# Práctica 3

Celia Arias Martínez

En esta práctica vamos a realizar el ajuste y selección del mejor predictor lineal para un conjunto de datos dados. Vamos a tener dos problemas: uno de regresión y otro de clasificación, por lo que haremos dos secciones, y desarrollaremos dentro de cada sección los pasos que llevaremos a cabo, todos ellos encaminados a seleccionar el mejor modelo y la mejor estimación de error $E_{out}$

## Regresión

Para este problema utilizamos la base de datos *Superconductivty Data Data Set* encontrada en https://archive.ics.uci.edu/ml/datasets/Dataset+for+Sensorless+Drive+Diagnosis.

### Comprensión del problema a resolver. Identificación de los elementos $X, Yyf$.

El problema que queremos resolver es la asignación de una temperatura crítica a un superconductor, dadas unas características como el número de elementos, la masa atómica, el radio atómico, el punto de fusión, la entropía, etcétera. En concreto tenemos datos sobre 21263 superconductores, y de cada uno de ellos tenemos 81 características.

Por tanto los elementos son: $X$ : matriz con las características de los superconductores $Y$: temperatura crítica del superconductor $f$: función que asocia a cada superconductor su temperatura crítica.

Para comprender mejor el problema vamos a visualizar los datos. Para ello realizaremos dos pasos: reducir el conjunto de datos a un número de dimensiones razonable con la técnica PCA y visualizar el conjunto de datos con la técnica t-SNE.

from sklearn.preprocessing import StandardScaler from sklearn.decomposition import PCA #Tenemos que escalar las características de nuestros datos antes de aplicar PCA. #Normalizamos los datos para que tengan media 0 y varianza 1

x = StandardScaler().fit_transform(x)

#Aplicamos PCA pca = PCA(n_components = 50) principales = pca.fit_transform(x) print(x[0])

**Selección de la clase de funciones a usar.**

**Identificación de las hipótesis finales que vamos a usar.**

**Partición en test y entrenamiento.**

**Preprocesado de datos.**

**Justificación de la métrica de error a usar.**

**Justificación de los parámetros y del tipo de regularización usada.**

**Selección de la mejor hipótesis del problema.**

**Modelo final con todos los datos.**

# Clasificación

Para este problema utilizamos la base de datos *Superconductivty Data Data Set* encontrada en https://archive.ics.uci.edu/ml/datasets/Dataset+for+Sensorless+Drive+Diagnosis.

## Comprensión del problema a resolver. Identificación de los elementos $X, Yyf$.

El problema que queremos resolver es dado un motor, asignarlo a una de las once clases que tenemos. Cada motor tiene unas características como DUDA. En concreto tenemos datos sobre 58509 motores, y de cada uno de ellos tenemos 49 características.

Para comprender mejor el problema vamos a visualizar los datos. Para ello aplicaremos el algoritmo t-SNE. En este caso no hace falta aplicar antes PCA ya que tenemos un número de atributos inferior a 50 (tenemos 49). t-SNE intenta reproducir la distribución que existe en el espacio original en otro espacio de dimensión menor, en este caso de dimensión 2 para que podamos visualizarlo. Al contrario que PCA, que simplemente maximiza la varianza, t-SNE hace que puntos con características parecidas queden cerca en el modelo final, y los que menos se parecen queden alejados.

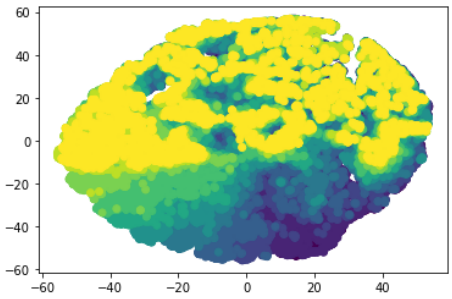



Figura 1: t-SNE parámetros por defecto

t-SNE admite algunos parámetros tales como:

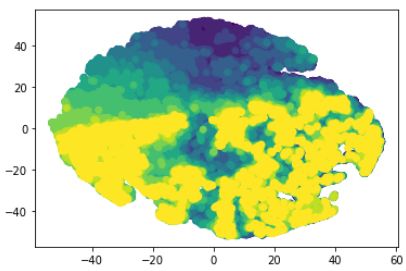
- *perplexity* : valor entre 5 y 50 (mayor cuanto mayor sea el dataset). Por defecto vale 30.
- *early_exaggeration*: Este parámetro controla la distancia entre bloques semejantes en el espacio final. La elección de este valor no es crítico. Por defecto vale 12.
- *learning_rate*: Habitualmente en el rango (10-1000). Si es muy elevado, los datos transformados estarán formados por una bola de puntos equidistantes unos de otros. Si es muy bajo, los puntos se mostrarán comprimidos en una densa nube con algunos outliers. Poe defecto vale 200.
- *n_iter*: Número máximo de iteraciones para la optimización. Debería ser, por lo menos, 250. Por defecto vale 1000.
- *metric*: métrica para la medición de las distancias.
- *method*: algoritmo a usar para el cálculo del gradiente.

Vamos a cambiar algunos valores de los parámetros, para ver cómo influye en el resultado final y cuales de ellos se ajustan mejor a nuestro modelo.

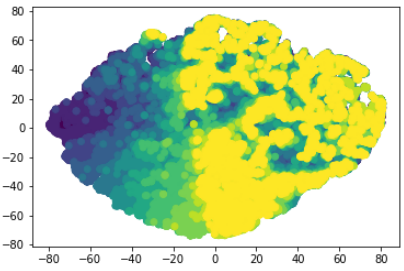
Vemos que no tenemos diferencias muy significativas al variar los parámetros, al menos no tenemos diferencias que nos añadan más información de la que disponemos. En cuanto al tiempo podemos ver que el método tarda mucho tiempo en ejecutarse, pero tampoco he podido disminuir ese tiempo al cambiar los parámetros.

## Selección de la clase de funciones a usar.

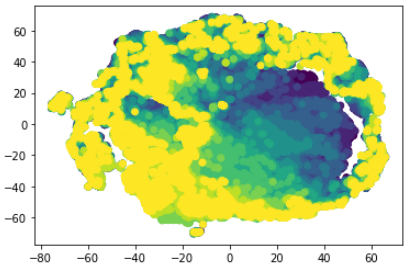
Primero vamos a ajustar un modelo lineal, ya que si obtenemos buenos resultados con él no hace falta probar con otros modelos más complejos.



(a) perplexity = 45



(b) learning rate = 400



(c) early-exaggeration=30

**Identificación de las hipótesis finales que vamos a usar.**

**Partición en test y entrenamiento.**

**Preprocesado de datos.**

**Justificación de la métrica de error a usar.**

**Justificación de los parámetros y del tipo de regularización usada.**

**Selección de la mejor hipótesis del problema.**

**Modelo final con todos los datos.**