APRENDIZAJE AUTOMÁTICO

PRÁCTICAS - PRÁCTICA 3

Pedro Gallego López

Doble Grado de Ingeniería Informática y Matemáticas

Universidad de Granada
Escuela Técnica Superior de Ingeniería Informática y Telecomunicaciones

4 de junio de 2021

Índice general

Anotaciones Previas

La memoria se organizará en dos capítulos y tres apéndices. Cada capítulo se corresponde con uno de los problemas y cada apéndice tiene un cometido distinto.

Los capítulos referentes a los problemas tienen una estructura idéntica. Están redactados de tal forma que se puede empezar leyendo cualquiera de los dos. Todas las explicaciones necesarias que no son el cometido de la resolución del problema como tal se encuentran en los distintos apéndices. A lo largo de los capítulos se hará referencia constantemente a los apartados que se referencian de según qué apéndice con según qué objetivo.

Los apéndices tienen objetivos distintos. En el Apéndice I se explican técnicas usadas para los problemas las cuales no han sido explicadas con anterioridad pero tampoco son técnicas que sean dependientes del problema en concreto que estamos resolviendo. Estas técnicas son PCA y t-SNE. En el Apéndice II se habla de las distintas herramientas usadas de ScikitLearn, dando las explicaciones y justificaciones que se han considerado que no eran completamente necesarias dentro del capítulo del problema.

Estos dos apéndices tienen el fin de hacer que los capítulos referentes al problema tengan una lectura mucho más limpia y corta yendo al grano del asunto.

Por último, el Apéndice III es de especial importancia ya que tiene que ver con una especie de *guía de usuario* de los códigos con algunas advertencias de tiempos, partes comentadas, etc.

Problema de Regresión

Empezaremos por el problema de regresión. Vamos a tratar con una base de datos que podemos encontrar en `https://archive.ics.uci.edu/ml/datasets/Superconductivty+Data`. Analizaremos el problema correspondiente a la base de datos y conforme a ello aplicaremos las técnicas lineales de Machine Learning oportunas.

1.1. Superconductivity Data

El conjunto de datos pretende servir para estimar un modelo que sea capaz de predecir la **temperatura crítica superconductora** basándose en las características que se pueden ver en la Tabla 1.1

Variable	**Descripción**
Masa Atómica	Masa total
Energía de primera ionización	Energía necesaria para separar un electrón en su estado fundamental de un átomo de un elemento en estado gaseoso
Radio Atómico	Radio atómico calculado
Densidad	Densidad a una temperatura y presión estándar
Afinidad del electrón	Energía requerida para añadir un electrón al átomo neutro
Calor de fusión	Energía para cambiar de sólido a líquido sin cambio de temperatura
Conductividad térmica	Conductividad térmica coeficiente k
Valencia	Número dependiente del elemento químico

Tabla 1.1: Características del problema

En la base de datos se ofrecen hasta 81 características distintas, estas características están formadas por 10 estadísticos sobre las 8 características de la Tabla 1.1 más una característica adicional que es una variable numérica que cuenta el número de elementos en el superconductor. Los 10 estadísticos son: media, media ponderada, media geométrica, media geométrica ponderada, entropía, entropía ponderada, rango, rango ponderado, desviación estándar y desviación estándar ponderada. Contamos con 21.263 instancias en nuestro conjunto de datos, 81 atributos y 1 valor real a predecir mediante regresión.

1.2. Análisis del problema

1.2.1. Comprensión del problema

Definimos nuestro problema: nuestro espacio de características será $\mathcal{X} = \mathbb{R}^{81}$. Nuestra característica a predecir es $\mathcal{Y} = \mathbb{R}$ que hace referencia a la *temperatura crítica superconductora* en grados Kelvin, y por lo tanto nuestro objetivo es construir un modelo que sea capaz de encontrar una hipótesis g que se aproxime lo máximo posible a $f : \mathcal{X} \to \mathcal{Y}$ que es la función verdadera.

1.2.2. Clase de funciones a usar $\mathcal{H}$

Como bien es sabido, tenemos que apostar por las clases más simples posibles. Así, en este conjunto de datos para predecir nuestro valor $y \in \mathcal{Y}$ no tenemos una idea muy clara de como se relacionan nuestros atributos de partida. En base a este problema, se ha decidido hacer alguna transformaciones de los datos para poder visualizar nuestros datos de *train* de manera que podamos entender, ahora mejor, la relación de nuestras características.

Para la visualización, se ha recurrido a la técnica de PCA (ver APÉNDICE I) para reducir la dimensionalidad del problema a la vez que las correlaciones entre los atributos (ahora llamadas componentes). Nos va a interesar en principio observar las componentes que mejor explican la muestra, estas son, las que mayor varianza explican de la misma: las primeras componentes. En concreto, visualizaremos la relación entre las cuatro primeras (ver Figura 1.1).

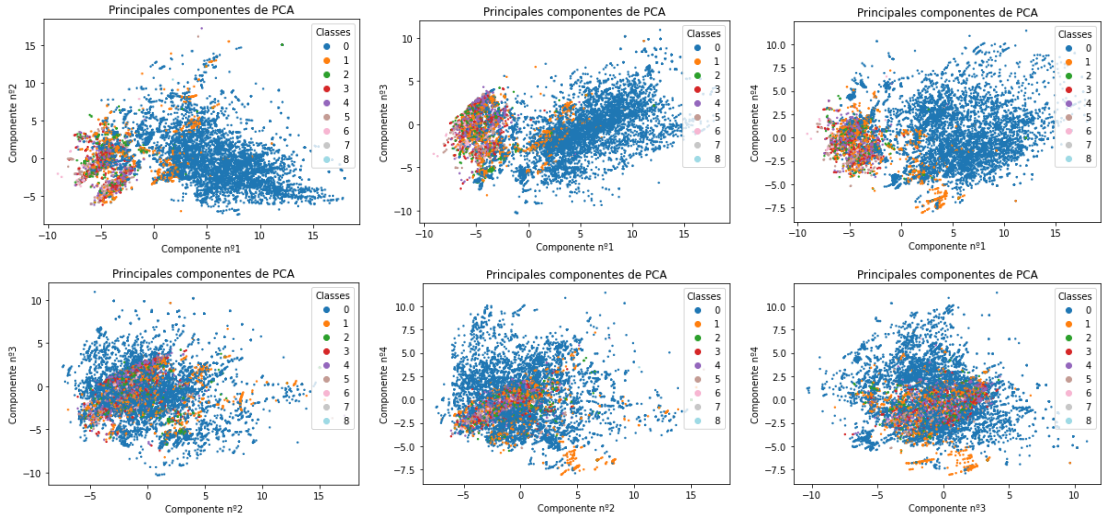


Figura 1.1: Visualización de las cuatro primeras componentes a través de PCA.
La "clase" i hace referencia a aquellas muestras con valores en $[20i,\ 20(i+1)[\in \mathcal{Y}$.

Mirando la Figura 1.1 vemos que hay una clara conclusión con la primera componente: conforme crece la componente parece tener un menor valor de *temperatura crítica superconductora* tendremos la muestra. Por otro lado las componentes 2,3 y 4 no se puede sacar mucho en claro, haciendo ver que la complejidad del problema no es tan sencilla como aparentaba ser al observar la componente principal.

En la Figura 1.2 podemos ver la evolución de la varianza explicada acumulada. Es importante observar que entre las 4 primeras componentes se explica casi un 70 % de la varianza de la muestra, y en concreto, la primera componente explica un 40 % del total. Esto significa que el observar las 4 componentes nos dan una idea de cómo se relacionan los datos con un 70 % de certeza podríamos decir.

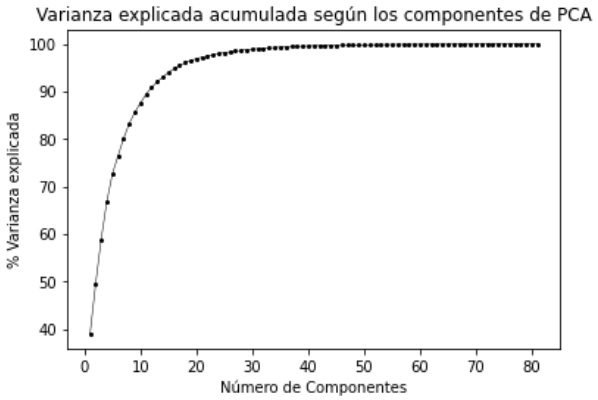


Figura 1.2: Varianza explicada acumulada con PCA
tras normalizar los datos con StandardScaler

Como conclusión, podemos decir que la componente principal es bastante representativa del valor final a predecir, es por ello que se espera que tenga un importante peso dentro de las hipótesis finales que tenga nuestro modelo.

Sin embargo, como las componentes 2,3 y 4 representan casi un 30 % de la varianza y las relaciones entre ellas no están nada claras, se ha decidido optar por un visionado de los datos a partir de t-SNE para ver si se pudiese sacar algo en claro. Lo esperado de esta representación es que agrupe a las clases que hemos definidos (valores en intervalos de tamaño 20).
NOTA: estas clases solo se han definido a la hora de hacer el plot. Los valores de y se han tomado como valores continuos.

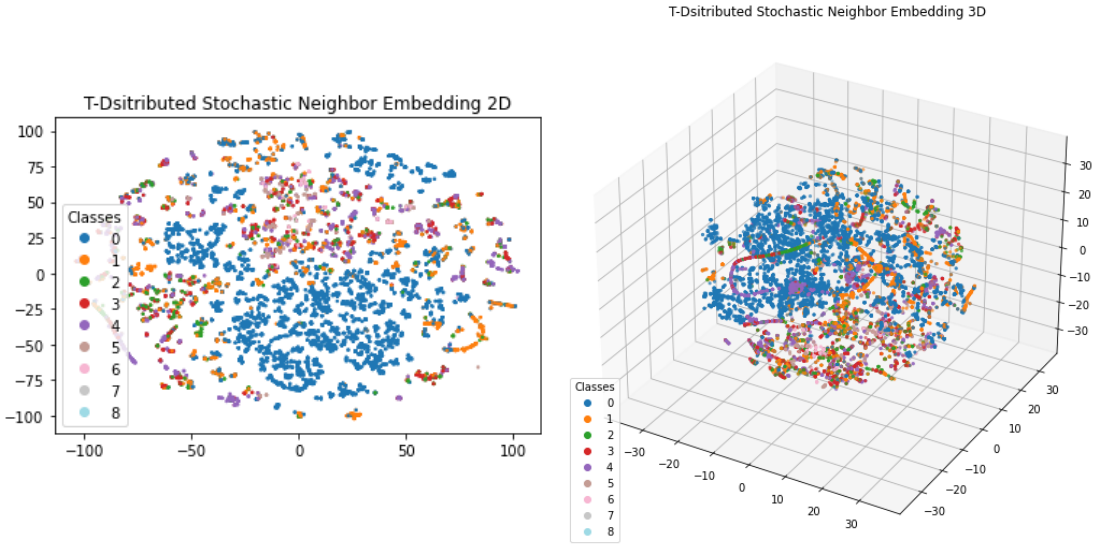


Figura 1.3: Visualización de la reducción de dimensionalidad realizada por t-SNE tras haber aplicado PCA

En la Figura 1.3 observamos el resultado de aplicar ahora t-SNE. Se ve como no hay ninguna agrupación de las clases que hemos definido, lo cual nos hace pensar una de estas dos cosas:

1. **Los atributos que tenemos realmente no son relevantes para predecir nuestro valor**. El hecho de que no estén las clases agrupadas es porque hay puntos en las clases que distan mucho de otros puntos de su misma clase y que distan menos de puntos de otras clases que no son consecuentes a ellas. Entonces lo idóneo con t-SNE es que nos saliesen las clases unas tras otra secuencialmente ordenadas aunque dejasen una separación difusa entre ellas (el valor $y_1 = 20{,}0^{\text{o}}K$ pertenece a la clase 1 y los valores $y_2 = 19{,}9^{\text{o}}K, y_3 = 0{,}1^{\text{o}}K$ pertenecen a la clase 0 y hay más distancia entre y_3 e y_2 que pertenecen a la misma clase, que entre y_2 e y_1 que pertenecen a clases distintas).

2. **La clase de funciones para hacer un buen modelo tiene que ser más compleja**. Podemos pensar que los puntos de una misma clase pueden estar separados porque la función es muy compleja.

Puesto que la primera opción no entra dentro de nuestras competencias ya que los datos son los que son, se ha decidido intentar subsanar el posible error del segundo punto. Por lo tanto, se ha decidido hacer transformaciones polinómicas de segundo orden sobre las componentes resultantes de `PCA(0.95)` para no aumentar demasiado la complejidad del modelo pero que tenga más capacidad para obtener mejores resultados.

Con el fin de no sobreajustar también se usarán métodos de regularización.

1.3. Resumen previo de las decisiones tomadas y Modelos

En esta sección se indica una vista panorámica del *modus operandi* de la resolución del problema. Las transformaciones de los datos, disminuciones de dimensionalidad y demás preprocesamiento que se incorporará al Pipeline será el siguiente:

```
steps = [("var",VarianceThreshold()),
         ("Standardize", StandardScaler()),
         ("PCA", PCA(0.95)),
         ("Second Order Pol", PolynomialFeatures(2)),
         ("var2",VarianceThreshold(0.1)),
         ("Standardize2", StandardScaler())]
```

Todo ello unido al modelo de aprendizaje.

1.3.1. Modelos

El modelo se elegirá entre los siguientes:

```
param_grid = [{'reg': [SGDRegressor(max_iter = maxiter,
                                    random_state = SEED)],
               'reg__alpha': np.logspace(-1,4,6, base=2),
               'reg__penalty': ['l2','l1'] },
              {'reg': [Ridge(max_iter = maxiter)],
               'reg__alpha': np.logspace(-1,4,6, base=2)},
              {'reg': [LinearSVR(max_iter=maxiter)]}]
```

Regresión Lineal

Aquí utilizamos varias variantes:

- `SGDRegressor`: regresión lineal usando gradiente descendente estocástico. Esta técnica se ha elegido porque regresión lineal es una técnica que evidentemente va bien con este problema de regresión. Al incluir SGD puede que consigamos alguna mejora. Las regulaciones que se han considerado para este estimador son:
  - `'l1'`: $||w||_1 = \sum_q |w_q|$
  - `'l2'`: $||w||_2 = \sum_q w_q^2$

  e irán reguladas bajo el parámetro `reg__alpha` que indicará la fuerza de la regularización. Como no tenemos muy claro su desempeño, se ha apostado por 6 valores distintos en potencias de 2 que se han creido oportunos para valorar cuál mejora más.
- `Ridge`: regresión lineal con penalización `'l2'`. Se ha elegido como representante de regresión lineal sin incluir gradiente de descenso estocástico. La fuerza de la regularización es exactamente igual que el punto anterior. El solver se dejó como *auto*.

Support Vector Regressor

`LinearSVR`: adaptación a regresión del SVM. En realidad su adaptación está en `SVR` pero se ha decidido optar por LinearSVR porque utiliza *liblinear* en lugar de *libsvm*, que está implementada para casos lineales concretamente, es decir, nuestro caso.

Esta adaptación está basada en el mismo principio que el SVM. De la misma forma que para el problema de clasificación teníamos una especie de pasillo que definía la máxima separación entre las clases, ahora tenemos un pasillo cuyo objetivo es ser lo más fino posible a la vez que contenga el máximo número de puntos posible.

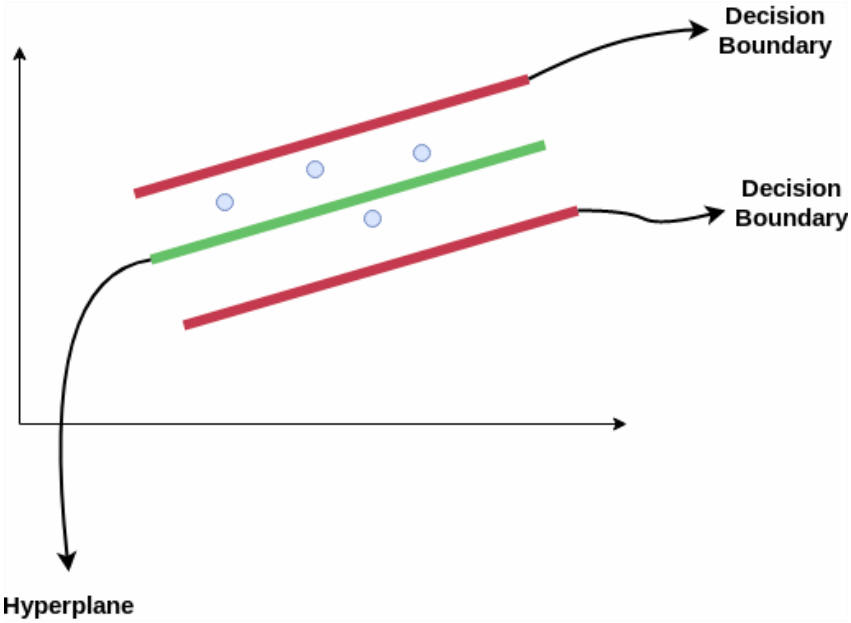


Figura 1.4: Suppor Vector Regressor. En la imagen se puede ver en ROJO la superficie/curva que delimita el pasillo, en VERDE nuestro hiperplano predictor. El objetivo es que el pasillo contenga el máximo de puntos posibles siendo todo lo estrecho que se pueda. FUENTE

Común

Se ejecutarán todos los modelos un máximo de 10.000 iteraciones. Este valor se ha puesto en base a algunos warnings de convergencia que lanzaba sklearn. Se ha optado por poner un máximo de iteraciones lo suficientemente grande como coherente para que converjan los modelos.

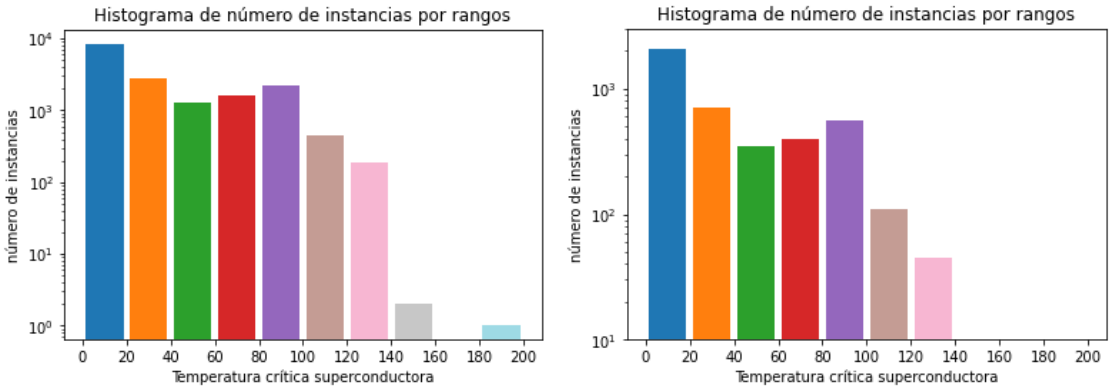
Se ejecutará cross-validation con 10 folds y se usará el error cuadrático medio como medida para elegir modelo. Además se calcularán otras métricas como el coeficiente de determinación R^2 para tener un valor que sea más fácil de analizar. Pero de todo esto se hablará más largo y tendido en posteriores apartados.

1.4. Partición de los datos: Train y Test

Puesto que la base de datos va en un único fichero, tendremos que particionar los datos nosotros mismos. Para ello se ha decidido dejar un 20 % de datos para test: una cantidad que se suele usar y que se cree conveniente en función a los 21.263 datos que tenemos: tendremos un conjunto amplio tanto de entrenamiento como de test.

Como los datos no tienen ninguna dependencia ordinal entre ellos que deba de ser útil para nuestro modelo, se ha decidido meter estocasticidad a la división, cogiendo un 20 % de datos aleatorios para nuestro conjunto test.

Por otra parte, como se hablará más adelante en esta memoria, se aplicará cross-validation 10-folds. Lo que implicará tener un conjunto de validación del tamaño del 10 % del 80 % de los datos totales, quedando el resto del 80 % para train durante la validación.



(a) Train (escala logarítmica) (b) Test (escala logarítmica)

Figura 1.5: Distribución de muestras en función de su *temperatura crítica superconductora* en los rangos propuestos: $[20i, 20(i+1)]$

Por la naturaleza de la técnica de cross-validation, estos conjuntos no son fijos durante la etapa de validación, pero sí será fijo el tamaño con el que se cogen estos conjuntos. El motivo de coger un 10 % es porque el tamaño de la muestra permite que se puedan ejecutar relativamente rápido todos los folds. De ser mayor el tamaño quizás se recurriría a 5-folds (que es lo que nos pasa con el problema de clasificación *sensorless drive diagnosis*).

1.5. Preprocesamiento de datos

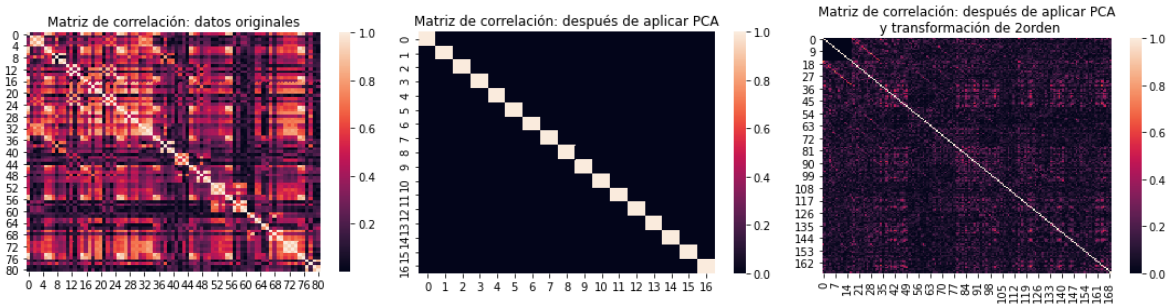
En esta sección se tratarán todas las manipulaciones sobre los datos iniciales $\mathcal{X}$ hasta fijar el conjunto de vectores de características que se usarán en el entrenamiento $\mathcal{Z}$.
Los valores de los datos son valores reales, y por lo tanto podemos tomarlos como tal sin hacer ninguna codificación distinta. Todo el preprocesamiento de los datos se guardará en un Pipeline, el cual se aplicará un `.fit_transform` sobre el conjunto de entrenamiento y únicamente un `.transform`, salvándonos de hacer *data snooping*, sobre el conjunto test.

En primer lugar, vamos a eliminar, si hay, las características que tengan el mismo valor en todas las muestras. Para ello aplicaremos Variance Threshold con un umbral igual a cero. No tenemos datos perdidos así que no tenemos que preocuparnos por esa parte. En cuanto a los outliers no vamos a tratarlos puesto que no sabemos a priori en este problema si los valores que son "extremadamente distintos" son valores que reales que enriquecen al modelo o son outliers.

Una vez nos hemos tratado los valores constantes y los valores perdidos, vemos que tenemos 80 características formadas a partir de información común (8 características -Tabla 1.1-). Podemos pensar que pueden estar correladas por este motivo. Así se ha decidido representar una matriz de correlación que podemos verla en la Gráfica 1.6a, y efectivamente, están bastante correladas. Este motivo unido al gran número de características hace que parezca conveniente realizar una disminución de dimensionalidad aplicando PCA (ver Apéndice I) para tanto reducir el número de características como para eliminar correlaciones.

Previo a aplicar PCA es necesario normalizar los datos así, que añadiremos a nuestro Pipeline StandardScaler. Ahora sí podemos aplicar PCA(0.95), con un 95 % de explicación de la varianza (ver Apéndice II para saber por qué 0.95).

En este punto[1], podemos ver cómo se han reducido las características a 17 y que están totalmente incorreladas en la Figura 1.6b. Ahora, como bien hemos deducido en el análisis del problema, se ha visto conveniente utilizar una transformación polinómica de segundo orden a los datos. Esta transformación se hace a través de `PolynomialFeatures` de Scikit Learn.



(a) Originales (b) PCA(0.95) (c) PCA(0.95), PolFeat(2)

Figura 1.6: Representación de la matriz de correlación en valor absoluto durante el preprocesado. La matriz de correlación nos indica cómo de correlada están las variables entre sí dando lugar a una matriz simétrica. El valor absoluto es simplemente para captar visualmente de forma sencilla si una variable está correlada (directa o inversamente) con otra: estará menos correlada cuanto más oscuro sea el "píxel" o "celda" que les corresponde al par de variables.

Aprovechamos este aumento de dimensionalidad para poder quitarnos variables por una varianza baja. Aplicamos ahora un umbral de varianza con `VarianceThreshold(0.1)`. Y por último volvemos a normalizar los datos con `textStandardScaler`.

Tras todas estas transformaciones podemos ver en la Figura 1.6c como queda de correlado nuestro nuevo espacio muestral $\mathcal{Z}$. A la vista está que tenemos ahora un espacio poco correlado y con una amplia gama de características que pueden sernos útiles.

1.6. Métricas

Métrica de error

Se ha usado la métrica de error cuadrático medio **MSE**. El error cuadrático medio de un estimador mide el promedio de los errores al cuadrado, es decir, la diferencia entre el estimador y lo que se estima.

$$MSE = \frac{1}{n}\sum_{i=1}^{n}(h(x_i) - y_i)^2 \tag{1.1}$$

donde h es el estimador.

El MSE es el segundo momento (sobre el origen) del error, y por lo tanto incorpora tanto la varianza del estimador así como su sesgo. Para un estimador insesgado, el MSE es la varianza del estimador. Al igual que la varianza, el MSE tiene las mismas unidades de medida que el cuadrado de la cantidad que se estima. En una analogía con la desviación estándar, tomando la raíz cuadrada del MSE produce el error de la raíz cuadrada de la media o la desviación de

[1]Una vez hayamos aplicado PCA, podremos visualizar los datos mediante t-SNE como bien se ha hablado en la sección anterior. Además, no se ha dicho, pero se recomienda en la documentación de scikit learn aplicar métodos de disminución de características para problemas de gran dimensionalidad antes de aplicar t-SNE.

la raíz cuadrada media (RMSE), que tiene las mismas unidades que la cantidad que se estima; para un estimador insesgado, el RMSE es la raíz cuadrada de la varianza, conocida como la desviación estándar.

El interés de esta métrica reside tanto en que la naturaleza del problema implica minimizar la distancia entre el estimador y lo estimado. Al final queremos predecir unos valores continuos, que serán mejores cuanto más próximos estén de su valor real, es decir, cuanto menor sea la distancia. De la misma manera, esta métrica permite tener unos resultados de fácil análisis tomando su raíz cuadrada, es decir, la desviación RMSE.

$$RMSE = \sqrt{MSE} = \sqrt{\frac{1}{n}\sum_{i=1}^{n}(h(x_i) - y_i)^2} \tag{1.2}$$

En nuestro caso, como hay que darle un *'score'* al entrenamiento nosotros pondremos como *score=-MSE*. Hay que multiplicarlo por -1 puesto que SKlearn toma los score como que cuanto mayores son, mejor puntuación es, cosa que en MSE pasa al revés: cuanto mayor es mayor distancia hay entre la estimación y lo estimado y por lo tanto peor es.

Otras métricas

Aquí hablaremos de métricas que se han usado para medir la calidad de la solución:

1. Como ya se ha comentado se usará RMSE para valorar resultados (no para guiar el entrenamiento).

2. Otra métrica que se va a usar es el coeficiente de determinación lineal R^2. El coeficiente de determinación, denominado R^2 es un estadístico usado en el contexto de un modelo estadístico cuyo principal propósito es predecir futuros resultados o probar una hipótesis. El coeficiente determina la calidad del modelo para replicar los resultados, y la proporción de variación de los resultados que puede explicarse por el modelo.
   Si representamos por σ^2 a la varianza y por σ_r^2 a la varianza residual, se tiene que:

   $$R^2 = 1 - \frac{\sigma_r^2}{\sigma^2} \tag{1.3}$$

   El interés de este estadístico reside en que permite ver la proporción de varianza explicada por el estimador.

1.7. Regularización

Como penalizaciones se han considerado para regresión lineal, como bien se ha explicado en el apartado de Modelos, las regularizaciones:

- `'l1'`: $||w||_1 = \sum_q |w_q|$
- `'l2'`: $||w||_2^2 = \sum_q w_q^2$

quedándose nuestro error como:

$$E_{aug} = E_{in} + \alpha\Omega(h) \tag{1.4}$$

donde $\Omega(h)$ es nuestra penalización. Las razones de estas penalizaciones son las siguientes:

- la regularización `'l2'` (*weight decay*) es una regularización muy recomendada siempre porque su propósito es disminuir los valores de todos los parámetros uniformemente, haciendo que en general los pesos tomen valores bajos. Esto es conveniente porque queremos

evitar sobreajustarnos: hay presencia de ruido determinístico y estocástico. El ruido suele tener frecuencias muy altas así que llevando los pesos a valores bajos conseguimos suavizar nuestra regresión obteniendo frecuencias bajas y alejándonos del ruido.

- la regularización `'l1'`. Esta regularización solo se le ha añadido al modelo con descenso de gradiente estocástico. Esta regularización lleva pesos a 0 y hace que sea un buen método para seleccionar características. Las características que tenemos se han creído suficientes y buenas, así que el hecho de poner solo esta regularización a regresión lineal con SGD para confirmar que no hay un exceso de características inútiles en nuestro problema.

Para el support vector machine de regresión no se ha impuesto ninguna penalización más de la que tiene por sí mismo que es "hacer el pasillo pequeño" (inversamente a lo que se hacía en clasificación, que era ensanchar el pasillo) ver Figura 1.4

1.8. Cross-Validation. Selección de la mejor hipótesis

Pasamos ahora a ejecutar técnicas de validación. Usaremos Cross-Validation con 10-folds puesto que el tamaño de datos del problema que se nos da así nos lo permite.

Usaremos la función `GridSearchCV` que viene bien explicada con la elección de los parámetros y demás en el Apéndice II. Básicamente en el Apéndice II se habla de la funcionalidad de la misma función, el por qué de los parámetros que hemos usado y demás cosas relacionadas.

La razón de usar esta técnica viene porque es el mejor *modus operandi* para elegir un modelo. Al final tenemos varias opciones de estimadores que cada uno de ellos conlleva varias opciones de parámetros. Es una forma en la que el error de validación se hace bastante representativo y se podría tomar como una estimación de lo que sería el error fuera de la muestra.

La mejor hipótesis escogida por nuestra validación cruzada ha sido referente al modelo de Ridge: regresión lineal con weight decay. En concreto tenemos la siguiente selección de parámetros:

```
{'reg': Ridge(alpha=16.0, max_iter=10000), 'reg__alpha': 16.0}
```

Su error en validación ha sido $E_{CV} = 289{,}14$ (ERM). Esto es equivalente a un $17.004^{\text{o}}K$(RMSE), un error que lejos de ser el peor, no es tampoco un resultado a priori excelente.

Del mejor estimador elegido por CV cabe destacar el alto valor del parámetro α que imprime una mayor o menor regularización en función de su valor. En este caso $\alpha = 16$ es un valor bastante alto, concretamente es el valor más alto que le habíamos asignado al α en el grid de parámetros.

Esto es en parte por la gran cantidad de características que tenemos. También nos puede indicar que hay muchas características que no son útiles, o si lo son, lo son en poca medida, ya que habrá muchos pesos muy cercanos a 0. Esto nos hace pensar en que la regularización `'l1'` de regresión lineal con SGD no se habrá tenido que quedar muy alejada (en cuanto a score) de este modelo, puesto que como se ha comentado anteriormente, su principal virtud es la selección de características.

Para ver el error de validación se han generado las curvas de aprendizaje que podemos ver en la Figura 4.2. El Score se ha transformado: como dijimos en su momento se usó el score$=-ERM$, pero para representar los datos se ha decidido usar $RMSE = \sqrt{ERM}$ ya que tiene una interpretación más amigable.

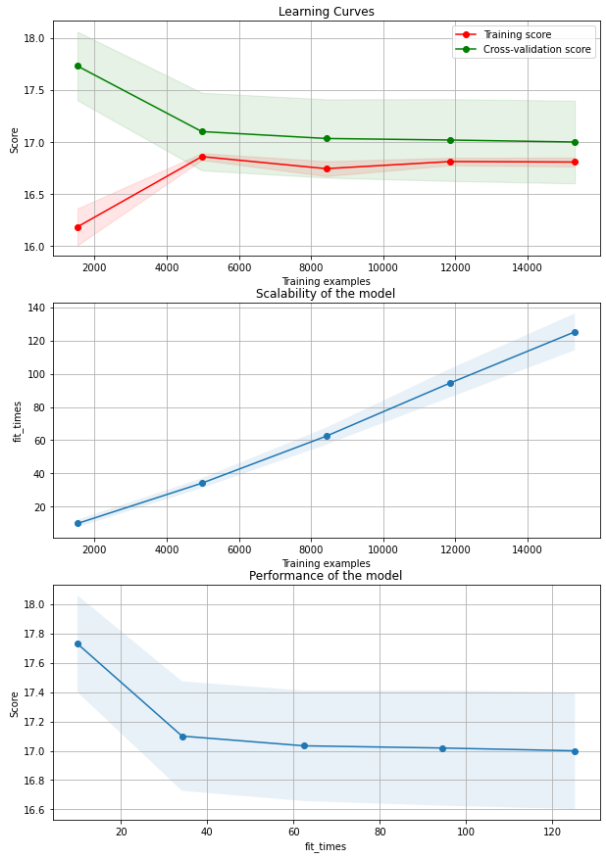


Figura 1.7: Estas gráficas se han generado a través de `learning_curve`

En la primera gráfica podemos ver la curva de aprendizaje, en donde se muestra el desempeño del modelo en los conjuntos de validación y entrenamiento. Como cabe esperar, con pocos ejemplos de entrenamiento, se obtienen Errores de validación más altos. Se observa como conforme crece el número de muestras se tiende a un Score de 17 en ambos score ($E_{train} \uparrow$, $E_{test} \downarrow$).

La estela verde y roja hacen referencia a los máximos y mínimos que se han obtenido durante cada entrenamiento con cada número de ejemplos, es decir, la desviación.

En la segunda gráfica se ve el número de entrenamientos necesarios en función de las muestras de entrenamiento. Por último, en la tercera gráfica se ve el desempeño del modelo en función del número de entrenamientos, que vemos que converge también a un número cercano a 17 para el $RMSE$.

1.9. Entrenamiento de la mejor hipótesis en el conjunto de entrenamiento completo

En nuestro caso, al usar `GridSearchCV` con el parámetro `refit` en True, él mismo se encarga de rentrenar el mejor modelo escogido por cross-validation con el conjunto completo de datos. De esta forma, nuestra hipótesis se refina con más información y pudiendo dar una mejor estimación del error fuera de la muestra. Nuestra estimación de E_{out} la haremos a través de E_{test}.

Así, al ejecutar la hipótesis final sobre los datos de entrenamiento y de test se han obtenido los siguientes resultados:

	TRAIN	**TEST**
`RMSE`	16.819	17.030
R^2	0.759	0.750

Tabla 1.2: Resultados del mejor modelo en los conjuntos de Train y Test

Recordamos que el mejor modelo ha sido

```
{'reg': Ridge(alpha=16.0, max_iter=10000), 'reg__alpha': 16.0}
```

A modo de comprobación, se han impreso los pesos de cada característica en la Tabla 1.3 y se ha hecho un pequeño análisis. Puesto que la tabla es muy engorrosa se ha puesto al final.

Ahora pasamos a ver algunas gráficas que muestran la bondad del modelo.

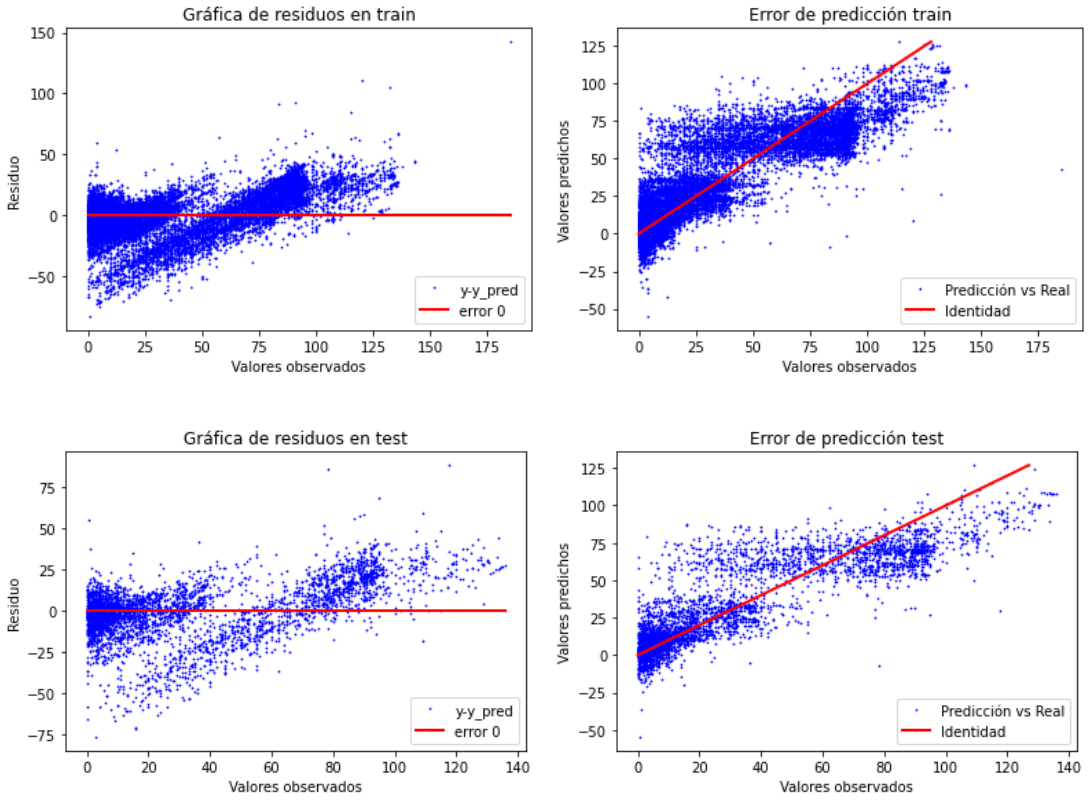


Figura 1.8: Gráficas que muestran la bondad de la hipótesis final. Las gráficas de la izquierda vemos los residuos. Muestra la diferencia entre todos los $h(x_i) - y_i$. Por otro lado en la gráfica de la derecha vemos la misma información pero con otro enfoque: comparamos el valor estimado frente al real. Lo ideal sería que en las gráficas de la izquierda los puntos estuviesen en la recta $ejeY = 0$ definida por la línea roja y que en las gráficas de la derecha los puntos se acumulasen en torno a la recta identidad marcada en rojo, que implicaría que $h(x_i) \approx y_i$

En base a las Gráficas 1.8 vemos que la estimación sigue un poco la tendencia de lo que sería el estimador ideal, pero dejando unos errores considerables. No hace falta más que mirar la Tabla 1.2 para darnos cuenta de esto:

Nos fijamos en la métrica R^2 que nos da un error relativo que hace que el valor sea independiente de las medidas del problema y sea un buen comparador general. Tenemos valores en torno al 0.750 un valor que nos dice que no es el mejor estimador pero que no hace una mala función. Ahora nos fijamos en el RMSE en donde nos dicen que los valores se distan aproximadamente en 17^{o}K, lo cual es bastante si lo pensamos: al principio hemos separado las clases (para pintarlas) en rangos de 20 porque era una distinción que se ha considerado sustancial, y ahora obtenemos un error RMSE cercano a 20. Cabe destacar que en las gráficas podemos ver valores que perfectamente podrían ser outliers, y estos también aportan a este error.

Como conclusión se ha obtenido un estimador que no tiene un mal desempeño como tal, pero que se ha considerado entre varias opciones dentro de modelos lineales (a pesar de las transformaciones) y ha sido la hipótesis ganadora. Sería interesante estudiar el problema con modelos no lineales como Redes Neuronales entre otros que tienen un mayor potencial, pero eso

queda fuera de esta práctica.

Pesos asignados

En la Tabla 1.3 podemos ver como los pesos en general son bajos. Los de las características correspondientes a segundo orden (como teníamos 17 componentes principales, tenemos que a partir de la 18+1, más uno por el bias, tenemos las características cuadráticas) que se encuentran a partir de la fila 7, tienen peso muy muy bajos su gran mayoría, esto es el efecto del gran valor del parámetro α que obtuvimos en cross-validation. Por lo general las características de primer orden actúan la mayoría de forma muy presencial con pesos no muy bajos en el ajuste.

FILAS-COLUMNAS	1	2	3	4
1	-3.09234958e+01	6.56434987e+00	-4.01150840e-01	-1.14715935e+01
2	1.76403891e+01	1.09703931e+01	6.07740565e+00	1.23615322e+00
3	5.10615940e+00	1.59752672e+00	3.50337033e+00	-1.44239814e+00
4	7.42106881e-01	2.05549204e+00	4.93304077e+00	1.35647437e+00
5	2.73005955e+00	1.72821718e+01	-4.91383135e+00	-4.00063589e+00
6	7.54371301e+00	-1.57459132e+01	-9.05214812e+00	-4.87849213e+00
7	6.81327234e-01	-3.73205429e+00	-9.07090131e-01	-3.71040890e+00
8	2.64475647e+00	-1.55705158e+00	-1.34621259e+00	-4.60318508e+00
9	-1.88848486e+00	-3.70290853e+00	5.42482658e-01	3.61050726e+00
10	6.86035973e-01	-1.81640901e-01	-7.07692174e-01	-1.05356395e+00
11	-7.92249756e-01	-8.24471105e-01	-3.73493121e-01	-9.44217982e-01
12	-1.11447587e+00	1.91172658e+00	4.10020487e-01	-2.16360058e-01
13	1.43747458e+00	-4.80003870e-01	-7.50134110e-01	-2.81884135e+00
14	8.36933710e+00	1.87480529e+00	1.18794400e+00	-1.24805781e+00
15	1.68807048e+00	1.53390448e+00	1.16093731e+00	-1.92031084e+00
16	1.00056373e-01	9.84045371e-01	-9.97103513e-01	4.39345200e-01
17	1.23758222e+00	-1.80152612e+00	2.91375823e+00	-2.34694814e-01
18	-5.68902720e-01	-4.35786427e-01	1.02088955e+00	2.07977668e+00
19	4.40758530e-01	-1.55851329e+00	1.44936215e-01	-3.80344240e-01
20	-7.43543257e-02	-1.37323698e+00	2.69028651e-01	5.20720265e+00
21	5.52153123e-01	1.33994302e+00	-1.98791588e+00	-1.53105910e+00
22	-1.10301741e+00	5.62953646e-01	-2.52177749e-01	-3.08626018e-02
23	4.86288063e-01	-1.60997531e+00	2.01981848e+00	1.47343990e+00
24	-8.73131451e-01	1.84576545e+00	-3.26590492e-01	-2.68318655e+00
25	1.05643538e+00	1.08693310e-01	1.36446814e+00	7.44027710e-01
26	2.35795459e-01	-1.93957819e-01	1.21320491e+00	-4.08891275e-02
27	-1.69961380e+00	-1.04939599e+00	1.91769023e-01	-1.40421285e+00
28	7.78086146e-01	2.14502022e-01	5.62989043e-02	1.57737243e+00
29	1.01512690e+00	-3.94297448e-01	-6.33305294e-02	-3.81579968e-03
30	-6.36547865e-02	1.41484014e+00	-1.72435845e+00	1.77054504e+00
31	-4.89615803e-01	-8.34200996e-01	1.50132081e+00	1.15012688e+00
32	-1.27690358e+00	6.67445566e-01	-2.40917297e+00	-2.95191316e-01
33	1.07101451e+00	-9.14944509e-01	-3.06157220e-01	1.93051504e+00
34	7.42938657e-01	-1.95567859e+00	-1.17111607e+00	4.95475221e-01
35	-1.13276784e+00	3.36460474e-02	2.15643649e+00	-1.34602140e+00
36	4.39859509e-01	-4.44472368e-01	9.80891646e-02	-8.56802541e-02
37	1.17065614e-01	-1.35738467e-01	2.65524924e-02	6.60760681e-01
38	5.51533650e-01	1.16992980e+00	1.83275636e-01	5.19043834e-01
39	6.92407613e-01	-7.78768168e-01	-3.69702040e-01	-5.41761363e-01
40	2.03267095e-02	2.58948063e-01	7.30484214e-01	9.74164536e-02
41	6.82790730e-01	8.19726609e-02	1.55251168e+00	-1.87245738e-01
42	8.97616210e-01	-3.29802376e-01	-6.20077393e-02	8.17996247e-02
43	-2.21961261e-02	2.50450906e-01		

Tabla 1.3: Pesos del mejor modelo

Problema de Clasificación

Continuamos ahora con el problema de clasificación. En esta ocasión se va a tratar con una base de datos que se puede encontrar en la url `https://archive.ics.uci.edu/ml/datasets/Dataset+for+Sensorless+Drive+Diagnosis`. Seguiremos el mismo procedimiento que con el problema de regresión: analizaremos el problema correspondiente a la base de datos y conforme al análisis se aplicarán las técnicas lineales de Machine Learning que se crean oportunas.

2.1. Sensorless Drive Diagnosis Data Set

El conjunto de datos pretende servir para estimar un modelo que sea capaz de clasificar motores en función de cómo de defectuosos sean sus componentes.

Las características se extraen de las señales de conducción de corriente eléctrica. La unidad tiene componentes intactos y defectuosos. Esto da como resultado 11 clases diferentes con diferentes condiciones. Cada condición se ha medido varias veces mediante 12 condiciones de funcionamiento diferentes, es decir, mediante diferentes velocidades, momentos de carga y fuerzas de carga. Las señales de corriente se miden con una sonda de corriente y un osciloscopio en dos fases.

Se utilizó la descomposición empírica en modos (EMD) para generar una nueva base de datos para la generación de características. La descomposición empírica en modos (EMD) es una técnica de multirresolución adaptativa de datos para descomponer una señal en componentes físicamente significativos. La EMD se puede utilizar para analizar señales no lineales y no estacionarias dividiéndolas en componentes con diferentes resoluciones. Los componentes de EMD se conocen como funciones de modo intrínseco (IMF).

Se utilizaron las tres primeras funciones de modo intrínseco (IMF) de las corrientes de dos fases y sus residuos (RES) y se dividieron en subsecuencias. Para cada una de estas subsecuencias, se calcularon la media de las características estadísticas, la desviación estándar, la asimetría y la curtosis.

Como resultado nuestra base de datos consta de 48 atributos: 4 estadísticos $\times$ 12 condiciones de funcionamiento diferentes. El atributo número 49 de nuestra base de datos es el atributo a predecir, es un valor natural entre el 1 y el 11 que determina una clase en función del estado de los componentes: intacto $\longrightarrow$ defectuoso. La base de datos consta de 58.509 instancias.

2.2. Análisis del problema

2.2.1. Comprensión del problema

Definimos ahora nuestro problema: el espacio de características es $\mathcal{X} = \mathbb{R}^{48}$. Nuestro atributo de clase para predecir va a ser $\mathcal{Y} = \{1, 2, ..., 11\}$ y nuestro objetivo es conseguir un modelo que sea capaz de obtener una hipótesis h que se aproxime lo máximo posible a $f : \mathcal{X} \to \mathcal{Y}$, que es nuestra función etiquetadora verdadera.

2.2.2. Clase de funciones a usar $\mathcal{H}$

Sabemos que un modelo simple siempre es una opción por la que apostar. Como no sabemos bien la relación que existe entre las 48 características para dar el valor $y \in \mathcal{Y}$, es conveniente intentar visualizar los datos de entrenamiento previamente para ver si podemos entender mejor el problema. Para poder visualizarlos se va a recurrir primeramente a una reducción de dimensionalidad a través de PCA, y sobre estas componentes se procederá al visionado mediante t-SNE (ver Apéndice I). Tras aplicar PCA y quedarnos con 20 componentes (que simbolizan el 95 % de explicación de la varianza), procedemos a aplicar t-SNE sobre estas componentes.

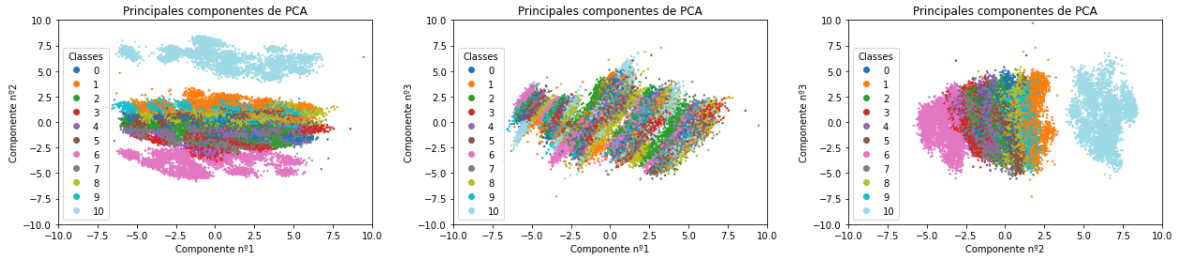


Figura 2.1: 1vs1 entre las tres primeras componentes arrojadas por PCA

Mirando la Gráfica 2.1 se observa a primera vista la poca relación que guardan las componentes 1 y 3. Sin embargo la componente 2 para relacionarse de buena forma tanto con la 1 como con la 3, en donde se puede intuir que la clase 10 y la clase 6 son bastante dependientes de la segunda componente.

Se han graficado estas 3 componentes porque, como se puede ver en la Gráfica 2.2, las componentes explican en su mayoría poca varianza, siendo las 3 primeras las que más explican, explicando entre las tres más de un 40 %. Nos ha servido para ver que hay alguna dependencia que si no es lineal está muy cerca de serlo, como hemos observado en la componente 2.

Para ver si hay más dependencias claras vamos a proceder ahora a la visualización de los gráficos arrojados por t-SNE.

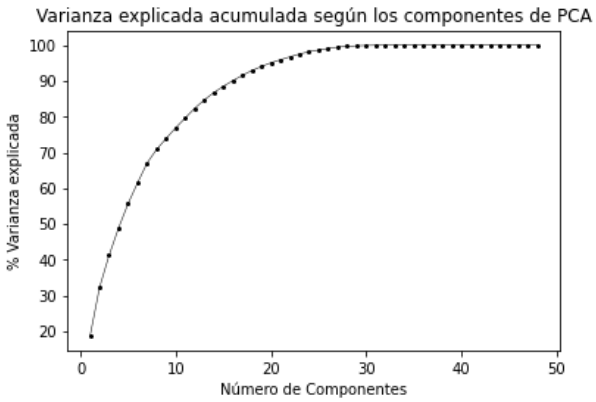


Figura 2.2: Varianza explicada acumulada con PCA tras normalizar los datos con StandardScaler

Observamos ahora las gráficas de la Gráfica 2.3. Las gráficas son prácticamente imposibles de interpretar para sacar conclusiones certeras. La distribución de los puntos de cada clase es

muy compleja, por lo que esto nos va a hacer decantarnos por hacer una ampliación de características polinómica de segundo orden sobre las componentes resultantes de `PCA(0.95)` para no aumentar demasiado la complejidad del modelo pero que tenga más capacidad para obtener mejores resultados. Se cree suficiente además que sean de segundo orden, porque hemos visto con las componentes de PCA de que hay aparentes relaciones lineales, lo que nos dice que las relaciones no tendrán que ser muy complejas.

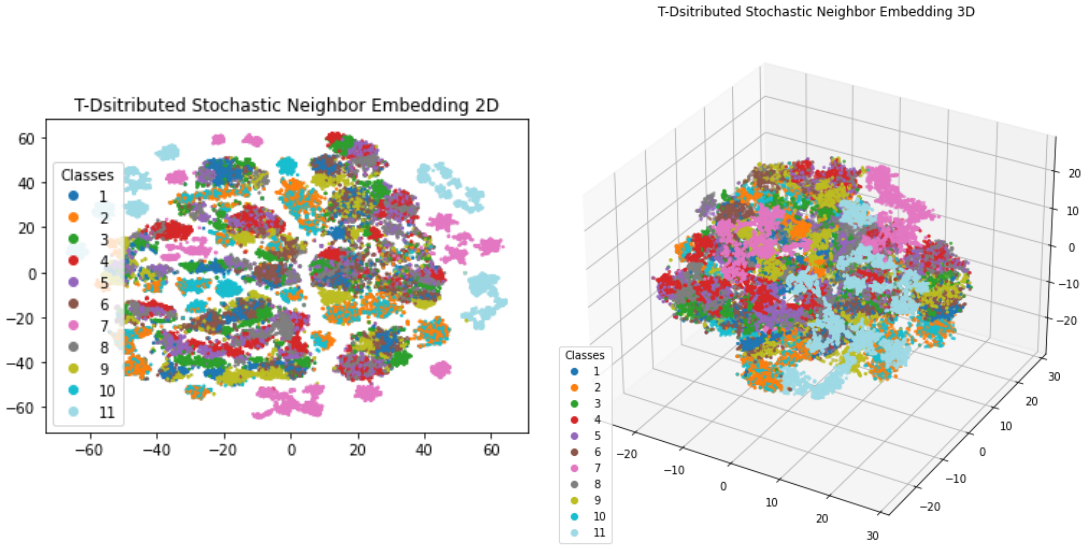


Figura 2.3: t-SNE tras aplicar PCA

2.3. Resumen previo de las decisiones tomadas y Modelos

En esta sección se indica una vista panorámica del *modus operandi* de la resolución del problema. Las transformaciones de los datos, disminuciones de dimensionalidad y demás preprocesamiento que se incorporará al Pipeline unido al modelo de aprendizaje será el siguiente:

```
steps = [("var",VarianceThreshold()),
         ("Standardize", StandardScaler()),
         ("PCA", PCA(0.95)),
         ("Second Order Pol", PolynomialFeatures(2)),
         ("var2",VarianceThreshold(0.1)),
         ("Standardize2", StandardScaler())]
```

2.3.1. Modelos

El modelo se elegirá entre los siguientes:

```
param_grid = [{'clf': [LogisticRegression(max_iter = maxiter,
                                          multi_class = 'ovr',
                                          penalty = 'l2')],
               'clf__alpha': np.logspace(-1,4,3, base=2),
              {'clf': [RidgeClassifier(random_state = SEED,
                                       max_iter = maxiter)],
               'clf__alpha': np.logspace(-1,4,3, base=2)},
              {'clf': [Perceptron(random_state = SEED,
                                  penalty = 'l2',
                                  max_iter = maxiter)],
               'clf__alpha': np.logspace(-1,4,3, base=2)}]
```

Regresión Logística

Regresión logística es un método de clasificación lineal. Es un método de clasificación estudiado que se encarga de asignar una probabilidad de que se pertenezca a una clase o no. Se ha utilizado la técnica *One versus Rest*. OvO no se ha utilizado porque se ha considerado que es muy costoso computacionalmente por el número de modelos que se tienen que aprender. El solver utilizado es el por defecto, lbfgs, que hace un buen desempeño.

Clasificador Ridge

Es regresión lineal con penalización `'l2'` usada como clasificador. Lo primero que hace este clasificador es convertir las etiquetas en {-1,1} y posteriormente trata de resolver el problema de regresión. La estrategia consiste en entrenar un modelo por clase, que a fin de cuentas es una estrategia *One versus Rest*. Se ha decidido utilizar este clasificador porque ya se vió en prácticas anteriores que un regresor lineal puede hacer bien el trabajo de un clasificador. El solver se ha dejado como auto.

Perceptron

Por último se ha optado por el perceptrón, el cual es el algoritmo más básico de todos pero que no tiene por qué hacer un mal desempeño. En principio hemos dejado todos los parámetros por defecto, añadiendo la semilla, y una penalización `'l2'`.

Común

Se ha puesto el número de iteraciones a 1000. La regresión logística tarda mucho en ejecutarse, y no se ha creído conveniente poner más. Los parámetros multiplicadores de la penalización que regulan la cantidad de regularización el modelo se abordarán en el apartado de regularización.

Se ejecutará cross-validation 5 fols y se usará el *Accuracy* como score para valorar que modelo es mejor.

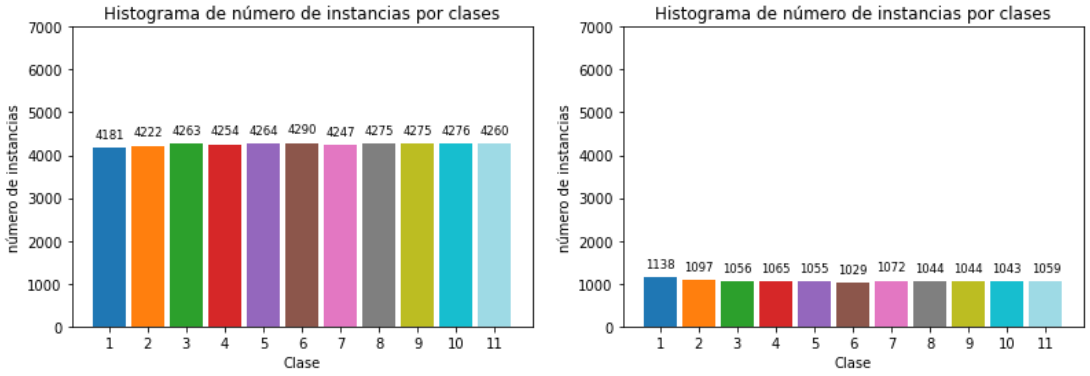
2.4. Partición de los datos: Train y Test

Puesto que la base de datos va en un único fichero, tendremos que particionar los datos nosotros mismos. Para ello se ha decidido dejar un 20 % de datos para test: una cantidad que se suele usar y que se cree conveniente en función a los 58.509 datos que tenemos: tendremos

un conjunto amplio tanto de entrenamiento como de test.

Los datos vienen ordenados según la clase, así que se hará una división aleatoria, cogiendo el 20 % de aleatorios para nuestro conjunto test.

Por otra parte, como se hablará más adelante en esta memoria, se aplicará cross-validation 5-folds. Lo que implicará tener un conjunto de validación del tamaño del 20 % del 80 % de los datos totales, quedando el resto del 80 % para train durante la validación.



(a) Train (b) Test

Figura 2.4: Distribución de las muestras en función d la clase

Por la naturaleza de la técnica de cross-validation, estos conjuntos no son fijos durante la etapa de validación, pero sí será fijo el tamaño con el que se cogen estos conjuntos. El motivo de coger un 20 % es porque el tamaño de la muestra es muy grande y conlleva mucho coste computacional subir el número de folds. De ser menor el tamaño quizás se recurriría a 10-folds (que es lo que nos pasa con el problema de regresión *Superconductivity*).

2.5. Preprocesamiento de datos

En esta sección se tratarán todas las manipulaciones sobre los datos iniciales $\mathcal{X}$ hasta fijar el conjunto de vectores de características que se usarán en el entrenamiento $\mathcal{Z}$.

Los valores de los datos son valores reales, y por lo tanto podemos tomarlos como tal sin hacer ninguna codificación distinta. Todo el preprocesamiento de los datos se guardará en un Pipeline, el cual se aplicará un `.fit_transform` sobre el conjunto de entrenamiento y únicamente un `.transform`, salvándonos de hacer data snooping, sobre el conjunto test.

En primer lugar, vamos a eliminar, si hay, las características que tengan el mismo valor en todas las muestras. Para ello aplicaremos Variance Threshold con un umbral igual a cero. No tenemos datos perdidos así que no tenemos que preocuparnos por esa parte. En cuanto a los outliers no vamos a tratarlos puesto que no sabemos a priori en este problema si los valores que son "extremadamente distintos" son valores que reales que enriquecen al modelo o son outliers.

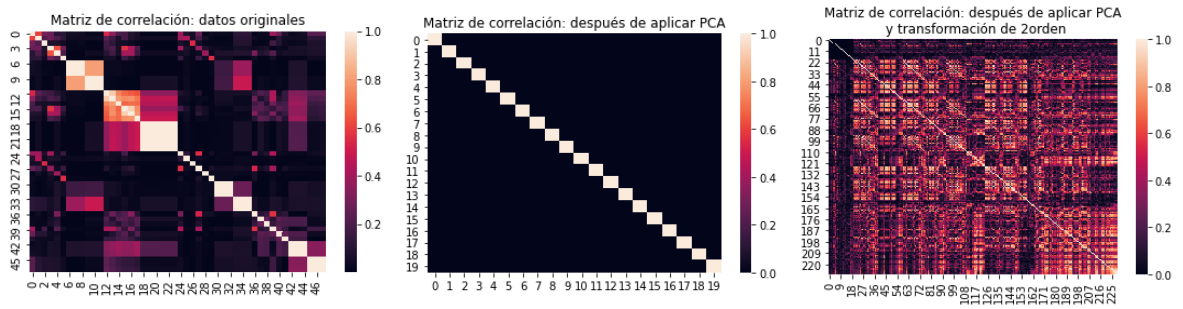
Una vez hemos tratado tanto los valores constantes como los valores perdidos, recordamos que tenemos 48 características formadas a partir de 4 estadísticos sobre 12 condiciones de funcionamiento diferentes. Se ha decidido representar una matriz de correlación para ver cómo de correladas están estas características (ver la Gráfica 2.5a). Si hacemos un esfuerzo por entender la matriz, recordamos que tenemos 12 condiciones distintas, y que sobre ellas se han aplicado

4 estadísticos. Los datos aparantemente están ordenados de forma que: primero va el primer estadístico sobre las 12 condiciones, después el segundo estadístico... Es más, si nos fijamos en la Gráfica 2.5a vemos como los "cuadrados de colores parecidos" tienen un tamaño de 12×12. Tomando esto de partida, nos fijamos que lo que sería el segundo estadístico tiene una correlación muy alta sobre las 12 condiciones. Pero es que además, parece haber una alta correlación entre las condiciones de funcionamiento {6,7,8} y además también entre las condiciones {9,10,11}.

Con el fin de eliminar correlaciones posibles y hacer una disminución de dimensionalidad se ha decidido aplicar PCA (ver Apéndice I).

Previo a aplicar PCA es necesario normalizar los datos así, que añadiremos a nuestro Pipeline StandardScaler. Ahora sí podemos aplicar PCA(0.95), con un 95 % de explicación de la varianza (ver Apéndice II para saber por qué 0.95).

En este punto[1], podemos ver cómo se han reducido las características a 20 y que están totalmente incorreladas en la Figura 1.6b. Ahora, como bien hemos deducido en el análisis del problema, se ha visto conveniente utilizar una transformación polinómica de segundo orden a los datos. Esta transformación se hace a través de `PolynomialFeatures` de Scikit Learn.



(a) Originales (b) PCA(0.95) (c) PCA(0.95), PolFeat(2)

Figura 2.5: Representación de la matriz de correlación en valor absoluto durante el preprocesado. La matriz de correlación nos indica cómo de correlada están las variables entre sí dando lugar a una matriz simétrica. El valor absoluto es simplemente para captar visualmente de forma sencilla si una variable está correlada (directa o inversamente) con otra: estará menos correlada cuanto más oscuro sea el "píxel" o "celda" que les corresponde al par de variables.

Aprovechamos este aumento de dimensionalidad para poder quitarnos variables por una varianza baja. Aplicamos ahora un umbral de varianza con `VarianceThreshold(0.1)`. Y por último volvemos a normalizar los datos con `textStandardScaler`.

Tras todas estas transformaciones podemos ver en la Figura 1.6c como queda de correlado nuestro nuevo espacio muestral $\mathcal{Z}$. Vemos que la correlación se mantiene a un nivel medio en general, lo que es seguro es que las transformaciones de segundo orden nos van a ofrecer potencia de desempeño, y es por ello por lo que se ha creído conveniente.

[1]Una vez hayamos aplicado PCA, podremos visualizar los datos mediante t-SNE como bien se ha hablado en la sección anterior. Además, no se ha dicho, pero se recomienda en la documentación de scikit learn aplicar métodos de disminución de características para problemas de gran dimensionalidad antes de aplicar t-SNE.

2.6. Métricas

Métrica de error

Lo usual en un problema de clasificación es usar la métrica:

$$Error(h) = \frac{1}{N}\sum_{i=1}^{N}[[h(x_n) \neq y_n]] \tag{2.1}$$

Esta métrica nos permite medir en proporción el error (cuanto más cercano a 1 peor, cuanto más cercano a 0 mejor). Sin embargo, para representar los resultados en este problema de clasificación se ha usado el *Accuracy*. El *Accuracy* lo definimos como:

$$Accuracy = \frac{TP + TN}{TP + FN + FP + TN} = 1 - Error(h) \tag{2.2}$$

Donde

$TP = True\ Positives$, $TN = True\ Negatives$, $FP = False\ Positives$, $FN = Flase\ Negatives$

Básicamente nos dice la proporción de aciertos que hemos tenido. Se ha seleccionado como métrica principal por su fácil interpretación y porque se considera que tiene un fuerte significado.

Además, se ha usado la métrica *F1-score* que es la media armónica de las métricas *precision* y *recall*. Recordamos que

$$\begin{aligned} Precision &= \frac{TP}{TP + FP} \\ Recall &= \frac{TP}{TP + FN} \end{aligned} \tag{2.3}$$

con lo que,

$$F1score = \left(\frac{Precision^{-1} + Recall^{-1}}{2}\right)^{-1} = 2\frac{Precision \times Recall}{Precision + Recall} \tag{2.4}$$

Que si lo desarrollamos nos quedaría que:

$$F1score = \frac{2TP}{2TP + FP + FN} \tag{2.5}$$

Vemos una clara semejanza con el *Accuracy*, donde en vez de evaluar TP y TN se ha decidido únicamente hacer caso a los TP. Como se espera con la fórmula es que conforme el *F1-score* sea próximo a 1, será un mejor resultado. Al igual que conforme se acerque a 0 tendremos un peor resultado. Para el caso multiclase, esta métrica será la media del *F1-score* para cada clase.

2.7. Regularización

Como penalizaciones se ha considerado únicamente el *weight decay* (`'l2'`) para todos los modelos. La regularización *weight decay* es una regularización muy recomendada siempre porque su propósito es disminuir los valores de todos los parámetros uniformemente, haciendo que en general los pesos tomen valores bajos. Esto es conveniente porque queremos evitar sobreajustarnos: hay presencia de ruido determinístico y estocástico. El ruido suele tener frecuencias muy altas así que llevando los pesos a valores bajos conseguimos suavizar nuestra regresión

obteniendo frecuencias bajas y alejándonos del ruido.

La fuerza de la regularización se ha decidido a través de cross-validation, en donde ha tenido 3 valores distintos para darle peso al *weight decay*: $2^{-1}, 2^{3/2}, 2^4$. Valores que aportan desde una fuerza suave, a una fuerte regularización.

2.8. Cross-Validation. Selección de la mejor hipótesis

Pasamos ahora a ejecutar técnicas de validación. Usaremos Cross-Validation con 5-folds puesto que el tamaño de datos del problema es bastante grande.

Usaremos la función `GridSearchCV` que viene bien explicada con la elección de los parámetros y demás en el Apéndice II. Básicamente en el Apéndice II se habla de la funcionalidad de la misma función, el por qué de los parámetros que hemos usado y demás cosas relacionadas.

La razón de usar esta técnica viene porque es el mejor *modus operandi* para elegir un modelo. Al final tenemos varias opciones de estimadores que cada uno de ellos conlleva varias opciones de parámetros. Es una forma en la que el error de validación se hace bastante representativo y se podría tomar como una estimación de lo que sería el error fuera de la muestra.

La mejor hipótesis escogida por nuestra validación cruzada ha sido referente al modelo de *Regresión Logística*. En concreto tenemos la siguiente selección de parámetros:

```
{'clf': LogisticRegression(C=16.0, class_weight=None,
                           dual=False, fit_intercept=True,
                           intercept_scaling=1, l1_ratio=None, max_iter=1000,
                           multi_class='ovr', n_jobs=None, penalty='l2',
                           random_state=None, solver='lbfgs', tol=0.0001,
                           verbose=0,warm_start=False), 'clf__C': 16.0}
```

Se observa que la fuerza de la regularización ha sido muy fuerte: `'clf__: 16.00`, esto se debe en gran parte a la cantidad de características que tenemos (230) tras la transformación polinómica de segundo orden.

El error de este modelo en el proceso de Cross-Validation ha sido de 0.05. En términos de *Accuracy* se ha obtenido un 0.95, es decir un 95 % de aciertos. Un resultado muy bueno.

Como tenemos un resultado tan bueno, veremos que en primer lugar esta libre de *data snooping* y en segundo lugar, cuando comprobemos con el conjunto de test el modelo, veremos si hay *overfitting* o no. `GridSearchCV` aplica nuestro Pipeline un `.fit_transform` sobre los folds de entrenamiento y únicamente un `.transform`, salvándonos de hacer *data snooping*, sobre el fold de validación. De la misma manera los datos no se han usado en ningún momento para decantarnos o descartar clases de funciones o modelos, solamente nos hemos basado en la naturaleza del problema para tomar decisiones. Por lo tanto, nos libramos de hacer *overfitting* a causa de *data snooping*.

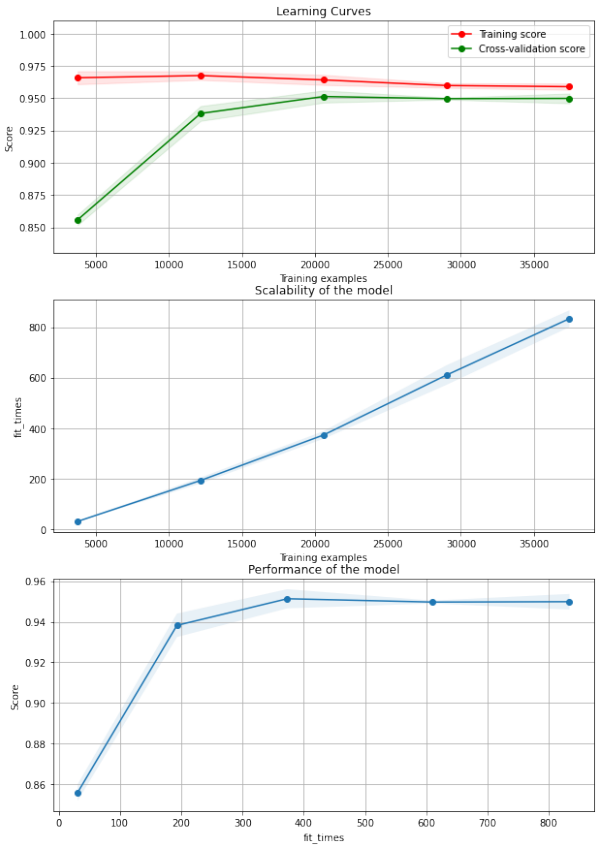


Figura 2.6: Estas gráficas se han generado a través de `learning_curve`

Para ver el desempeño (1-*Error*) de validación se han generado las curvas de aprendizaje que podemos ver en la Gráfica 2.6. Por tanto, el Score representado en las gráficas hace referencia al *Accuracy* del modelo.

En la primera gráfica podemos ver la curva de aprendizaje, en donde se muestra el desempeño del modelo en los conjunto de validación y entrenamiento. Como cabe esperar, con pocos ejemplso de entrenamiento se obtiene un *Accuracy* de validación menor, puesto que ha habido pocos ejemplos para entrenar. Conforme crece el número de ejemplos de entrenamiento, se tiende a un *Accuracy* similar en torno al 0.95: $Accuracy_{train} \downarrow$, $Accuracy_{test} \uparrow$

La estela verde y roja hacen referencia a los máximos y mínimos que se han obtenido durante cada entrenamiento con cada número de ejemplos, es decir, la desviación.

En la segunda gráfica se ve el número de entrenamientos necesarios en función de las muestras de entrenamiento. Por último, en la tercera gráfica se ve el desempeño del modelo en función del número de entrenamientos, que vemos que converge también a un valor cercano al 0.95

2.9. Entrenamiento de la mejor hipótesis en el conjunto de entrenamiento completo

En nuestro caso, al usar `GridSearchCV` con el parámetro `refit` en True, él mismo se encarga de rentrenar el mejor modelo escogido por cross-validation con el conjunto completo de datos. De esta forma, nuestra hipótesis se refina con más información y pudiendo dar una mejor estimación del error fuera de la muestra. Nuestra estimación de E_{out} la haremos a través de E_{test}.

Recordamos que el mejor modelo ha sido

```
{'clf': LogisticRegression(C=16.0, class_weight=None,
                           dual=False, fit_intercept=True,
                           intercept_scaling=1, l1_ratio=None, max_iter=1000,
                           multi_class='ovr', n_jobs=None, penalty='l2',
                           random_state=None, solver='lbfgs', tol=0.0001,
                           verbose=0,warm_start=False), 'clf__C': 16.0}
```

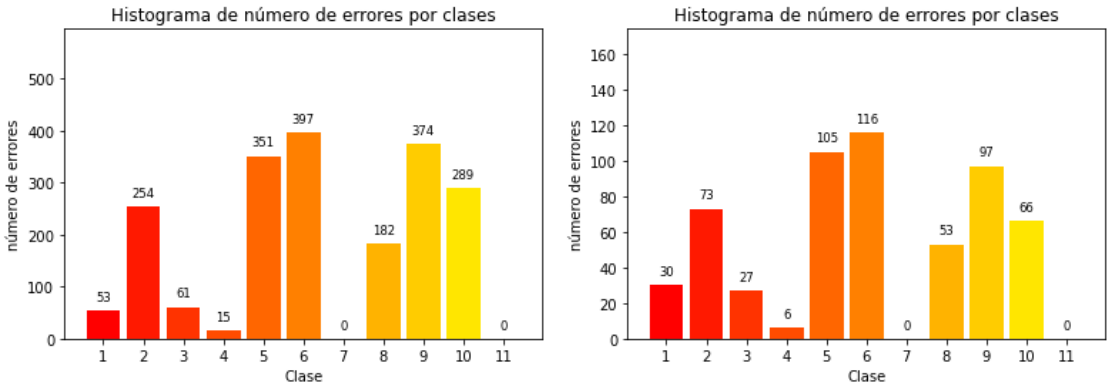
Así, al ejecutar la hipótesis final sobre los datos de entrenamiento y de test se han obtenido los resultados visibles en la Tabla 2.1. Además hay dos gráficas en la Figura 2.7 que muestra el número de errores en cada clase.

	TRAIN	**TEST**
`Tamaño`	46807	11702
`Número de fallos`	1976	573
`Error`	0.0422	0.0489
`Accuracy`	0.958	0.951
`F1-score`	0.958	0.951

Tabla 2.1: Resultados del mejor modelo en los conjuntos de Train y Test

En base a la Tabla 2.1, vemos que los resultados son muy buenos. Algo más de un 95 % de *Accuracy* es indicativo de que tenemos un muy buen modelo y que está exento de overfitting.

Es significativo ver que *F1-score* tiene la misma puntuación que el *Accuracy*. Esto es símbolo de que, al tener una muestra de tamaño tan grande, detectamos igual de bien los verdaderos positivos como los verdaderos negativos.



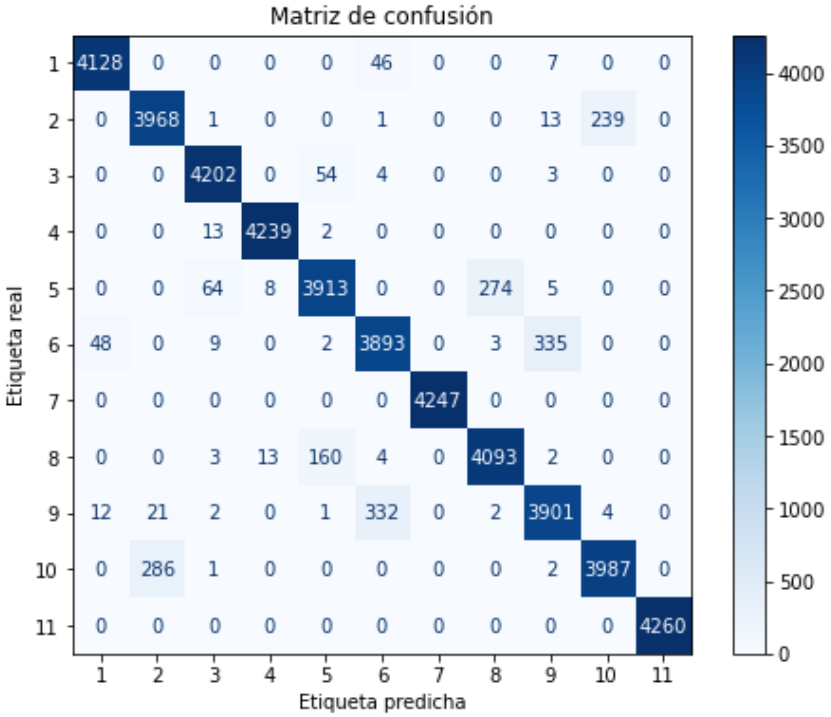
(a) Train (b) Test

Figura 2.7: Gráficas que muestran el número de errores cometidos en cada clase. Hay un error en la clase c si la etiqueta de la instancia i es igual a c $y_i = c$ y la predicción es distinta: $h(x_i) \neq c$

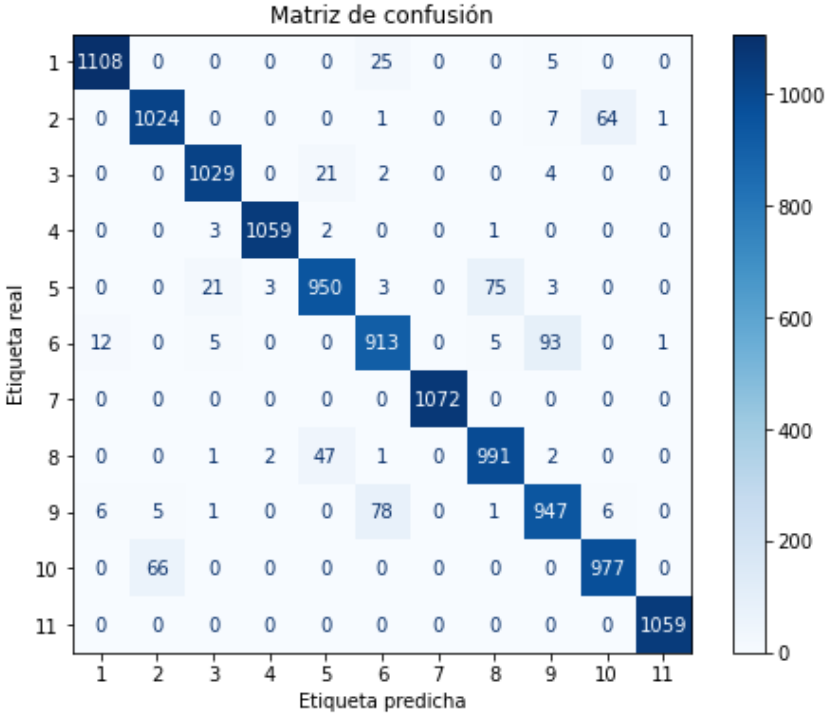
Comentar ahora las gráficas de la Figura 2.7. Se ve como los errores no se distribuyen de la misma manera en todas las clases, esto puede ser debido a que se requiere de un modelo más complejo.

Ahora recordamos al inicio del problema, cuando estudiamos las componentes principales realizadas por PCA, como en la Gráfica 2.1 veíamos que la componente 2 explicaba bien tanto la clase 6 como la 10. En realidad son la clase 7 y 11, puesto que en las gráficas de Gráfica 2.1 se numeran desde el 0. Ahora volvemos a la Figura 2.7 y vemos como en las clases 7 y 11 tenemos un error de 0, símbolo de la simplicidad que ya habíamos visto que tenía la predicción de esas clases.

Por último valorar las matrices de confusión de la Figura 2.8, donde simplemente tenemos que observar la fuerza de la diagonal principal que es símbolo de un muy buen desempeño. La diagonal principal nos dice el número de etiquetas que han sido predichas correctamente para cada clase.



(a) Train



(b) Test

Figura 2.8: Matriz de confusión: herramienta que permite la visualización del desempeño de un algoritmo que se emplea en aprendizaje supervisado. Cada columna de la matriz representa el número de predicciones de cada clase, mientras que cada fila representa a las instancias en la clase real. Uno de los beneficios de las matrices de confusión es que facilitan ver si el sistema está confundiendo dos clases.

Apéndice I

En este apéndice se explicarán todos los conceptos o técnicas que no se han visto previamente. En el texto, todos aquellos conceptos que no se han explicado y se considera que es necesaria su explicación tendrán una referencia a este apéndice.

3.1. PCA: Principal Component Analysis

Ver uso en los problemas en PCA-Sklean . El método de *Análisis de Componentes Principales* es un método estadístico que permite simplificar la complejidad de espacios muestrales con muchas dimensiones a la vez que conserva su información. Supongamos que existe una muestra con n individuos cada uno con p variables $(X_1, X_2, ..., X_p)$, es decir, el espacio muestral es de p dimensiones. *PCA* permite encontrar un número de factores subyacentes $(z < p)$ que explican, aproximadamente, lo mismo que las p variables originales. Cada una de estas z nuevas variables recibe el nombre de componente principal.

El método de *PCA* permite por lo tanto "condensar" la información aportada por múltiples variables en solo unas pocas componentes. Esto lo convierte en un método muy útil de aplicar previa utilización de otras técnicas estadísticas tales como regresión, clustering...

Cálculo de las componentes principales

Cada componente principal (Z_i) se obtiene por combinación lineal de las variables originales.

$$Z_i = \phi_{1,i}X_1 + \phi_{2,i}X_2 + ... + \phi_{p,i}X_p \tag{3.1}$$

La combinación lineal tiene que cumplir:

$$\sum_{j=1}^{p} \phi_{j,i}^2 = 1, \qquad i \in 0, 1, .., z \tag{3.2}$$

El proceso a seguir para calcular las componentes es:

1. Centralización de las variables restando la media
2. Se resuelve un problema de optimización para encontrar el valor de los pesos con los que se maximiza la varianza. Una forma de resolver esta optimización es mediante el cálculo de eigenvector-eigenvalue de la matriz de covarianzas.

Una vez calculada la primera componente se calculan las siguientes repitiendo el mismo proceso, pero añadiendo la condición de que la combinación lineal no puede estar correlacionada con las anteriores componentes (esto es equivalente a decir que Z_i tiene que ser ortogonal con Z_j con $j \neq i$).

El proceso se repite de forma iterativa hasta calcular todas las posibles componentes, es decir, mín$(n-1,p)$ veces, o hasta que se decida detener el proceso. El orden de importancia de las componentes viene dado por la magnitud del *eigenvalue* asociado a cada *eigenvector*. Una forma de detener el proceso es obtener tantas componentes como varianza acumulada queramos que expliquen.

3.2. t-SNE: t-Distributed Stochastic Neighbor Embedding

T-SNE es un método estadístico para visualizar datos de alta dimensión dando a cada punto de datos una ubicación en un mapa bidimensional o tridimensional. Es una reducción de dimensionalidad no lineal muy adecuada para incrustar datos de alta dimensión para visualización en un espacio de baja dimensión de dos o tres dimensiones. Específicamente, modela cada objeto de alta dimensión por un punto bidimensional o tridimensional de tal manera que objetos similares son modelados por puntos cercanos y objetos diferentes son modelados por puntos distantes con alta probabilidad.

El algoritmo t-SNE comprende dos etapas principales. Primero, t-SNE construye una distribución de probabilidad sobre pares de objetos de alta dimensión de tal manera que a los objetos similares se les asigna una probabilidad más alta mientras que a los puntos diferentes se les asigna una probabilidad más baja. En segundo lugar, t-SNE define una distribución de probabilidad similar sobre los puntos en el mapa de baja dimensión y minimiza la divergencia Kullback-Leibler (divergencia KL) entre las dos distribuciones con respecto a las ubicaciones de los puntos en el mapa. Si bien el algoritmo original usa la distancia euclidiana entre objetos como base de su métrica de similitud, esto se puede cambiar según corresponda.

Dado un conjunto de N objetos de alta dimensión $x_1, x_2, ..., x_N$, t-SNE calcula probabilidades p_{ij} que son proporcionales a la similitud de los objetos x_i, x_j. Se define para $i \neq j$

$$p_{j|i} = \frac{exp(-||x_i - x_j||^2/2\sigma_i^2)}{\sum_{k\neq i} exp(||x_i - x_k||^2/2\sigma_i^2)} \tag{3.3}$$

y $p_{i|i} = 0$. Ahora se define

$$p_{ij} = \frac{p_{j|i} + pi|j}{2N} \tag{3.4}$$

y notar que $p_{ij} = p_{ji}, p_{ii} = 0, \sum_{i,j} p_{ij} = 1$.

t-SNE tiene como objetivo representar un mapa en el que se reflejen las similitudes p_{ij} tan bien como sea posible. Para ello, mide las similitudes q_{ij} entre dos puntos en el mapa y_i, y_j, utilizando un enfoque similar. Para $i \neq j$ definimos q_{ij} como:

$$q_{ij} = \frac{1 + ||y_i - y_j||^2)^{-1}}{\sum_k \sum_{l\neq k}(1 + ||y_k - y_l||^2)^{-1}} \tag{3.5}$$

estableciendo $q_{ii} = 0$. Aquí se usa la distribución t-Student para medir similitudes entre puntos de baja dimensión para así permitir que puntos diferentes se modelen separados en el mapa.

Las ubicaciones de los puntos y_i en el mapa se determinan minimizando la divergencia de Kullback-Leibler de la distribución P respecto de la distribución Q:

$$KL(P||Q) = \sum_{i\neq j} p_{ij} log \frac{p_{ij}}{q_{ij}} \tag{3.6}$$

Esta minimización se hace a través de mi método de gradiente descendente.

Apéndice II

Funciones de Scikit-Learn

En este apéndice se tocará el tema de las funciones de sklearn: para qué sirven y por qué se han usado.

4.1. Grid Search Cross Validation

Cross Validation: `sklearn.model_selection`. Busca sobre un grid de parámetros el mejor modelo. Una instancia de `GridSearchCV` busca los parámetros dentro de un grid (se le puede pasar una lista de grids, que es lo que se hace precisamente en Modelos) en base a los folds que se le han dicho por parámetro: en nuestro caso 10. Los grids que se le pasen tienen que hacer referencia a parámetros del estimador o elementos del Pipeline que se la haya pasado como estimador.

El parámetro `refit` que por defecto está en `True` indica que, una vez que se ha encontrado el mejor modelo (el que mejor score tiene), este modelo se entrene con todo el conjunto de datos.

De score ponemos las dos métricas que hemos elegido: -MSE (regresión) y Accuracy (clasificación). El parámetro `n_jobs` hace referencia al número de procesadores que intervienen en el cómputo: está a menos 1 que en cierto modo quiere decir, que se usen todos los que se puedan.

4.2. Learning Curve

Curvas de aprendizaje: `sklearn.model_selection`. Nos permite obtener los puntajes de score durante el entrenamiento en cross validation. Para las gráficas de la Figura 1.7 se ha usado el código disponible en: Plotting Learning Curves

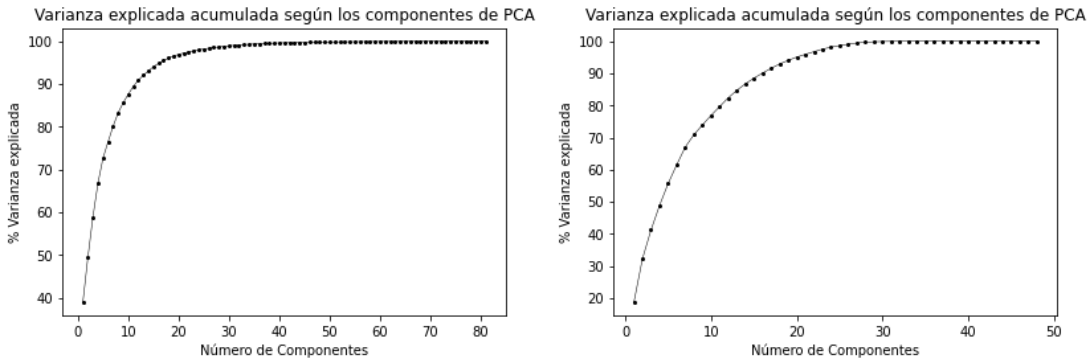
4.3. PCA

Método de Análisis de Componentes Principales (`sklearn.decomposition`). Información extra sobre la técnica de PCA se puede ver en el Apéndice I, en la sección PCA. PCA se ha aplicado en los problemas tanto sin argumentos como con argumentos. En el caso de la aplicación al problema como tal se ha usado:

```
PCA(0.95)
```

En donde, se ha considerado que, que se explique el 95 % de la varianza es conveniente para ambos problemas. Esta decisión se ha tomado a raíz de ver la varianza explicada por cada

componente. Para ver esta varianza se ha usado `PCA()`, que nos explica el 100 % de la varianza. Aplicándoselo a los datos, podemos sacar el ratio de varianza explicada por cada componente. Se ha representado gráficamente la varianza acumulada en ambos problemas (Figura 4.1).



(a) Regresión (b) Clasificación

Figura 4.1: Explicación de la varianza acumulada por componentes

En ambos problemas, como se puede ver, a partir del 95 % las componentes explican muy poco por lo que se ha creído conveniente dejar en 0.95 la explicación de las componentes en base a estas gráficas.

4.4. Polynomial Features

Características polinómicas dentro de `sklearn.preprocessing`. Genera una nueva matriz que consiste en todas las combinaciones polinómicas de las características del orden que se indique por parámetro. El único parámetro que se ha necesitado usar es `degree`, que ya tiene por defecto el grado 2, que es el que queremos.

4.5. Standard Scaler

Es una herramienta de preprocesado (`sklearn.preprocessing`) que ofrece Scikit Learn. En la propia documentación se nos dice que la operación que realiza es restar por la media y dividir por la varianza. Los parámetros se han dejado por defecto para que pueda, precisamente, centrar en cero los datos y hacer que tengan una varianza de 1.

El motivo de usar esta función ha sido por la necesidad de rescalar los datos a unas varianzas similares para el proceso de aprendizaje, además de faciliar la tarea a técnicas como PCA, en donde se recomienda encarecidamente realizar este tipo de normalizaciones o estandarizaciones.

4.6. Variance Threshold

Es un método de selección de características (`sklearn.feature_selection`) que elimina todas aquellas que tengan una varianza por debajo de un umbral. En ambos problemas se utilizará el umbral=**0** para eliminar valores constantes (no varían en ninguna muestra) y el umbral=**0.1** para eliminar valores que varían mínimamente.

Apéndice III

En este apéndice se tratará todo lo referente al código de los ficheros python de cada uno de los problemas. Ambos problemas comparten muchas similitudes, lo que hará más simple la compresión en su conjunto.

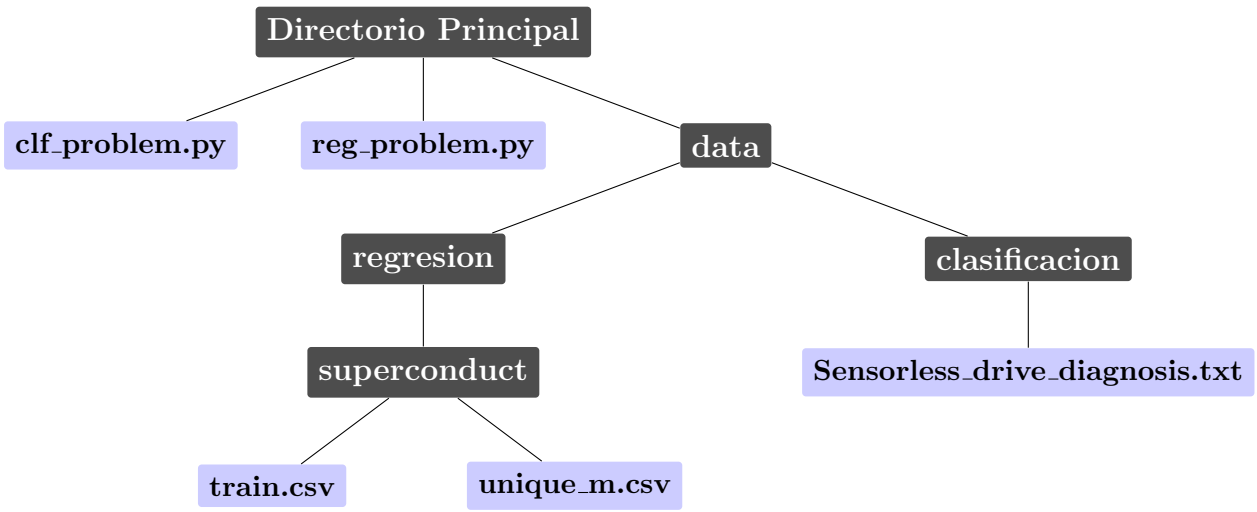
5.1. Requisitos

5.1.1. Datos

Tener descargados los datos y tener acceso a ellos. Bien sea desde tu ordenador o desde entornos online como Google Collab. En el caso de ordenador personal, descargar los ficheros y tenerlos accesibles desde el fichero python. En el caso de entornos online como Google Collab, guardar los datos en sitios accesibles, como sería Drive desde Google Collab.

5.1.2. Estructura de directorios

Tener la siguiente estructura de ficheros



En caso de no tener esta estructura y no querer adaptarla, puedes adaptar el `path` de los archivos de datos en cada fichero dentro de la función

`ejecucion_regresion , ejecucion_clasificacion`

en donde se llama a `read_data`. En concreto hay que poner

`read_data(PATH,ARCHIVO.EXTENSION)`

5.2. Estructura del código

Ambos códigos poseen una estructura similar. El código está dividido por apartados, en cada apartado se implementan una serie de funciones que tienen una funcionalidad común. Estos apartados son:

1. **Variables Globales**: aquí tendremos variables que determinan la ejecución del problema, en concreto tenemos las siguientes variables en ambos problemas:
   a) `SEED`: Semilla utilizada para funciones aleatorias.
   b) `MOSTRAR_GRAFICOS_TSNE`: variable booleana que determinará si se quiere ejecutar la técnica de **t-SNE** o no.
   c) `MOSTRAR_CURVA_DE_APRENDIZAJE`: variable booleana que determinará si se quiere ejecutar la función de **learning_curve** o no.
2. **Lectura de Datos**: alberga a la función `read_data` de donde se leen los datos.
3. **Visualización de Datos**: contiene todas las funciones que muestran resultados y gráficas.
4. **Funciones Principales**: contiene cuatro funciones. Una para crear los pasos del preprocesado. Otra para visualizar los datos para entender el problema. Otra para resolver el problema que se nos plantean (esta última función es `fit_reg` ó `fit_clf`). Una última que ejecuta la lectura de datos más las tres anteriores secuencialmente.
5. **Ejecución**: Primera parte en ejecutarse del código

5.3. Variables de modificación de ejecución

Las variables siguientes están por defecto en `False`, pero se pueden cambiar a `True` si así se desea con el correspondiente pago en tiempo de ejecución:

1. `MOSTRAR_GRAFICOS_TSNE`: variable booleana que determinará si se quiere ejecutar la técnica de **t-SNE** o no.
2. `MOSTRAR_CURVA_DE_APRENDIZAJE`: variable booleana que determinará si se quiere ejecutar la función de **learning_curve** o no.

son importantes ya que de ellas depende un gran cambio en el tiempo de ejecución del programa. La primera, mostrar los gráficos t-SNE o no puede llevar una diferencia de hasta media hora de ejecución. La segunda, más importante aún, hace que se ejecute o no el cálculo para la curva de aprendizaje, algo que conlleva horas (3 en el caso de clasificación, 1 y pico en el de regresión).

5.4. Tiempos de ejecución

Los tiempos que tenemos en caso de que las variables `MOSTRAR_CURVA_DE_APRENDIZAJE` y `MOSTRAR_GRAFICOS_TSNE` estén a false son:

PROBLEMA	**Tiempo(min)**
REGRESIÓN	4
CLASIFICACIÓN	20

Tabla 5.1: Tiempos

Bibliografía

[SKlearn] Scikit Learn: `https://scikit-learn.org/`. Todo lo relativo al Apéndice II. Recurso online.

[PCA] Análisis de componentes principales: `https://www.cienciadedatos.net/documentos/35_principal_component_analysis`. Recurso online.

[t-SNE] t-Distributed Stochastic Neighbor Embedding: `https://en.wikipedia.org/wiki/T-distributed_stochastic_neighbor_embedding`. Recurso online.

[MSE] Error Cuadrático Medio: `https://es.wikipedia.org/wiki/Error_cuadr%C3%A1tico_medio`. Recurso online.

[R^2] Coeficiente de determinación lineal R^2: `https://es.wikipedia.org/wiki/Coeficiente_de_determinaci%C3%B3n`. Recurso online.