

# Doble Grado en Ingeniería Informática y Matemáticas

APRENDIZAJE AUTOMÁTICO  
(E. Computación y Sistemas Inteligentes)

## **TRABAJO-3: Programación AJUSTE DE MODELOS LINEALES**



**UNIVERSIDAD  
DE GRANADA**

Carlos Santiago Sánchez Muñoz

Grupo de prácticas 3 - Lunes

*Email:* carlossamu7@correo.ugr.es

*29 de mayo de 2020*

# Índice

<b>1. Clasificación</b>	<b>2</b>
1.1. Comprensión del problema a resolver . . . . .	2
1.2. Selección de las clase/s de funciones a usar . . . . .	2
1.3. Fijando los conjuntos de training y test . . . . .	3
1.4. Preprocesado de los datos . . . . .	4
1.5. Fijando la métrica de error a usar . . . . .	5
1.6. Discusión de la técnica de ajuste elegida . . . . .	5
1.7. Discusión de la necesidad de regularización . . . . .	6
1.8. Identificación de los modelos a usar . . . . .	6
1.9. Estimación de hiperparámetros y selección del mejor modelo . . . . .	7
1.10. Estimación por validación cruzada de $E_{out}$ y comparación con $E_{test}$ . . . . .	8
1.11. Modelo a proponer a la empresa . . . . .	9
<b>2. Regresión</b>	<b>10</b>
2.1. Comprensión del problema a resolver . . . . .	10
2.2. Selección de las clase/s de funciones a usar . . . . .	11
2.3. Fijando los conjuntos de training y test . . . . .	12
2.4. Preprocesado de los datos . . . . .	13
2.5. Fijando la métrica de error a usar . . . . .	15
2.6. Discusión de la técnica de ajuste elegida . . . . .	16
2.7. Discusión de la necesidad de regularización . . . . .	16
2.8. Identificación de los modelos a usar . . . . .	16
2.9. Estimación de hiperparámetros y selección del mejor modelo . . . . .	16
2.10. Estimación por validación cruzada de $E_{out}$ y comparación con $E_{test}$ . . . . .	18
2.11. Modelo a proponer a la empresa . . . . .	18

# 1. Clasificación

El primer problema a abordar es un problema de clasificación. La base de datos es a usar es *Optical Recognition of Handwritten Digits* que contiene imágenes de dígitos del sistema de numeración arábigo. El objetivo consiste en aprender de esta base de datos para poder clasificar otras imágenes con dígitos manuscritos.

## 1.1. Comprensión del problema a resolver

El conjunto de datos lo obtenemos en dos ficheros sacados de la dirección proporcionada en el guión (<https://archive.ics.uci.edu/ml/datasets/optical+recognition+of+handwritten+digits>):

- `optdigits.tra` para el conjunto de entrenamiento.
- `optdigits.tes` para para el conjunto de test.

Ambos conjuntos de datos son una batería de imágenes de dígitos manuscritos con un preprocesamiento previo. Estos ficheros tienen un formato CSV (*Comma-Separated Values*) lo cual significa que las filas están separadas por saltos de línea y las columnas por el símbolo ‘,’ (coma).

Vamos a explicar la técnica de reducción de la dimensionalidad que se ha aplicado a cada dígito para obtener su vector de características. Los dígitos son un *bitmap* de tamaño  $32 \times 32$  en donde cada píxel es blanco o negro. Se han dividido en bloques de tamaño  $4 \times 4$  no superpuestos. El vector de características son 64 valores del intervalo  $[0, 16]$  que indican el número de píxeles negros de cada uno de los  $8 \times 8$  bloques de tamaño  $4 \times 4$ . La etiqueta de cada dato es el dígito al que se corresponde, un entero en  $[0, 9]$ .

El problema a resolver es un problema de aprendizaje supervisado. La matriz  $\mathcal{X}$  es la matriz de características que en cada fila tendrá los 64 valores ya explicados y tantas filas como instancias. El vector de etiquetas  $\mathcal{Y}$  está compuesto por enteros en  $[0, 9]$  y su dimensión es el número de instancias. La función de objetivo es  $f : \mathcal{X} \rightarrow \mathcal{Y}$  que para cada instancia  $(x_n, y_n) \in \mathcal{X} \times \mathcal{Y}$  verifica  $f(x_n) = y_n$ .

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

En la búsqueda de la función objetivo  $f$  es necesario fijar la clase de funciones o conjunto de hipótesis  $\mathcal{H}$ . Más adelante elegiremos  $g \in \mathcal{H}$  de modo que  $g \approx f$ .

La clase de funciones elegida es la de combinaciones lineales de los datos. Con las dimensiones que se manejan en este problema usar una clase no lineal puede ser problemático y dar lugar a *overfitting*. Además carecería de sentido pues qué significa elevar al cuadrado el número de unos que hay en un bloque  $4 \times 4$ . A priori no parece que eso funcionase así que nuestra clase elegida es:

$$\mathcal{H}_{clas} = \left\{ w_0 + \sum_{i=1}^N w_i x_i, \quad w \in \mathbb{R}^{N+1} \right\}.$$

### 1.3. Fijando los conjuntos de training y test

En este caso el conjunto de datos proporcionado venía separado distinguiendo el conjunto de entrenamiento del conjunto de test. Vamos a limitarnos a estudiar y averiguar algunos datos de este reparto para decir si es lo que deseamos.

El número de instancias del conjunto de entrenamiento es 3823 y el de test es 1797. Están repartidos por tanto con un porcentaje 68,025 % y 31,975 % respectivamente. Estos datos son muy razonables, falta saber si el porcentaje de aparición de cada dígito está bien repartido en cada conjunto de entrenamiento. En la Tabla 1.

Tabla 1: Reparto de los dígitos en 'train' y 'test'

Díg.	Instancias 'train'	Porcentaje 'train' (%)	Instancias 'test'	Porcentaje 'test' (%)
0	376	9.84	178	9.91
1	389	10.18	182	10.13
2	380	9.94	177	9.85
3	389	10.18	183	10.18
4	387	10.12	181	10.07
5	376	9.84	182	10.13
6	377	9.86	181	10.07
7	387	10.12	179	9.96
8	380	9.94	174	9.68
9	382	9.99	180	10.02

En unas gráficas de barras lo podemos apreciar mejor:

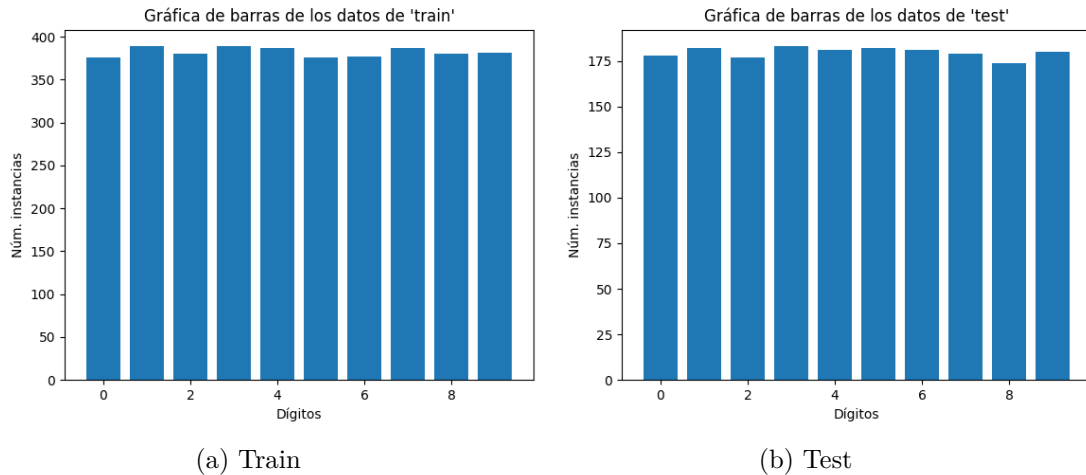


Imagen 1: Gráficas de barras de los datos

En efecto, las apariciones está bien estratificadas y dicha uniformidad es necesaria para el buen aprendizaje del algoritmo que vamos a construir.

## 1.4. Preprocesado de los datos

Antes de hacer el preprocesado vamos a verificar algunas cosas que debe de cumplir la base de datos después de la comprensión y estudio realizado. En primer lugar se ha comprobado la no existencia de valores perdidos u *outliers*. En un base de datos tan conocida era obvio que esto no iba a pasar. A continuación se ha comprobado que todos los valores sean enteros. Por último, todos los valores de las características deben de estar en el rango  $[0, 16]$  y los de las etiquetas en  $[0, 9]$ . Veamos estas comprobaciones:

```
Outliers en 'train': 0
Outliers en 'test': 0
Todos los valores son enteros en 'train': True
Todos los valores son enteros en 'test': True
Intervalo en el que están las características de 'train': [0,16]
Intervalo en el que están las etiquetas de 'train': [0,9]
Intervalo en el que están las características de 'test': [0,16]
Intervalo en el que están las etiquetas de 'test': [0,9]
```

Imagen 2: Comprobaciones sobre los datos

El preprocesamiento de los datos es muy importante para el buen ajuste del modelo. Es sin duda un paso fundamental ya que previene del sobreajuste a los datos y de la redundancia a la vez que reduce la dimensionalidad del problema haciéndolo menos complejo y más eficiente en tiempo. Para ello después de diferentes pruebas se ha procedido a tres pasos:

- Eliminar aquellas características cuyo valor sea constante. Para ello simplemente se eliminan aquellas que tienen varianza cero lo cual es equivalente. En este paso se usa `VarianceThreshold` de `sklearn.feature_selection`.
- Análisis de Componentes Principales PCA (Principal Component Analysis). Esta técnica reduce la dimensionalidad del problema sin perder mucha capacidad para explicar la varianza de los datos. El espacio de trabajo se transforma mediante esta técnica a uno más pequeño conservando un porcentaje de varianza explicada elegido, en nuestro caso un 95 %. Es probable que algunas zonas de la imagen no sean tan importantes y con esta técnica las podamos descartar. Para este paso usamos `StandardScaler` de `sklearn.preprocessing`.
- Para poder aplicar PCA los datos deben de tener una distribución gaussiana. Por consiguiente se van a normalizar para que la media sea 0 y la varianza 1. Para este paso se importa PCA de `sklearn.decomposition`.

Estos tres pasos los unimos con Pipeline disponible en `sklearn.pipeline`. Básicamente dados unos datos en cada paso se hace el `fit_transform` a ellos y se pasa al siguiente elemento del Pipeline.

```
Número de características de 'train' antes del preprocesado: 64
Número de características de 'train' después del preprocesado: 41
```

Imagen 3: Número de características antes y después del preprocesamiento

Tal y como vemos los resultados del preprocesamiento son un decremento considerable en el número de características. Además, voy a pintar las matrices de correlación antes y después del preprocesamiento.

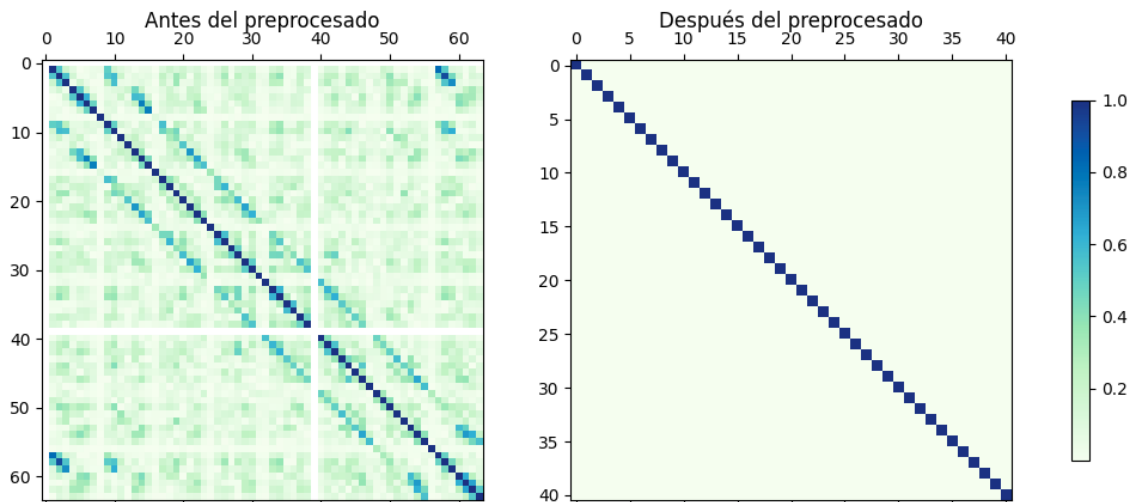


Imagen 4: Matriz de correlación antes y después del preprocesado

La columna y fila de píxeles en blanco en la matriz de correlación antes del preprocesamiento indica que esos valores son constantes (los elimina `VarianceThreshold`).

### 1.5. Fijando la métrica de error a usar

El error que voy a usar es el porcentaje de acierto en la clasificación de los elementos. Este error se denomina *accuracy* y así lo indicaremos a los modelos de `sklearn`.

Es necesario justificar esta elección porque no siempre sería acertada. Si mi base de datos estuviese desbalanceada y hubiese un alto porcentaje de instancias de una misma clase es probable que el error *accuracy* fuese en general bueno aunque el modelo no fuese el idóneo. Sin embargo, aquí los datos están repartidos para cada dígito de manera equitativa.

### 1.6. Discusión de la técnica de ajuste elegida

Para poder establecer una comparación y elegir un modelo de cara a proponer a la empresa es conveniente usar varios. En este caso se va a usar regresión logística y como un modelo no lineal, SVM (*Support-Vector Machine*). Este segundo modelo es sólo una ampliación de la práctica (que está centrada en modelos lineales) de cara a poder comparar si otros modelo mejorarían la clasificación. No obstante aunque el mejor modelo sea SVM se elegirá regresión logística.

En este apartado vamos a establecer las técnicas de ajuste usadas en cada modelo.

- Para regresión logística la optimización que voy a usar es `lbfgs` que además es la que viene por defecto. Es un método `bfgs` con memoria limitada. Calcula el mínimo usando el gradiente pero no la matriz Hessiana, sólo usa una estimación. Esto es una ventaja con respecto al método de Newton.

- La técnica de ajuste que usa SVM es mediante una función de pérdida **hinge** o **squared\_hinge**. He elegido la primera, la cual es:

$$V(f(x), y) = \max\{0, 1 - yf(x)\} = \max\{0, 1 - yw^T x\}.$$

Fíjese que esta función de pérdida es un indicador 0-1 cuando  $\text{sign}(f(x)) = y$  y  $|yf(x)| \geq 1$ . Además de clasificar correctamente maximiza la distancia a los puntos soporte proporcionando la solución buscada.

### 1.7. Discusión de la necesidad de regularización

Considero que en la mayoría de problemas de Aprendizaje Automático la regularización es fundamental ya que controla que el *overfitting* a los datos y ayuda por tanto al correcto aprendizaje de los mismos. Si el error es que comete el modelo es  $J$  la regularización controla la complejidad del modelo limitando el *overfitting* en función añadiendo un término  $\alpha C$ .

El problema de clasificación que estamos trabajando tiene una gran dimensionalidad lo cual favorece el *overfitting*. Por este motivo creo correcto usar la regularización y los resultados que más adelante veremos confirman esta decisión.

En función del valor de  $C$  los dos tipos de regularización más conocidos son  $\ell_1$  o *Regularización Lasso* y  $\ell_2$  o *Regularización Ridge*. He escogido la regularización de tipo  $\ell_2$  que añade como término a la función de error un sumando de la forma  $\alpha \|w\|_2^2$ . En concreto:

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

De esta manera penalizamos con más fuerza los valores muy grandes (cosa que no ocurre en  $\ell_1$ ). El parámetro  $\alpha$  regula el efecto de la regularización.

Información usada: <https://iartificial.net/regularizacion-lasso-l1-ridge-l2-y-elasticnet/>

### 1.8. Identificación de los modelos a usar

Ya hemos adelantado que se van a usar dos modelos (Regresión Logística y SMV) de cara a poder realizar una comparación y escoger el mejor de ellos.

- El primero es Regresión Logística. Hay dos estrategias especialmente conocidas para realizar la separación multiclase. La primera es *One vs Rest* que realiza tantos problemas de clasificación binaria (con regresión logística) como clases haya. La otra es *Multinomial* que realiza la clasificación de manera simultánea entre todas las clases. He elegido la segunda porque minimiza el error de toda la distribución de probabilidad y me parece más eficiente.
- El segundo modelo es Support-Vector Machines (SVM). Encontramos dos estrategias para la separación multiclase. La primera al igual que antes es *One vs Rest* que realiza tantos problemas de clasificación binaria (con SVM) como clases haya. La segunda es *Crammer-Singer* que optimiza un objetivo conjunto sobre todas las clases. En la documentación indica que *Crammer-Singer* es teóricamente consistente pero es algo menos eficiente que *One vs Rest* y los resultados rara vez son mejores. He probado ambos y los resultados sobre este problema son totalmente equivalentes. Finalmente me he decantado por *Crammer-Singer*.

## 1.9. Estimación de hiperparámetros y selección del mejor modelo

He usado la función `LogisticRegression` de `sklearn.linear_model`. La mayoría de los parámetros los he dejado por defecto como `max_iter=1000` y `tol=10-4` (tolerancia del criterio de parada). Aquellos que he cambiado son:

- **C**: He usado 4 valores [0,01, 0,1, 1,0, 10,0] para escoger el mejor.
- **multi\_class**: como ya se anunció se va a usar **multinomial**.

Como hay varios valores de **C** se ha construido un modelo (con el preprocesado indicado) para cada uno de ellos. Mostramos este fragmento de código:

```
""" Funcion para crear una lista de pipelines con el modelo RL para diferentes
valores de C sobre los datos preprocesados. Devuelve dicha lista.
- Cs: Lista de valores C.
"""
def RL_clasificators(Cs):
    # Inicializando lista de Pipeline
    pipes = []

    for c in Cs: # Para cada C se inserta un modelo.
        pipes.append(Pipeline([("var", VarianceThreshold(threshold=0.0)),
                                ("scaled", StandardScaler()),
                                ("PCA", PCA(n_components=0.95)),
                                ("log", LogisticRegression(C=c,
                                                            multi_class='multinomial'))]))

    return pipes
```

He usado la función `LinearSVM` de `sklearn.svm`. La mayoría de los parámetros los he dejado por defecto como `max_iter=1000` y `tol=10-4`. Aquellos que he cambiado son:

- **C**: He usado 4 valores [0,01, 0,1, 1,0, 10,0] para escoger el mejor.
- **random\_state**: se ha establecido a 1 y controla la generación pseudo-aleatoria de números para barajar los datos en el descenso coordinado del problema dual.
- **loss**: como ya se ha explicado la función de pérdida va a ser **hinge**.
- **multi\_class**: también se ha comentado la elección **crammer\_singer**.

De manera análoga a como se hizo antes se muestra el pseudocódigo de los modelos:

```
""" Funcion para crear una lista de pipelines con el modelo SVM para diferentes
valores de C sobre los datos preprocesados. Devuelve dicha lista.
- Cs: Lista de valores C.
"""
def SVM_clasificators(Cs):
    # Inicializando lista de Pipeline
    pipes = []

    for c in Cs: # Para cada C se inserta un modelo.
        pipes.append(Pipeline([("var", VarianceThreshold(threshold=0.0)),
                                ("scaled", StandardScaler()),
                                ("PCA", PCA(n_components=0.95)),
                                ("svm", LinearSVC(C=c, random_state=1, loss='hinge',
                                                  multi_class='crammer_singer'))]))

    return pipes
```

La ejecución de los dos modelos para cuatro valores de **C** distintos se realiza usando validación cruzada dividiendo en 5 conjuntos los datos de entrenamiento. Se han hallado las medias y desviaciones típicas de cara a elegir el mejor modelo. Los resultados son:



Modelo	C	Media	Desv. típica
LR	0.01	0.953704	0.00562458
LR	0.1	0.966783	0.00470287
LR	1	0.966261	0.00739849
LR	10	0.96469	0.00553305
SVM	0.01	0.962858	0.00456008
SVM	0.1	0.962075	0.00611391
SVM	1	0.955796	0.00517247
SVM	10	0.953441	0.00416937

Imagen 5: Resultados de los diferentes modelos

Las varianzas son pequeñas y muy similares entre ellas por lo que atendiendo a las medias se ha escogido regresión logística con  $C=0.1$  como el mejor modelo. A pesar de esta elección los resultados eran similares y escoger SVM también daría una buena clasificación.

### 1.10. Estimación por validación cruzada de $E_{out}$ y comparación con $E_{test}$

En esta sección vamos a entrenar el mejor modelo con todos los datos de entrenamiento y posteriormente se realizarán pruebas con el conjunto de test de cara a estimar  $E_{out}$ . Los resultados son

$$E_{in} \approx 0,02328, \quad E_{out} \approx 0,05676.$$

La matriz de confusión nos proporciona información acerca de qué clases clasifica mejor nuestro modelo cuáles peor. Veamos:

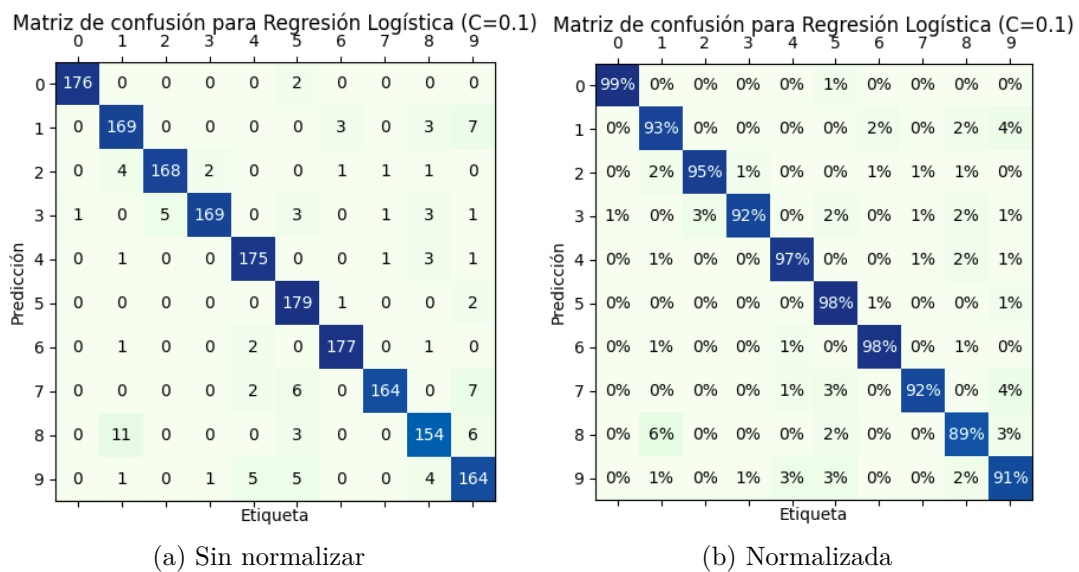


Imagen 6: Matriz de confusión

El error  $E_{out}$  es bastante bueno. Observando la matriz de confusión normalizada la clase que mejor clasifica es la del dígito 0 y la que peor es la del 8. Una posible explicación de por qué es el peor puede ser por el parecido que tiene el dígito 8 a otros dígitos como 0, 6 y 9.

### **1.11. Modelo a proponer a la empresa**

Después del estudio llevado a cabo recomendaría a la empresa la elección del modelo de Regresión Logística con el parámetro de regularización  $C=0.1$ . Les diría que el porcentaje de acierto fuera de la muestra es por lo general mayor al 90 %. Con esta elección tendrían un error satisfactorio a la vez que un modelo con poca complejidad.

Si la empresa necesitase un error más pequeño en la clasificación intentaría optimizar más parámetros o en la validación cruzada dividir en más campos que aunque aumente el tiempo de cómputo es posible que los pesos obtenidos sean mejores. Esto no mejoraría demasiado el modelo así que otra opción es probar con otros modelos y otras clases de hipótesis diferentes.

## 2. Regresión

### 2.1. Comprensión del problema a resolver

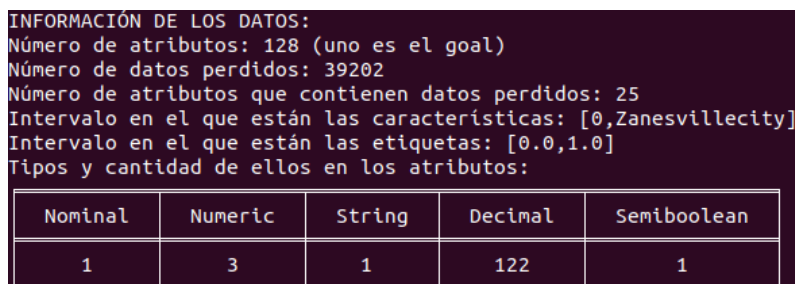
El problema a estudiar está enmarcado en los delitos de diferentes comunidades dentro de los Estados Unidos de América. Los datos combinan datos socioeconómicos del censo estadounidense de la década de los 90, datos de aplicación de la ley de la encuesta LEMAS estadounidense de 1990 y crimen datos de la UCR del FBI de 1995.

El conjunto de datos está compuesto por 128 atributos de los cuales el último es el objetivo a predecir. Este es **ViolentCrimesPerPop** que es el número total de crímenes violentos por cada 100 mil habitantes (valor decimal). Para ello disponemos de 1994 instancias en la base de datos. El resto de los 127 atributos vienen explicados en el fichero `communities.names`:

<http://archive.ics.uci.edu/ml/datasets/Communities+and+Crime>

El problema es un problema de aprendizaje supervisado en el que identificamos  $\mathcal{X}$  como los 127 atributos restantes de los cuales 5 son no predictivos y los otros 122 son valores en el intervalo  $[0, 1]$  ya que se proporcionan normalizadas. Las etiquetas  $\mathcal{Y}$  (correspondientes al atributo **ViolentCrimesPerPop**) están normalizadas al  $[0, 1]$ . Por último la función de clasificación  $f$  es aquella que hace corresponder las características de cualquier instancia con su etiqueta, es decir,  $f : \mathcal{X} \rightarrow \mathcal{Y}$  y para cada  $(x_n, y_n) \in \mathcal{X} \times \mathcal{Y}$  se verifica  $f(x_n) = y_n$ .

Para comprender mejor el problema vamos a mostrar alguna información general del conjunto de datos:



```
INFORMACIÓN DE LOS DATOS:
Número de atributos: 128 (uno es el goal)
Número de datos perdidos: 39202
Número de atributos que contienen datos perdidos: 25
Intervalo en el que están las características: [0,Zanesvillecity]
Intervalo en el que están las etiquetas: [0.0,1.0]
Tipos y cantidad de ellos en los atributos:
```

Nominal	Numeric	String	Decimal	Semiboollean
1	3	1	122	1

Imagen 7: Información general de los datos

Como vemos en la base de datos existen abundantes valores perdidos correspondientes a 25 atributos que será necesario tratar más adelante. Como existen diferentes tipos de datos el cálculo del intervalo de las características no se realiza como esperamos. Solucionaremos esto más adelante. También observamos los diferentes tipos del dataset. Una vez que prescindimos de los 5 no predictivos los tipos se reducen a 122 de tipo decimal y uno que es semiboollean. Este atributo especial es **LemasGangUnitDeploy** que indica la unidad de pandillas desplegada en donde 0 significa NO, 1 significa SÍ y 0,5 significa a tiempo parcial.

Para llegar a un nivel más alto de entendimiento del problema sería muy interesante analizar cómo se reparte el valor de la variable objetivo. Seguramente habrá muchos estados con pocos crímenes violentos y pocos estados con muchos de estos. Observemos los datos en la siguiente tabla:

Intervalo	Núm. instancias
[0.0, 0.1)	679 (34.05%)
[0.1, 0.2)	470 (23.57%)
[0.2, 0.3)	285 (14.29%)
[0.3, 0.4)	174 (8.73%)
[0.4, 0.5)	97 (4.86%)
[0.5, 0.6)	98 (4.91%)
[0.6, 0.7)	69 (3.46%)
[0.7, 0.8)	32 (1.6%)
[0.8, 0.9)	33 (1.65%)
[0.9, 1.0]	57 (2.86%)

Imagen 8: Cantidad de instancias en intervalos de etiquetas

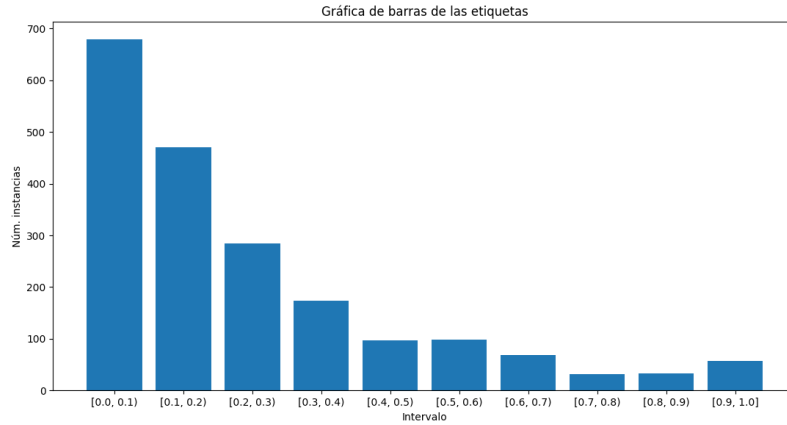


Imagen 9: Gráfica de barras de las etiquetas

Se ha construido una gráfica de barras asociada a estos datos:

Efectivamente los datos se agrupan como habíamos anunciado. Es lógico que así sea y también es visible un repunte en el intervalo  $[0.9, 1]$ .

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

En la búsqueda de la función objetivo  $f$  es necesario fijar la clase de funciones o conjunto de hipótesis  $\mathcal{H}$ . El objetivo es escoger  $g \in \mathcal{H}$  de modo que  $g \approx f$ .

La clase de funciones elegida es la de combinaciones lineales de los datos, al igual que se hizo en el problema de clasificación. Con las dimensiones que se manejan en este problema usar una clase no lineal puede ser problemático y dar lugar a *overfitting*.

$$\mathcal{H}_{reg} = \left\{ w_0 + \sum_{i=1}^N w_i x_i, \quad w \in \mathbb{R}^{N+1} \right\}.$$

### 2.3. Fijando los conjuntos de training y test

En este problema la división de estos dos conjuntos está dejada totalmente al albedrío de lo usuarios de la base de datos. En mi caso he separado un 25 % de las instancias para el conjunto de test y el otro 75 % las voy a usar para entrenar.

Para ello se ha usado `train_test_split` de `sklearn.model_selection` de la siguiente forma:

```
X_train, X_test, y_train, y_test = train_test_split(newX, y, test_size=0.25)
```

El resultado de la separación es:

```
Separando en 'train' y 'test' los datos de 'communities'
Núm. instancias: 1495 (train) 499 (test)
Porcentaje (%): 74.975 (train) 25.025 (test)
```

Imagen 10: Separación en train y test

Es de vital importancia cuestionarse si el reparto de las instancias ha sido totalmente aleatorio y hay un estratificado por los intervalos similar al original de la Imagen 8.

Intervalo	Núm. instancias 'train'	Núm. instancias 'test'
[0.0, 0.1)	502 (33.58%)	177 (35.47%)
[0.1, 0.2)	358 (23.95%)	112 (22.44%)
[0.2, 0.3)	217 (14.52%)	68 (13.63%)
[0.3, 0.4)	117 (7.83%)	57 (11.42%)
[0.4, 0.5)	75 (5.02%)	22 (4.41%)
[0.5, 0.6)	75 (5.02%)	23 (4.61%)
[0.6, 0.7)	57 (3.81%)	12 (2.4%)
[0.7, 0.8)	28 (1.87%)	4 (0.8%)
[0.8, 0.9)	25 (1.67%)	8 (1.6%)
[0.9, 1.0]	41 (2.74%)	16 (3.21%)

Imagen 11: Reparto de las instancias después de la partición

Efectivamente es muy similar y estamos satisfechos con la división de ambos conjuntos. Antes de continuar con el preprocesado de los datos mostramos las gráficas de barras asociadas.

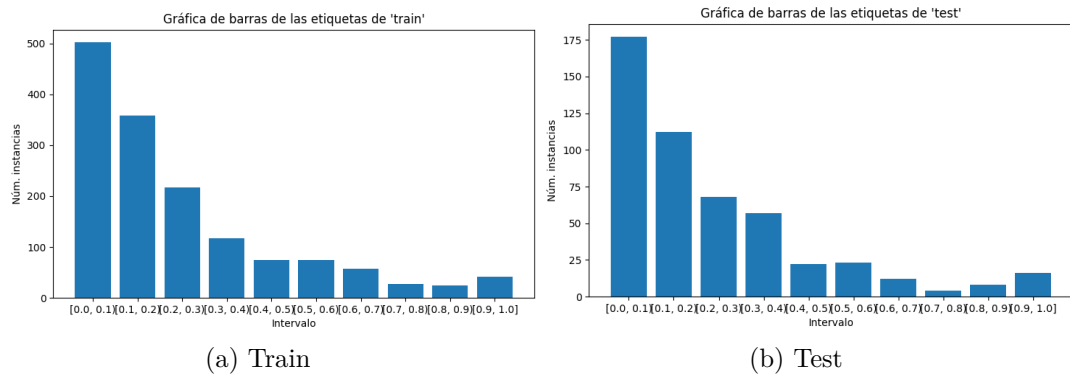


Imagen 12: Gráfica de barras de los datos

## 2.4. Preprocesado de los datos

El primer paso que llevado a cabo en el preprocesado de los datos es descartar aquellos atributos no predictivos. Éstos eran los cinco primeros correspondientes a:

`'state'`, `'county'`, `'community'`, `'communityname'`, `'fold'`.

Después de este paso se pueden tratar a los atributos como `float` y puedo hacer operaciones sobre ellos. Para ello es necesario cambiar el carácter `'?'` por `NAN`. Se había detectado la cantidad de valores perdidos de esta base de datos. Vamos a realizar un conteo de ellos y ver a qué atributos se corresponden para tomar una decisión.

Cabecera	Val. perdidos	Porcentaje (%)
OtherPerCap	1	0.05
LemasSwornFT	1675	84
LemasSwFTPerPop	1675	84
LemasSwFTFieldOps	1675	84
LemasSwFTFieldPerPop	1675	84
LemasTotalReq	1675	84
LemasTotReqPerPop	1675	84
PolicReqPerOffic	1675	84
PolicPerPop	1675	84
RacialMatchConnPol	1675	84
PctPolicWhite	1675	84
PctPolicBlack	1675	84
PctPolicHisp	1675	84
PctPolicAsian	1675	84
PctPolicMinor	1675	84
OfficAssgnDrugUnits	1675	84
NumKindsDrugsSeiz	1675	84
PolicAveOTWorked	1675	84
PolicCars	1675	84
PolicOperBudg	1675	84
LemasPctPolicOnPatr	1675	84
LemasGangUnitDeploy	1675	84
PolicBudgPerPop	1675	84

Imagen 13: Valores perdidos de los atributos

Observamos que en la mayoría de los atributos el porcentaje de valores perdidos es mayor al 80 % por lo que es obligatorio descartarlos. Cualquier sustitución o intento de sustitución de esos valores perdidos desvirtuaría demasiado la información y las predicciones futuras podrían resultar inadecuadas.

Respecto al atributo `OtherPerCap` sólo posee un valor perdido. Sería una mala idea descartar todo el atributo por ese valor perdido. Una opción es hacer la media de los datos que sí poseemos de ese atributo. Pero pensándolo un poco más detenidamente podemos hacer la media de los datos del mismo estado (atributo `state`) que el del dato perdido.

Hemos llevado a cabo la última opción. De manera teórica también es posible sustituir los datos perdidos por la suma del valor medio de la variable más un valor aleatorio en el intervalo  $[-1,5\sigma, 1,5\sigma]$ . Resultado:

```
Índices de atributos con menos de un 10% de datos perdidos que hay que tratar:
[25]
Haciendo la media de las instancias del mismo estado para dicho atributo...
Datos perdidos en las posiciones: [130]
'State' del 'outlier': 28
Hay 18 coincidencias de estado y se inserta el valor medio: 0.2422
```

Imagen 14: Sustitución del valor perdido

El valor sustituido es 0.2422. Para continuar con el preprocesado en este caso después de las pruebas realizas PCA (Análisis de Componentes Principales) no mejora mucho. Una posibilidad es por la no existencia o una existencia suave de *overfitting*. La única comprobación que me queda por hacer es mirar si existen atributos constantes (o con varianza cero). Comprobamos:

```
PREPROCESANDO LOS DATOS
Número de características con varianza cero: 0
```

Imagen 15: Características con varianza cero

Dando por finalizado el preprocesamiento vamos a mostrar la matriz de correlación. En ella se observan algunas zonas con un azul más fuerte. Esto quiere decir que el valor es más cercano a 1 y los atributos correspondientes a la fila y columna de ese elemento están correladas.

No obstante en la Imagen 16 esto no ocurre con frecuencia. Podríamos eliminar manualmente alguna de las variables más correladas pero realmente no influiría demasiado en el error final de la regresión.

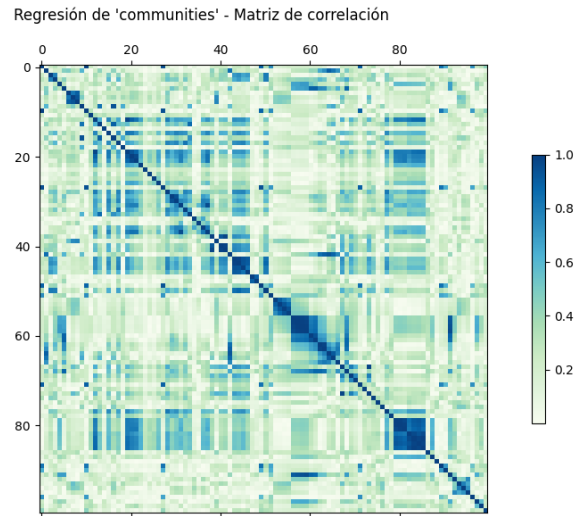


Imagen 16: Matriz de correlación

Finalmente si se vuelve a imprimir por pantalla la información de los datos (ya preprocesados) se obtiene quedan 101 atributos de los cuales uno es el que hay que predecir. El intervalo de las características es como se espera solucionando el fenómeno de la Imagen 7.

```
INFORMACIÓN DE LOS DATOS:
Número de atributos: 101 (uno es el goal)
Número de datos perdidos: 0
Número de atributos que contienen datos perdidos: 0
Intervalo en el que están las características: [0.0,1.0]
Intervalo en el que están las etiquetas: [0.0,1.0]
```

Imagen 17: Información de los datos después del preprocesado

## 2.5. Fijando la métrica de error a usar

En <https://sitiobigdata.com/2018/08/27/machine-learning-metricas-regresion-mse/> se dispone de información respecto de este tema. En concreto aquí se va a usar el error cuadrático medio (MSE):

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

El error cuadrático medio básicamente calcula el cuadrado de la diferencia entre el valor real  $y_i$  y la predicción  $\hat{y}_i$ .

La otra medida que vamos a tener, que además va a ser nuestra métrica principal es el coeficiente de determinación  $R^2$ :

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



donde  $\bar{y}$  es la media de las  $y_i$ . De este modo el numerador es el MSE de las predicciones y el denominador es el MSE de un modelo base en el que las predicciones son la media. Debido a esto es invariante a la escala, da igual si los valores son grandes o pequeños. Otro detalle importante es que  $R^2 \in (-\infty, 1]$  y  $R^2 = 1$  es lo mejor posible (todas las etiquetas correctamente predichas).

## 2.6. Discusión de la técnica de ajuste elegida

Se va usar un modelo de regresión lineal basado en la técnica de ajuste Gradiente Descendente Estocástico. Este tipo de descenso resulta especialmente eficiente cuando el número de datos a manejar es alto (es nuestro caso) y los datos son a gran escala.

La técnica de ajuste SGD actualiza los pesos cada vez mediante el gradiente y la tasa de aprendizaje.

$$w = w - \eta \nabla E_{in}.$$

El valor de la tasa de aprendizaje es fundamental para el buen ajuste de la técnica. Su valor puede ser constante o puede actualizarse. Discutiremos este parámetro más adelante.

## 2.7. Discusión de la necesidad de regularización

Los tipos de regularización más conocidos ya se explicaron con detalle para el problema de clasificación (Sección 1.7). De manera análoga a como se hizo allí el conjunto de datos actual tiene una dimensión alta por lo que la regularización es un aspecto a tener en cuenta para el buen ajuste del modelo a los datos.

El mejor tipo de regularización después de diferentes pruebas para éste modelo es  $\ell_2$  o *Ridge*. Esta regularización ayuda a omitir el efecto de la correlación entre algunos atributos.

## 2.8. Identificación de los modelos a usar

Como ya se ha ido adelantando el modelo lineal que voy a usar para el problema de regresión es `SGDRegressor` disponible en `sklearn.linear_model`. Su técnica de ajuste es gradiente descendente estocástico y la regularización a usar la *Ridge*.

## 2.9. Estimación de hiperparámetros y selección del mejor modelo

Los parámetros más importantes y los valores elegidos de este modelo son:

- `loss='squared_loss'` Es la función de pérdida y se refiere a los mínimos cuadrados. Es óptima para mejorar la métrica  $R^2$ .
- `penalty='l2'` Regularización elegida y explicada.
- `max_iter=1000` El número máximo de iteraciones (es el valor por defecto).
- `alpha` Constante que multiplica el término de la regularización. Este parámetro es importante por lo que se va a realizar pruebas con diferentes valores:  $[0,00001, 0,0001, 0,001]$ .
- `tol` Tolerancia para el criterio de parada. También se han realizado pruebas con varios valores:  $[1e-9, 1e-8, 1e-7]$ .

- **shuffle=True** Indica que se desea barajar los datos en cada época.
- **learning\_rate='invscaling'** Es la tasa de aprendizaje y se ha elegido una que sí actualiza conforme avanza el algoritmo. En concreto **invscaling** indica que se desea actualizar conforme a  $\eta = \eta_0 / \text{pow}(t, \text{power\_t})$ . Los parámetros se han fijado por defecto:  $\eta_0=0.01$  y  $\text{power\_t}=0.25$ .
- **validation\_fraction=0.1** Proporción del conjunto de datos para usar como validación. El valor 0.2 me daba peores resultados por lo que lo establecí en 0.1.

Como hay varios valores de **alpha** y **tols** se ha construido un modelo para cada uno de ellos. Mostramos este fragmento de código:

```
""" Funcion para crear una lista de modelos basados en SGDR para diferentes valores
de alpha y tol sobre los datos preprocesados. Devuelve dicha lista.
- alphas: lista de valores de alpha.
- tols: lista de tolerancias.
"""
def SGD_regressors(alphas, tols):
    models = []
    for al in alphas:
        for to in tols:
            models.append(SGDRegressor(loss="squared_loss", penalty="l2",
                                      alpha=al, tol=to))
    return models
```

A continuación se han entrenado todos los modelos con el conjunto de entrenamiento y se han evaluado obteniendo los valores de la métrica  $R^2$  tanto para el conjunto de entrenamiento como para el de test. Los resultados son:

Modelo	Alpha	Tol.	R <sup>2</sup> 'train'	R <sup>2</sup> 'test'
SGDR	1e-05	1e-09	0.677579	0.623585
SGDR	1e-05	1e-08	0.674869	0.622873
SGDR	1e-05	1e-07	0.676164	0.623709
SGDR	0.0001	1e-09	0.672437	0.618146
SGDR	0.0001	1e-08	0.674197	0.622222
SGDR	0.0001	1e-07	0.670412	0.621365
SGDR	0.001	1e-09	0.677248	0.622565
SGDR	0.001	1e-08	0.670999	0.621536
SGDR	0.001	1e-07	0.671176	0.621778

Imagen 18: Resultados de los diferentes modelos

El criterio que he seguido para elegir el mejor modelo (los mejores hiperparámetros) es aquel que tenga mejor valor de  $R^2$  fuera de la muestra. Así el máximo es 0,623709 correspondiente a **alpha=1e-05** y **tol=1e-07**.

## 2.10. Estimación por validación cruzada de $E_{out}$ y comparación con $E_{test}$

Los errores obtenidos para el modelo elegido son:

$$\begin{aligned}R_{in}^2 &\approx 0,67616 \\ R_{out}^2 &\approx 0,62371 \\ MSE &\approx 0,019787\end{aligned}$$

El MSE es claramente sobre el conjunto de test. Es un error excelente desde luego y podemos afirmar que la regresión ha sido exitosa. Respecto a  $R^2$  también es bastante bueno. El mejor valor posible es 1 y no está muy lejos.

Que el valor de la métrica  $R^2$  sea similar en el conjunto de entrenamiento y el de test es satisfactorio porque indica que no existe *overfitting*.

## 2.11. Modelo a proponer a la empresa

Recomendaría a la empresa el modelo de regresión basado en SGD con los parámetros explicados (en particular `alpha=1e-05` y `tol=1e-07`).

Les diría que pueden usar las predicciones con un error bastante pequeño debido al MSE obtenido. Además el número de instancias que se han usado, 1994, es bastante grande lo cual asegura mejor el éxito. Del mismo modo les diría que gracias a las técnicas de regularización usadas se ha prevenido de un posible sobreajuste a los datos de entrenamiento. Esto evita un posterior error en instancias fuera de la muestra que son aquellas que ellos están interesados en predecir.