Aprendizaje automático

Práctica 3: Ajuste de modelos lineales

Antonio Coín Castro

Curso 2019-20

Índice

Introducción

En esta práctica perseguimos ajustar el mejor modelo lineal en dos conjuntos de datos dados, para resolver un problema de clasificación y otro de regresión. En ambos casos seguiremos una estrategia común que nos llevará finalmente a elegir un modelo y estimar su error.

1. Analizaremos la bases de datos y entenderemos el contexto del problema a resolver.
2. Preprocesaremos los datos de forma adecuada para trabajar con ellos.
3. Elegiremos una clase de hipótesis (lineal) para resolver el problema.
4. Fijaremos algunos modelos concretos y seleccionaremos el mejor según algún criterio.
5. Estimaremos el error del modelo.

Trabajamos en su mayoría con las funciones del paquete `scikit-learn`, apoyándonos cuando sea necesario en otras librerías como `numpy`, `matplotlib` ó `pandas`. El código de la práctica se ha estructurado en tres *scripts* de `Python`:

- En `p3_classification.py` se resuelve el problema de clasificación.
- En `p3_regression.py` se resuelve el problema de regresión.
- En `p3_visualization.py` se recogen todas las funciones de visualización de gráficas, tanto comunes como propias de cada problema.

La ejecución de los dos programas principales está preparada para que se muestren solo algunas gráficas además del procedimiento de ajuste del modelo (aquellas que consumen menos tiempo). Este comportamiento puede cambiarse mediante el parámetro `show` de la función principal en cada caso, eliminando todas las gráficas (valor 0) o mostrándolas todas (valor 2). Además, en las operaciones de cálculo intensivo que lo permitan se utiliza el parámetro `n_jobs = -1` para paralelizar el flujo de trabajo en tantas hebras como se pueda. Por pantalla se muestra información sobre el tiempo de ejecución de los ajustes de los modelos y del programa completo.

Para que los experimentos sean reproducibles se fija una semilla aleatoria al incio del programa. Todos los resultados y gráficas que se muestran a continuación se han obtenido con el valor 2020 para la semilla, y pueden reproducirse si se ejecuta el programa tal y como se proporciona.

Bases de datos y descripción del problema

Ambas bases de datos se han descargado del repositorio UCI.

Clasificación: *optical recognition of handwritten digits*

El primer conjunto de datos es un conjunto para clasificación. Se trata de un conjunto de 5620 dígitos manuscritos codificados en 64 atributos de tipo entero, al que contribuyeron 43 personas diferentes escribiendo dígitos. Cada dígito se escaneó con una resolución

de 32×32, siendo cada píxel blanco o negro. Posteriormente, para cada bloque 4×4 no solapado se calculó el número de píxeles negros que hay en ese bloque, obteniendo finalmente 64 atributos que tienen un valor entero entre 0 y 16. La salida o variable de clase es un atributo categórico entre 0 y 9 que representa el dígito dibujado.

Por tanto, como disponemos de las etiquetas reales de cada instancia estamos ante un problema de aprendizaje supervisado; en particular, un problema de **clasificación multiclase**. En principio, consideramos como espacio muestral $\mathcal{X} = \{0, \dots, 16\}^{64}$ y como conjunto de etiquetas $\mathcal{Y} = \{0, \dots, 9\}$. Queremos aprender o estimar una función $f : \mathcal{X} \to \mathcal{Y}$ que asigne a cada muestra codificada como hemos dicho anteriormente su correspondiente dígito.

Podemos visualizar inicialmente la distribución de clases, tanto en el conjunto de entrenamiento como en el de *test*. Aunque ya nos lo decían en el informe de la base de datos, podemos observar en la Figura 1 que la distribución de clases es uniforme en ambos conjuntos; no hay problema de clases desbalanceadas.

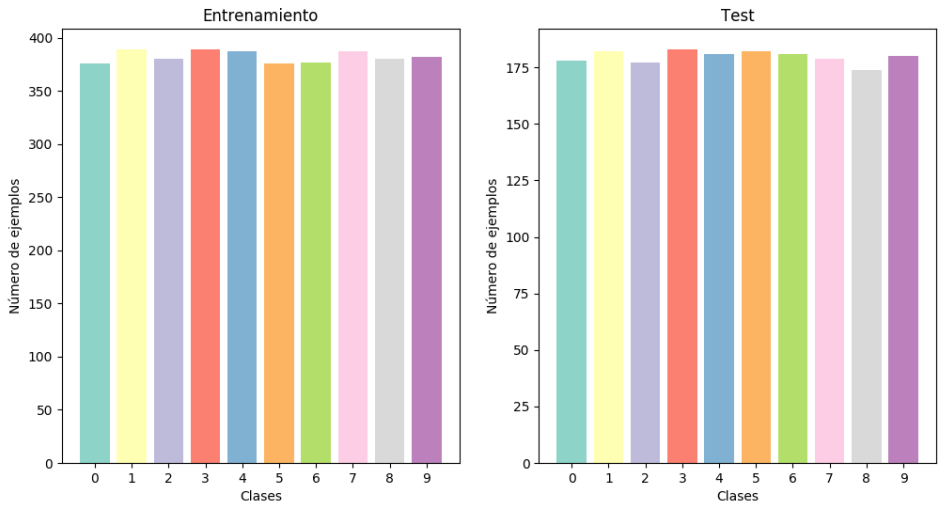


Figura 1: Distribución de clases en entrenamiento y *test* para el problema de clasificación.

TODO: imagen de alguna cifra + TSNE. Comentar que se espera una buena calidad de predicción porque hay grupos.

Regresión: *communities and crimes*

El segundo conjunto de datos con el que trabajamos es un conjunto para regresión. Esta vez se trata de 1994 instancias que recogen información de tipo muy variado sobre ciertas regiones de Estados Unidos, para intentar predecir a partir de ellas el número total de crímenes violentos por cada 100.000 habitantes en dicha región. Por ejemplo,

se recoge información sobre el número de habitantes, su raza, la renta *per cápita* o el porcentaje de familias divorciadas, entre otros.

Cada ejemplo consta de 128 atributos, de donde los cinco primeros son no predictivos (*state*, *county*, *community*, *communityname* y *fold*), los 122 atributos son predictivos (con valores reales) y el último, de tipo real, es la variable que queremos predecir (`ViolentCrimesPerPop`). Se trata entonces de un problema de aprendizaje supervisado; en particular, un problema de **regresión**. Nos dicen que todos los datos están normalizados al intervalo $[0, 1]$, por lo que consideramos $\mathcal{X} = [0, 1]^{122}$, $\mathcal{Y} = [0, 1]$ y queremos aprender una función $f : \mathcal{X} \to \mathcal{Y}$ que asigne a cada ejemplo la cantidad (normalizada) de crímenes violentos que se producen por cada 100.000 habitantes.

Para intentar visualizar los datos hemos seguido dos estrategias. Por un lado, se han seleccionado (mediante el criterio de importancia que otorga un `Random Forest`) las dos variables con mayor relevancia y poder predictivo, que resultan ser el tanto por uno de niños de 4 años o menos en familias biparentales (`PctYoungKids2Par`) y el tanto por uno de hombres que nunca han estado casados (`MalePctNevMarr`). Una representación de las mismas junto a la variable a predecir puede verse en la Figura 2, donde observamos que hay una correlación substancial (en el primer caso positiva y en el segundo negativa).

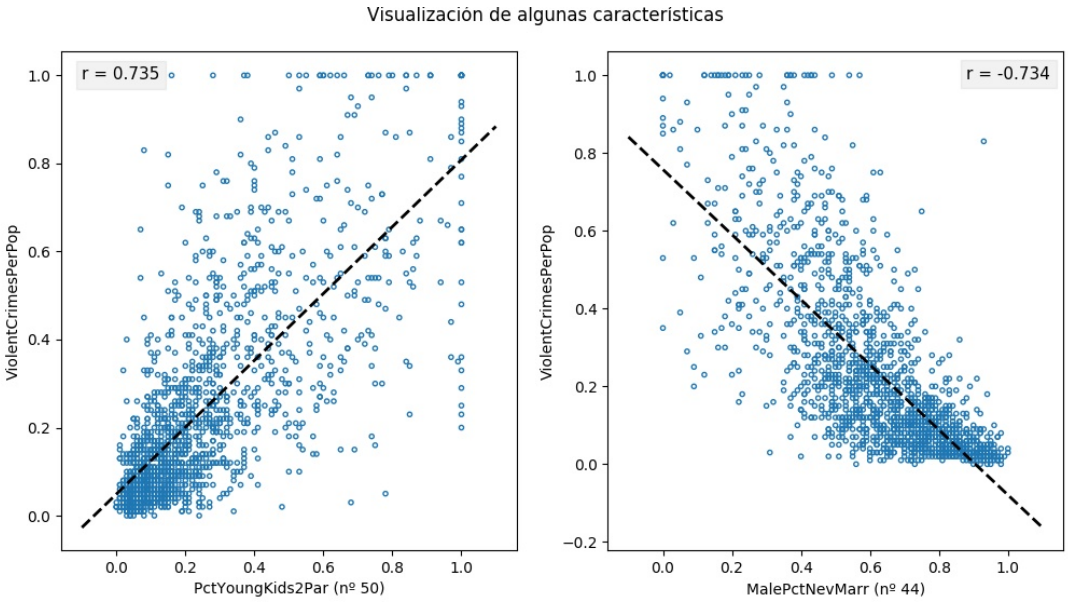


Figura 2: Representación de algunas características frente a la variable a predecir, junto al coeficiente de correlación lineal.

Por otro lado, podemos estudiar cómo es la distribución de la variable a predecir. Para ello realizamos un histograma de la misma en la Figura 3, donde vemos que en la mayoría de ejemplos el número de crímenes violentos se encuentra en la parte inferior del espectro,

mientras que un número muy alto de crímenes es relativamente raro.

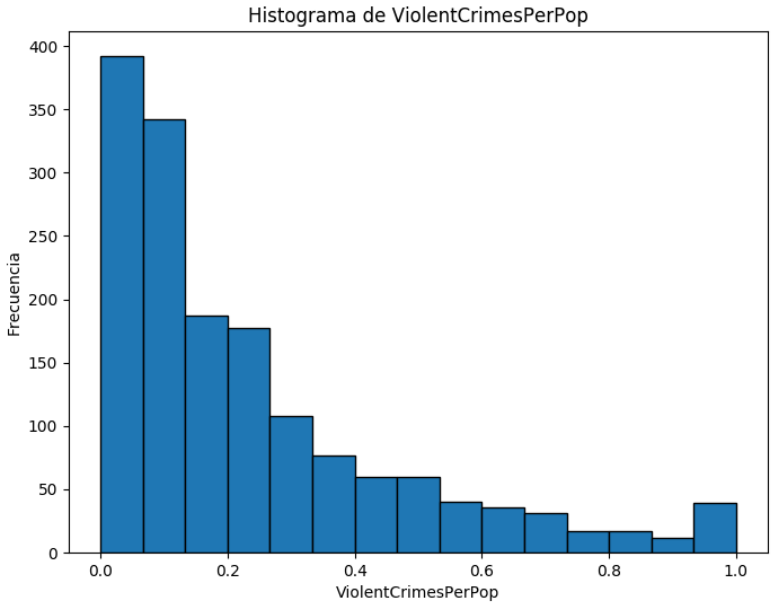


Figura 3: Histograma de la variable `ViolentCrimesPerPop`.

Selección de la clase de funciones

Como veremos en la sección Preprocesado de datos, emplearemos una técnica de selección de variables para reducir la dimensionalidad del problema y evitar los inconvenientes que un elevado número de variables conlleva. Así podremos seleccionar una clase de funciones de combinaciones cuadráticas de las observaciones sin elevar de forma exagerada la dimensión de cada punto. De esta manera aumentamos la flexibilidad del modelo, pues presuponemos que los datos no serán linealmente separables en el espacio original, pero podemos aspirar a que lo sean en el espacio transformado. Estamos pagando un precio al aumentar también la complejidad del conjunto de hipótesis, pero en ambos casos mi opinión es que los problemas a los que nos enfrentamos son lo suficientemente complejos como para que esté justificado el empleo de tranformaciones polinómicas. También corremos el riesgo de caer en el sobreajuste, pero añadiremos a nuestros modelos técnicas de regularización que ayuden a paliar este problema.

No considero que aumentar el grado de la transformación polinómica por encima de 2 vaya a resultar en un mejor ajuste, y además ya tendríamos un número excesivo de componentes que se traduciría en un mayor tiempo de ejecución y un riesgo más elevado de sobreajuste.

Como hemos comentado vamos a realizar selección de variables, por lo que si el espacio original era d-dimensional, de cara a expresar la clase de funciones trataremos con un espacio $\mathcal{X}'$ de dimensión d' ($d' \leq d$) genérico, resultado de la eventual selección que hagamos. Consideramos entonces para cada $x = (x_1, \ldots, x_{d'}) \in \mathcal{X}'$ la transformación

$$\Phi_2(x) = (1, x_1, \ldots, x_{d'}, x_1^2, \ldots, x_{d'}^2, x_1x_2, x_1x_3, \ldots, x_1x_{d'}, x_2x_3, \ldots, x_{d-1}x_{d'}),$$

que incluye el término de *bias* al principio, resultando en que el espacio transformado $\mathcal{Z} = \Phi_2(\mathcal{X}')$ tiene dimensión

$$\tilde{d} = 1 + 2d' + \frac{d'(d'-1)}{2}.$$

Clasificación

Como estamos ante un problema de clasificación multiclase, debemos hacer una serie de ajustes para expresar la clase de hipótesis, empleando la técnica de clasificadores *one-versus-all*. En primer lugar, consideramos para cada $i \in \{0, \ldots, 9\}$ el clasificador binario en el espacio transformado $\mathcal{Z}$ dado por

$$h_i(z) = w^T z, \quad w \in \mathbb{R}^{\tilde{d}},$$

que salvo un reescalado por $||w||^{-1}$ mide la distancia con signo del punto z al hiperplano definido por w. Interpretaremos que una distancia positiva se corresponde a un etiquetado a favor de la i-ésima clase, y una distancia negativa a *cualquiera de las otras*. A la hora de predecir la clase de una instancia, digamos $c(z)$, elegiremos aquella clase para la cual la clasificación positiva sea más fuerte, o en caso de no haber ninguna, aquella que se quede más cerca de la frontera de clasificación. Esto se puede expresar como

$$c(z) = \arg\max_i \ h_i(z).$$

Por tanto, la clase de funciones en este caso quedaría

$$\mathcal{H}_C = \left\{ \arg\max_i w_{(i)}^T \Phi_2(x) : w_{(i)} \in \mathbb{R}^{\tilde{d}},\ i = 0, \ldots, 9 \right\}.$$

Notamos que estamos considerando un clasificador binario por cada clase, de forma que el número total de parámetros del modelo sería $10\tilde{d}$.

Regresión

En este caso la clase de funciones no requiere ningún tratamiento especial; se trata simplemente de las combinaciones lineales en el espacio transformado $\mathcal{Z}$:

$$\mathcal{H}_R = \left\{ w_0 + \sum_i w_i x_i + \sum_{i \leq j} w_{ij} x_i x_j : w_i, w_{ij} \in \mathbb{R} \right\} = \left\{ w^T \Phi_2(x) : w \in \mathbb{R}^{\tilde{d}} \right\}.$$

Conjuntos de entrenamiento, validación y *test*

La estrategia para la selección de modelos será usar *K-fold cross validation*, por lo que no es necesario considerar ningún conjunto de validación explícito.

Clasificación

En este caso, el conjunto de datos viene ya dividido en entrenamiento y *test*: disponemos de 3823 ejemplos de entrenamiento (sobre un 70 %) y 1797 ejemplos de *test* (sobre un 30 %). No veo ningún motivo para alterar esta división; de hecho, nos dicen que los dígitos manuscritos en ambos conjuntos fueron dibujados por personas distintas, por lo que es deseable que esta división se mantenga. De esta forma cuando evaluemos el modelo no solo intentará predecir ejemplos que no ha visto antes, sino que estos habrán estado dibujados por personas distintas a aquellos con los que fue entrenado.

Regresión

En este caso, debemos establecer nosotros la separación en entrenamiento y test. Debido a que no tenemos un número excesivo de ejemplos, se ha decidido hacer una división de 80 % para entrenamiento y 20 % para *test*, mediante una llamada a la función `train_test_split`.

Preprocesado de datos

Hemos visto que en ambos conjuntos los datos ya están normalizados en el sentido de que todas las columnas se mueven en el mismo rango. Sin embargo, para aumentar nuestras posibilidades de obtener un buen ajuste debemos realizar un preprocesado adicional de los datos. Es importante que cualquier transformación de los datos se haga primero únicamente en el conjunto de entrenamiento, y después a la hora de evaluar se haga la misma transformación (con los mismos parámetros) en el conjunto de *test*.

Para la fase de preprocesado usaremos los *pipelines* de `sklearn` [1], que nos permiten aplicar las mismas transformaciones en varias fases de forma cómoda. En ambos casos, consideramos cinco pasos en el preprocesado: tratamiento de valores perdidos, selección de variables, transformaciones polinómicas, umbral de varianza y estandarización.

Valores perdidos

Es importante tratar con los valores perdidos, pues en otro caso no podremos entrenar correctamente ningún modelo.

[1] Documentación sobre `Pipeline` en `sklearn`.

Clasificación

En el conjunto de dígitos no hay ningún valor perdido, por lo que este paso podemos saltarlo. Tampoco encontramos valores que podamos considerar inconsistentes.

Regresión

En esta ocasión nos encontramos que 22 columnas tienen más de un 80 % de valores perdidos. Decidimos eliminar directamente los predictores asociados de nuestro conjunto de datos, pues difícilmente vamos a poder establecer ninguna relación útil para predecir. Tras esto, nos queda un único valor perdido en uno de los ejemplos, al que decidimos atribuirle la mediana de su columna. Por un lado, decidimos rellenar el valor porque no tenemos ejemplos de sobra y la información que aportan el resto de sus atributos puede ser útil, y por otro decidimos usar la mediana como estrategia por su robustez ante posibles *outliers*.

Selección de características

Como ya comentamos, reducir la dimensionalidad de los datos es un paso muy importante, pues además de una reducción en el tiempo de entrenamiento de los modelos, podemos eliminar características poco relevantes y/o con alta correlación que haga que la calidad de los modelos aumente. También podemos combinar las características para obtener otras con mayor poder predictivo. En general, el fenómeno que hace que los modelos en *machine learning* empeoren o sean casi inviables de utilizar conforme aumenta la dimensión se conoce como *maldición de la dimensionalidad*.

En cualquier caso, se considera la opción de no realizar selección de características de ningún tipo, para constatar *a posteriori* que efectivamente obtenemos una mejoría en los resultados, además de en el tiempo de ejecución.

Clasificación

A la hora de realizar selección de variables se han considerado dos alternativas: análisis de componentes principales (PCA) ó análisis de varianza (ANOVA). La primera se trata de una técnica de reducción de dimensionalidad que considera combinaciones lineales de las variables, llamadas componentes principales, de forma que la primera recoge la mayor varianza según una cierta proyección de los datos, la segunda la segunda mayor, etc. La segunda alternativa, por su parte, basa su funcionamiento en el test estadístico conocido como *F-test*, que en este contexto se utiliza para estimar el grado de dependencia lineal de las variables dos a dos. Una vez hecho esto, se ordenan las variables de la más a la menos discriminativa.

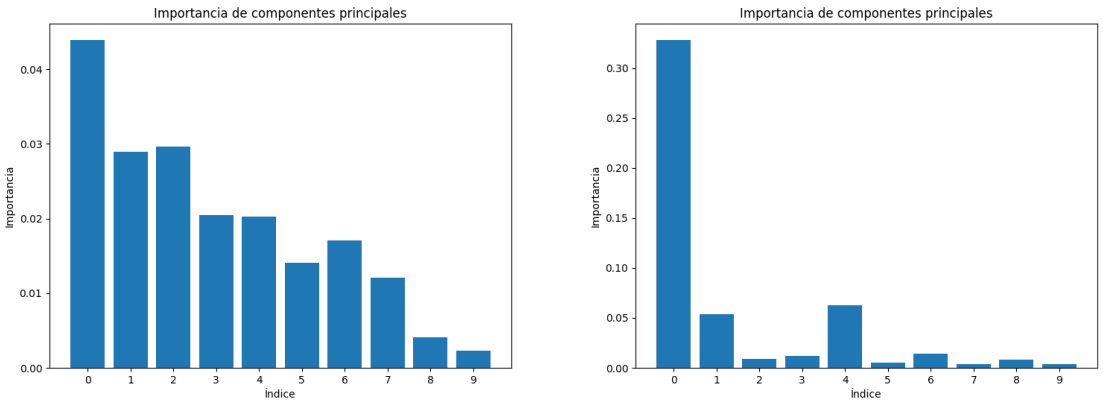
En el caso de PCA establecemos que la varianza acumulada de las variables resultantes debe ser del 95 % de la original, obteniendo **29 variables** tras su aplicación. Para el test ANOVA imponemos que se seleccionen de entre todas las variables las 32 que mejor puntuación obtengan (la mitad).

Se han elegido estas dos estrategias como representantes de los dos grandes campos en la selección de variables: PCA realiza una selección *no supervisada*, mientras que el análisis de varianza es un método *supervisado* de selección (es decir, utiliza la información de las etiquetas). Se estimó que era interesante realizar una comparación entre estos métodos para ver cuál de ellos tenía un mejor desempeño.

Regresión

En el caso de regresión, como los datos recogidos son de naturaleza muy dispar y no tan uniformes como en el caso de los dígitos, optamos por fijar directamente una aplicación de PCA con un valor más agresivo de reducción, imponiendo únicamente que se tenga que explicar el 80 % de la varianza del conjunto. Fijamos un valor más pequeño porque intuimos que puede haber muchas fuentes de variabilidad distinta en los datos, y que el porcentaje de ruido puede ser mayor.

Podemos ver en la Figura 4 cómo se refleja la transformación de componentes principales que realiza PCA en el criterio de importancia predictiva de las variables, medido de nuevo a través de un `Random Forest`. Observamos que efectivamente la primera componente principal es en ambos caso la que más relevancia tiene, y en el caso del problema de clasificación se aprecia mejor como en general la relevancia va disminuyendo conforme aumentamos el índice (pues la varianza explicada por cada variable va disminuyendo).



(a) Clasificación.

(b) Regresión.

Figura 4: Relevancia de componentes principales en clasificación y regresión.

Transformaciones polinómicas

Como ya comentamos cuando definimos la clase de funciones, emplearemos transformaciones polinómicas de segundo grado. Simplemente llamaremos a la función

`PolynomialFeatures(2)`, que se encarga de realizar las transformaciones pertinentes.

Estandarización y umbral de varianza

En primer lugar, procederemos a eliminar las variables con varianza o, pues son constantes, no aportan ninguna información y pueden provocar problemas en el ajuste. También eliminaremos aquellas variables con varianza muy pequeña, todo ello utilizando el transformador `VarianceThreshold`.

Por otro lado, un paso muy usual y que proporciona buenos resultados en la práctica es *estandarizar* las variables, pues hay algoritmos que implícitamente asumen que los datos están centrados en o y con varianza similar. A cada atributo le restamos la media de la columna y dividimos por la desviación típica, de forma que las variables resultantes quedan centradas en o y con varianza unidad. Esto es especialmente relevante en el caso en el que las variables no están medidas en las mismas unidades, como es el caso del problema de regresión.

Orden de las transformaciones

Como dijimos antes, encadenaremos todas estas transformaciones en un *pipeline*. Para el caso de clasificación tenemos dos opciones, siendo la primera de ellas la siguiente:

```
preproc = [
    ("selection", PCA(0.95)),
    ("standardize", StandardScaler()),
    ("poly", PolynomialFeatures(2)),
    ("var", VarianceThreshold(0.1)),
    ("standardize2", StandardScaler())]
```

Vemos que realizamos primero la selección de variables, después estandarizamos, seguimos con la transformación polinómica y finalmente acabamos con la eliminación de variables con varianza menor a 0.1 y otra estandarización (pues tras la transformación polinómica se han podido perturbar las variables). Decidimos hacer primero la selección para reducir la dimensión sobre el conjunto original, haciendo que por un lado no surjan demasiadas variables tras hacer las combinaciones cuadráticas, y por otro que estas combinaciones partan de una base que explique mejor los datos para poder encontrar relaciones entre variables más fácilmente.

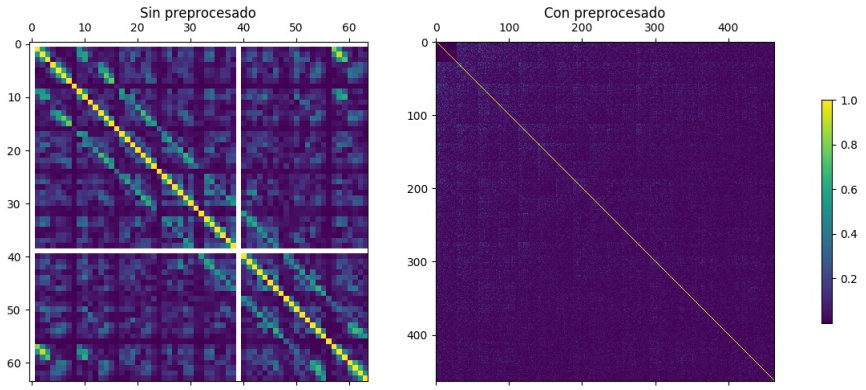
La segunda opción es análoga, sustituyendo el primer elemento del cauce por la otra estrategia de selección. Además añadimos primero un nuevo transformador `VarianceThreshold`, pues en el *F-test* hay problemas si tenemos variables constantes:

```
preproc = [
    ("var2", VarianceThreshold()),
    ("selection", SelectKBest(f_classif, k = 32)),
    ("standardize", StandardScaler()),
```

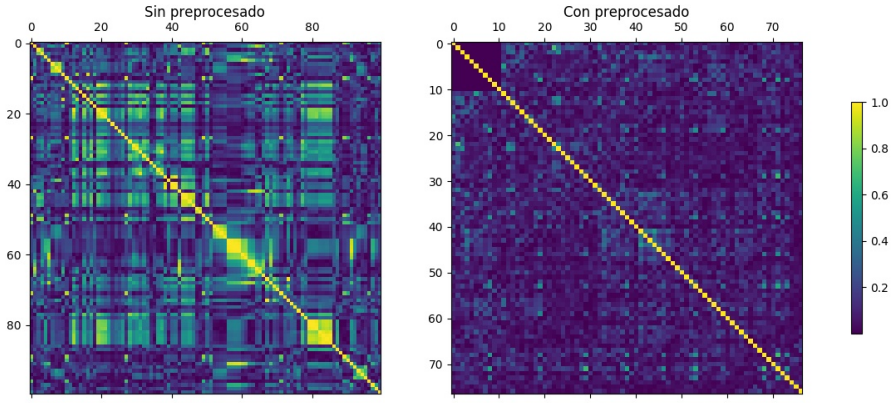
```
("poly", PolynomialFeatures(2)),
("var", VarianceThreshold(0.1)),
("standardize2", StandardScaler())]
```

Para regresión el cauce queda como en el correspondiente caso de clasificación, sustituyendo el parámetro de PCA por 0.8 y eliminando el umbral de la varianza, pues en este caso muchas de las variables tienen varianza pequeña y perderíamos demasiadas.

Mostramos finalmente en la Figura 5 un ejemplo de cómo evoluciona la matriz de correlación (en valor absoluto) en ambos problemas. Podemos apreciar que inicialmente hay muchas variables con fuerte correlación (especialmente en el caso de regresión), pero que tras el preprocesado muchas de las variables resultan casi incorreladas, haciendo que disminuya el número de variables potencialmente redundantes.



(a) Clasificación.



(b) Regresión.

Figura 5: Matriz de correlaciones absolutas en ambos problemas, antes y después del preprocesado. Las líneas blancas indican variables constantes.

Métricas de error

Fijamos a continuación las métricas que usaremos para la selección y evaluación de los modelos.

Clasificación

En el caso del problema de clasificación no hay muchas dudas; la métrica por excelencia es el *error de clasificación*, una medida simple pero efectiva del error cometido. Si denotamos los ejemplos de entrada a un clasificador h por (x_n, y_n), podemos expresar el error como

$$E_{in}(h) = \frac{1}{N}\sum_{i=1}^{N}[\![h(x_n) \neq y_n]\!].$$

Sabemos que esta medida de error se encuentra en el intervalo $[0, 1]$, siendo 0 lo mejor posible y 1 lo peor. Se trata de una medida de fácil interpretación; podemos expresarla en porcentaje o invertirla, de forma que $1 - E_{in}$ es lo que se conoce como el *accuracy* del modelo. Presentaremos los resultados utilizando esta descripción ya que parece más ilustrativa.

Regresión

En este caso la gama de opciones es más amplia, por lo que decidimos realizar una elección múltiple. Emplearemos como métrica principal el *error cuadrático medio*, y como métrica secundaria el *coeficiente de determinación* R^2.

El error cuadrático medio (MSE) cuantifica la diferencia entre las predicciones y las etiquetas reales, y puede expresarse como

$$E_{in}(h) = \frac{1}{N}\sum_{n=1}^{N}(h(x_n) - y_n)^2.$$

Además, aunque la métrica de error sea el MSE, para mostrar los resultados emplearemos su raíz cuadrada (abreviada RMSE), simplemente porque de esa forma el error estará en unidades de la variable a predecir y facilitará su interpretacón. Aunque esta es una de las métricas de error más usadas, tiene como desventaja que no está acotada y no tenemos un valor de referencia; solo podemos usarla en general para comparar los ajustes dentro de un mismo conjunto de datos, y no de manera absoluta.

Para tener una medida absoluta de error consideramos también el coeficiente de determinación R^2. Este coeficiente indica la bondad del ajuste, tomando su máximo valor en 1 (ajuste perfecto), y pudiendo tomar también valores negativos (a pesar de su nombre). Su definición es la siguiente:

$$R^2(h) = 1 - \frac{\sum_{n=1}^{N}(y_n - h(x_n))^2}{\sum_{n=1}^{N}(y_n - \bar{y})^2}, \quad \text{con } \bar{y} = \frac{1}{N}\sum_{i=1}^{N} y_n.$$

Técnicas y selección de modelos

- meter preprocesado en CV
- técnicas de ajuste
- regularización
- modelos
- grid search regresión usa neg_mse

Análisis de resultados y estimación del error

Se proporciona una medida del tiempo de ejecución de los procesos de ajustes de modelos en mi máquina (`Intel i5 7200U (4) @ 3.1GHz`).

- learning curve: escala importa; en realidad están pegados.
- Estimar Ecv y Etest.
- interpretar los errores.
- Cota hoeffding en clasificación (vale la misma)

Clasificación

- confusion matrix
- pca projections

Clasificador no lineal para comparar (no se realiza preprocesado, explicar por qué)

```
--- Clasificador no lineal (RandomForest) ---
Número de árboles: 200
Profundidad máxima: 32
Número de variables usadas: 64
Accuracy en training: 100.000%
Accuracy en test: 97.440%
Tiempo: 1.341s

Clasificador dummy para comparar

--- Clasificador aleatorio ---
Número de variables usadas: 64
Accuracy en training: 10.097%
Accuracy en test: 9.683%
Tiempo: 0.002s

------------PCA------------
```

```
Selecciona 29 variables pre-poly. Fijado a 0.95.

--- Mejor clasificador lineal ---
Parámetros:
LogisticRegression(C=1.0, class_weight=None, dual=False, fit_intercept=True,
                   intercept_scaling=1, l1_ratio=None, max_iter=500,
                   multi_class='ovr', n_jobs=None, penalty='l2',
                   random_state=None, solver='lbfgs', tol=0.0001, verbose=0,
                   warm_start=False)
Número de variables usadas: 464
Accuracy en CV: 98.718%
Accuracy en training: 100.000%
Accuracy en test: 98.831%
Tiempo: 10.962s

---- SELECT K BEST (f_test ANOVA) ------

K = 32

--- Mejor clasificador lineal ---
Parámetros:
LogisticRegression(C=1.0, class_weight=None, dual=False, fit_intercept=True,
                   intercept_scaling=1, l1_ratio=None, max_iter=500,
                   multi_class='ovr', n_jobs=None, penalty='l2',
                   random_state=None, solver='lbfgs', tol=0.0001, verbose=0,
                   warm_start=False)
Número de variables usadas: 560
Accuracy en CV: 98.561%
Accuracy en training: 100.000%
Accuracy en test: 97.551%
Tiempo: 20.625s


----- SIN SELECCIÓN DE VARIABLES -----

--- Mejor clasificador lineal ---
Parámetros:
LogisticRegression(C=10000.0, class_weight=None, dual=False, fit_intercept=Tr
                   intercept_scaling=1, l1_ratio=None, max_iter=500,
                   multi_class='ovr', n_jobs=None, penalty='l2',
                   random_state=None, solver='lbfgs', tol=0.0001, verbose=0,
                   warm_start=False)
```

```
Número de variables usadas: 1935
Accuracy en CV: 98.692%
Accuracy en training: 100.000%
Accuracy en test: 98.052%
Tiempo: 69.435s
```

Regresión

```
-------- PCA(0.8) ------------

Explicar que se ha probado sin PCA y va peor.

--- Mejor regresor lineal ---
Parámetros:
Lasso(alpha=0.000774263682681127, copy_X=True, fit_intercept=True,
      max_iter=2000, normalize=False, positive=False, precompute=False,
      random_state=2020, selection='cyclic', tol=0.0001, warm_start=False)
Número de variables usadas: 77
RMSE en CV: 0.139
RMSE en training: 0.131
R2 en training: 0.684
RMSE en test: 0.128
R2 en test: 0.702
Tiempo: 2.599s

----- SIN SELECCIÓN DE VARIABLES -----

--- Mejor regresor lineal ---
Parámetros:
Lasso(alpha=0.005994842503189409, copy_X=True, fit_intercept=True,
      max_iter=2000, normalize=False, positive=False, precompute=False,
      random_state=2020, selection='cyclic', tol=0.0001, warm_start=False)
Número de variables usadas: 5150
RMSE en CV: 0.137
RMSE en training: 0.127
R2 en training: 0.700
RMSE en test: 0.129
R2 en test: 0.698
Tiempo: 100.955s

--- Regresor no lineal (RandomForest) ---
Número de árboles: 200
Número de variables usadas: 100
```

```
RMSE en training: 0.062
R2 en training: 0.929
RMSE en test: 0.134
R2 en test: 0.673
Tiempo: 9.659s

--- Regresor aleatorio ---
Número de variables usadas: 100
RMSE en training: 0.233
R2 en training: 0.000
RMSE en test: 0.234
R2 en test: -0.001
Tiempo: 0.000s
```