación asociados a la evaluación, almacenamiento e inversión de la matriz Hessiana, como requiere el método de Newton. En este algoritmo de entrenamiento la búsqueda se realiza a lo largo de direcciones conjugadas, lo que normalmente produce una convergencia más rápida que las direcciones obtenidas con el Descenso del Gradiente. Este método ha demostrado ser más eficaz en muchos problemas que el de Descenso del Gradiente, y no requiere el cálculo de la matriz Hessiana. Lo probaremos en nuestro problema. Lo indicaremos en Python mediante el parámetro $solver = `sparse\_cg'$ [4].

- *SGDRegressor*: como su propio nombre indica, la técnica de ajuste que utiliza es el **SGD**, que como sabemos en el descenso de gradiente estocástico el gradiente de la pérdida se estima en cada muestra a la vez y el modelo se actualiza a lo largo del camino con una tasa de aprendizaje. Es el método de minimización por defecto en el modelo [4].

- *LinearSVR*: la técnica de ajuste que utiliza es método conocido como **descenso coordinado**. Tal y como explicamos en la sección 1.6, dicho método es un algoritmo de optimización que minimiza sucesivamente a lo largo de las direcciones de coordenadas para encontrar el mínimo de una función. En cada iteración, el algoritmo determina una coordenada o un bloque de coordenadas a través de una regla de selección de coordenadas, luego minimiza de manera

exacta o inexacta sobre el hiperplano de coordenadas correspondiente mientras fija todas las demás coordenadas o bloques de coordenadas. Se puede realizar una búsqueda de línea a lo largo de la dirección de coordenadas en la iteración actual para determinar el tamaño de paso apropiado. El descenso coordinado es aplicable tanto en contextos diferenciables como libres de derivadas,lo cual es perfecto para nuestro problema. Es el método de minimización por defecto en el modelo. [3]

2.7. Discusión de la necesidad de regularización.

Esta sección comparte toda la información con el apartado 1.7 explicado anteriormente y no vamos a repetirla en ésta sección.

Adaptándola a nuestro problema: como nuestro dataset tiene una alta dimensionalidad, usaremos la regularización para intentar evitar el sobreajuste.

Como hemos hecho una buena selección de atributos, consideramos que todos los atributos seleccionados son relevantes para nuestro problema, además intentamos evitar el efecto de la correlación entre los atributos de entrada. Por todo ésto creemos que el mejor tipo de regularización es la L2 o Ridge.

2.8. Identificación de los modelos a usar.

Como anteriormente hemos comentado, vamos a usar dos tipos de algoritmos para entrenar los datos y poder predecir los resultados: dos tipos de regresión lineal.

Recordamos que ya hemos discutido en la sección anterior (2.7) la necesidad de regularización y el tipo, entre todos las posibles, que usaremos:

Para la primera regresión lineal (que utiliza una técnica de optimización llamada gradiente conjugado), usaremos la clase de Sklearn *Rigde* (clase que usa solamente regularización L2) con el parámetro $solver = `sparse\_cg'$. [4]

Para la segunda regresión lineal (que utiliza el descenso de gradiente estocástico), usaremos la clase de Sklearn $SGDRegressor$. La función de pérdida que usaremos será la función de pérdida cuadrática (especificada en Sklearn con $loss = `squared\_loss'$) que resulta especialmente óptima para que la técnica de ajuste $R^2$ la minimice de forma correcta. Además le especificamos que aplique regularización L2 con $penalty = `l2'$. [2]

Para el SVM lineal, que aplicaremos solo como método comparativo en la práctica, usaremos la clase de Sklearn $LinearSVR$. Usaremos la forma primal (que

indicaremos '$dual = False$') debido a que el número de muestras es mayor que el número de atributos, tal y como indica la documentación de Sklearn. Y usaremos la función de pérdida $\epsilon - insensitive$ al cuadrado (que especificamos para Sklearn con $loss =$ '$squared\_epsilon\_insensitive$'), que sabemos que es la función de pérdida asociada a la regularización L2 para los SVMs de regresión, discutida en la sección 2.7. [3]

A continuación realizaremos una estimación de los hiperparámetros y seleccionaremos el mejor modelo resultante definiendo todas sus características.

2.9. Estimación de hiperparámetros y selección del mejor modelo.

Vamos a usar la validación cruzada para ajustar los hiperparámetros de los modelos. Sabemos que, al aplicar la validación cruzada, el conjunto de datos de entrenamiento se divide en grupos de igual tamaño. Una vez realizada la partición se procede a entrenar el modelo una vez por cada uno de los grupos. Utilizando todos los grupos menos el de la iteración para entrenar y este para validar los resultados.

Como hemos dicho usaremos la clase *Rigde* de la librería Sklearn [4] para una regresión lineal. La inicialización de los parámetros de dicha clase serán:

- *solver* = '*sparse_cg*': Algoritmo a utilizar en el problema de optimización. Explicado en la sección 2.6.
- *tol = 1e-7*: Indica la precisión de la solución. Es un hiperparámetro, pero sabemos que cuanto más pequeño sea dicho valor (siempre positivo), mejores resultados obtendremos. Sin embargo, es una sucesión convergente y las mejoras a partir de dicho valor (10^{-7}) son insignificantes, por eso lo mantenemos fijado a ese valor.
- Los demás valores se han dejado por defecto: el número máximo de iteraciones (*max_iter*) es determinado internamente por el algoritmo, ya hemos aplicado una normalización de los datos en el preprocesado, por lo que *normalize = False*, y usaremos validación cruzada para calcular el coeficiente de regularización (esta vez llamado *alpha*, donde $\alpha = \frac{1}{2C}$). Se pueden consultar los demás parámetros en la salida del código, ya que hemos usado la función *get_params* del clasificador y permitimos visualizar su salida.

Para la otra regresión lienal usaremos la clase *SGDRegressor* de la librería Sklearn [4]. La inicialización de los parámetros de dicha clase serán:

- *loss* = '*squared_loss*': Función de pérdida usada. Explicada en la sección 2.8.
- *penalty='l2'*: Regularización Ridge (L2). Explicada en la sección 2.7.

- *max_iter* = 2000: Número máximo de épocas. Hemos elevado dicho valor $x2$ con respecto al valor por defecto para permitir mayor exploración de los datos, ya que el algoritmo no es cumputacionalmente especialmente lento.

- *tol = 1e-7*: Indica la precisión de la solución. Es un hiperparámetro, pero sabemos que cuanto más pequeño sea dicho valor (siempre positivo), mejores resultados obtendremos. Sin embargo, es una sucesión convergente y las mejoras a partir de dicho valor (10^{-7}) son insignificantes, por eso lo mantenemos fijado a ese valor.

- *learning_rate* = '*invscaling*' Indica que la actualización de la tasa de aprendizaje (*eta*) se realiza como:

  $$eta = eta0/pow(t, power\_t)$$

  donde *eta*0 y *power_t* son hiperparámetros del modelo.

- Los demás valores se han dejado por defecto: usaremos validación cruzada para calcular el coeficiente de regularización (*alpha*, como en la clase anterior), *eta*0 y *power_t*. Se pueden consultar los demás parámetros en la salida del código, ya que hemos usado la función *get_params* del clasificador y permitimos visualizar su salida.

Por último y como ya hemos comentado, usaremos la clase $LinearSVR$ de la librería Sklearn [4], pero solo como ampliación a la práctica, puesto que no se pedía que usaramos SVMs. La inicialización de los parámetros de dicha clase serán:

- *loss = 'squared_epsilon_insensitive'*: Función de pérdida $\epsilon - insensitive$ al cuadrado. Explicado en la sección 2.8.

- *dual = False*: Indicamos que usamos forma primal. Explicado en la sección 2.8.

- *tol = 1e-7*: Indica la precisión de la solución. Es un hiperparámetro, pero sabemos que cuanto más pequeño sea dicho valor (siempre positivo), mejores resultados obtendremos. Sin embargo, es una sucesión convergente y las mejoras a partir de dicho valor (10^{-7}) son insignificantes, por eso lo mantenemos fijado a ese valor.

- Los demás valores se han dejado por defecto: usaremos validación cruzada para calcular el coeficiente de regularización C (explicado en la sección 1.7) y *epsilon*, que es el hiperparámetros de la función de pérdida $\epsilon - insensitive$ al cuadrado (explicado en la sección 2.8). Se pueden consultar en la salida del código, ya que hemos usado la función *get_params* del clasificador y permitimos visualizar su salida.

Para la validación cruzada, dividiremos en 5 partes el conjunto de entrenamiento. Ésto es para reservar aproximadamente el 15 % del total de las instancias para validar, un valor apropiado para el conjunto de validación teniendo en cuenta el

tamaño del conjunto de entrenamiento.

De entre los dos regresores posibles en la práctica (recordamos que el SVR sólo lo hemos incluído para discutirlo en la memoria posteriormente), el mejor clasificador, para los datos de entrenamiento es la regresión lineal con regularización Rigde y que utiliza la técnica de ajuste del gradiente conjugado, es decir, el la instancia de la clase Rigde que hemos definido, con el hiperparámetro $alpha = 10.0$. Hemos obtenido con él un valor de $R^2 = 0.7029625929614456$ para los datos de entrenamiento después de la validación cruzada entre el conjunto de entrenamiento y el conjunto de validación.

2.10. Estimación por validación cruzada del error del modelo.

Una vez realizada la validación cruzada entrenando los distintos modelos con cada hiperparámetro, hemos obtenido distintos errores de validación y hemos escogido el mejor de acuerdo a nuestra métrica para poder así elegir el modelo resultante (todo ésto lo hemos hecho mediante la función $GridSearchCV$ de Sklearn). Así entrenado, con los datos de entrenamiento, éste modelo elegido, vamos a predecir los resultados de los datos de test y a compararlos con su correspondiente salida.

Podemos medir el error del clasificador en los datos de entrenamiento ($E_{in} = 1 - R^2_{train}$) (aunque ésto no es pedido en la práctica) y estimamos el error del clasificador en los datos de test (E_{test}) que nos servirá para estimar E_{out}, obteniendo:

$$E_{in} : 0.2970374070385544$$
$$E_{val} : 0.3370830014499828$$
$$E_{test} : 0.36873809028676086$$

Así, por la desigualdad de Hoeffding, tenemos con probabilidad $1 - \delta$ (transparencias de teoria, diapositiva 22 Tema 3), una cota del error:

$$E_{out} \leq E_{test} + \sqrt{\frac{1}{2N_{test}} \log \frac{2|H|}{\delta}}$$

Aplicando ésta fórmula, sabiendo que $|H| = 1$, ya que hemos estimado la función anteriormente y escogido una concreta, tenemos que con probabilidad de un 95 %:

$$E_{out} : 0.42953506388010804$$

El resultado no es demasiado bueno ya que recordamos que buscamos un valor cerano a 0. La complejidad del dataset (con demasiadas intancias), puede ser fruto de éste resultado. Además los datos pueden que no se adapten de forma correcta a nuestra clase de funciones, por lo que como mejora, podríamos intentar aumentar la complejidad de éstas.

2.11. Modelo propuesto para la empresa

Hasta ahora hemos estudiado dos clase de modelos para este problema. Podríamos haber cogido varias clases (ahora es donde entra en juego el SVM lineal, por ejemplo). Si hubiesemos realizado una validación cruzada con la misma separación de los datos para los conjuntos de validación y la hubiesemos aplicado a los dos modelos, hubiesemos obtenido el mejor modelo resultante. Sin embargo al haber entrenado ya los datos con nuestro modelo de regresión logística ya no podemos establecer ésta comparación en igualdad de condiciones.

Si entrenamos el SVM lineal haciendo validación cruzada sobre los datos de entrenamiento para obtener los hiperparámetros obtenemos un error de validación de 0.33690061282617145, es decir, muy cercano al que obtuvimos con al aplicar la validación cruzada sobre las dos regresiones lineales estudiadas. Pero como hemos dicho, éste dato no permite compararlos y decir cual es el mejor modelo, porque no se han entrenado en igualdad de condiciones.

Sin embargo si podemos decir que tanto el mejor modelo de regresión lineal obtenido como el modelo de SVM lineal se adaptan de forma muy parecida a los datos de nuestro problema a pesar que la calidad de los dos modelos no es bastante buena por lo explicado en la sección anterior.

Así no podemos usar un modelo de los anteriormente propuestos para la empresa: habría que estudiar si ampliando la complejidad de los modelos propuesto y de la clase de funciones a usar, las predicciones mejoran.

Referencias

[1] Información sobre Crammer-Singer:
http://jmlr.csail.mit.edu/papers/volume2/crammer01a/crammer01a.pdf

[2] Información usada para la elección de la función de pérdida Squared-loss:
https://heartbeat.fritz.ai/5-regression-loss-functions-all-machine-learners-should-know-4fb140e9d4b0

[3] Información matemática de LinearSVR sobre la función de pérdida y la técnica de ajuste:
https://www.csie.ntu.edu.tw/ cjlin/papers/linear-svr.pdf

[4] Documentación de Sklearn.
https://scikit-learn.org/stable/_downloads/scikit-learn-docs.pdf

[5] Métricas de regresión.
https://sitiobigdata.com/2018/08/27/machine-learning-metricas-regresion-mse/

[6] PCA - PrincipalComponent Analysis
https://www.cs.cmu.edu/ elaw/papers/pca.pdf

[7] L-BFGS
http://www.jmlr.org/papers/volume16/mokhtari15a/mokhtari15a.pdf

[8] Función de pérdida Hinge
http://www.jmlr.org/papers/volume9/bartlett08a/bartlett08a.pdf

[9] Algoritmo del descenso coordinado
http://www.optimization-online.org/DB_FILE/2014/12/4679.pdf

[10] Regularización
https://iartificial.net/regularizacion-lasso-l1-ridge-l2-y-elasticnet/

[11] Softmax
http://users.isr.ist.utl.pt/ wurmd/Livros/school/Bishop%20-%20Pattern%20Recognition%20And%20Machine%20Learning%20-%20Springer%20%202006.pdf