UNIVERSIDAD
DE GRANADA

Aprendizaje Automático
grado en ingeniería informática

PRÁCTICA 3

Programación

Autor
Vladislav Nikolov Vasilev

Rama
Computación y Sistemas Inteligentes

Escuela Técnica Superior de Ingenierías Informática y de
Telecomunicación

Curso 2018-2019

Índice

1. Problema de Regresión

1.1. Descripción del problema

En este problema vamos a trabajar con el conjunto de datos *Airfoil Self-Noise*, el cuál ha sido proporcionado por la NASA, y contiene los resultados de haber realizado un conjunto de pruebas aerodinámicas y acústicas en un túnel de viento sobre perfiles alares de dos y tres dimensiones.

El conjunto de datos está compuesto por 1503 filas y 6 columnas, los valores de las cuáles son todos números reales. Los datos de las 5 primeras columnas se corresponden con los datos de entrada, y la última columna se corresponde con la información de salida. A continuación se puede ver que representa cada uno de los atributos de forma ordenada:

1. Frecuencia, medida en Hz.
2. Ángulo de ataque (ángulo que forman la cuerda geométrica de un perfil alar con la dirección del aire incidente), medida en grados.
3. Longitud de la cuerda del perfil alar, medida en metros.
4. Velocidad *free-stream*, medida en metros por segundo.
5. Distancia de desplazamiento de succión, medida en metros.
6. Nivel de presión sonora, medida en dB.

1.2. Análisis de los datos

Antes de comenzar con todo el proceso de elección y selección de un modelo lineal, vamos a pararnos un momento para analizar los datos de los que disponemos con el fin de obtener más información sobre el problema.

Lo primero que tenemos que hacer es cargar los datos. Para ello, vamos a usar una función genérica que nos permita leer ficheros de datos y obtener un *DataFrame* que podamos usar luego. Vamos a ver como sería esta función:

```
In [1]: import numpy as np
        import pandas as pd
        import matplotlib.pyplot as plt
```

```
# Establecer la semilla que vamos a utilizar
np.random.seed(1)

def read_data_values(in_file, separator=None):
    """
    Funcion para leer los datos de un archivo

    :param in_file Archivo de entrada
    :param separator Separador que se utiliza en el archivo
                     (por defecto None)

    :return Devuelve los datos leidos del archivo en un DataFrame
    """

    # Cargar los datos en un DataFrame
    # Se indica que la primera columna no es el header
    if separator == None:
        df = pd.read_csv(in_file, header=None)
    else:
        df = pd.read_csv(in_file, sep=separator, header=None)

    return df
```

Con la función ya mostrada, vamos a cargar los datos y mostrar los primeros valores de la muestra, para tener una idea de como serán los datos:

Con la función ya mostrada, vamos a cargar los datos y mostrar los primeros valores de la muestra, para tener una idea de como serán los datos:

```
In [2]: df = read_data_values('datos/airfoil_self_noise.dat', separator='\t')

        # Asignamos nombres a las columnas (según los atributos)
        column_names = ['Frequency', 'Angle of attack', 'Chord length',
                        'Free-stream velocity', 'SSD thickness',
                        'Sound Pressure']
        df.columns = column_names

        # Mostrar primeros valores de los datos
        df.head()
```

	Frequency	Angle of attack	Chord length	Free-stream velocity	SSD thickness	Sound Pressure
0	800	0.0	0.3048	71.3	0.002663	126.201
1	1000	0.0	0.3048	71.3	0.002663	125.201
2	1250	0.0	0.3048	71.3	0.002663	125.951
3	1600	0.0	0.3048	71.3	0.002663	127.591
4	2000	0.0	0.3048	71.3	0.002663	127.461

Figura 1: Tabla con las 5 primeras muestras del conjunto de training.

Antes de proseguir, vamos a dividir los datos en los conjuntos de training y test, ya que en este caso disponemos solo de un conjunto de datos (no viene separado por defecto). Para esto, vamos a crear primero una función que nos permita dividir los datos que tenemos en las características (a lo que llamaremos $\mathbf{X}$) y las etiquetas (a lo que llamaremos y). Una vez hecha esta separación, podremos dividir los datos en los dos conjuntos anteriormente mencionados. Vamos a hacer que el 80 % de los datos se quede en training y que el 20 % de los datos esté en test. Por tanto, en resumidas cuentas, estamos haciendo *hold-out*, ya que nos quedamos con una parte de los datos para poder estimar un E_{test} que nos permita acotar E_{out} posteriormente. Esto tiene sus efectos negativos, como que por ejemplo tengamos menos datos con los que entrenar y que los resultados pueden ser un poco peores por este motivo, pero al menos tenemos una capacidad para probar como de bueno es nuestro ajuste fuera de la muestra con la que lo hemos entrenado.

Con esto dicho, vamos a ver como ser haría:

```
In [3]: # Función para dividir los datos en train y test
        from sklearn.model_selection import train_test_split

        def divide_data_labels(input_data):
            """
            Funcion que divide una muestra en los datos y las etiquetas

            :param input_data Conjunto de valores que se quieren separar
                              juntados en un DataFrame

            :return Devuelve los datos y las etiquetas
            """

            #Obtener los valores
            values = input_data.values
```

```
        # Obtener datos y etiquetas
        X = values[:, :-1]
        y = values[:, -1]

        return X, y

    # Obtener valores X, Y
    X, y = divide_data_labels(df)

    # Dividir los datos en training y test
    X_train, X_test, y_train, y_test = train_test_split(X, y,
                    test_size=0.2, random_state=1, shuffle=True)
```

Con los datos ya cargados y divididos en los conjuntos de training y test, vamos a obtener cierta información sobre éstos. En problemas de este tipo nos interesa conocer por ejemplo si en ciertos casos faltan datos (no se ha podido obtener información sobre todos los atributos debido a que es imposible hacerlo, han habido errores a la hora de tomarlos o no se disponía de las herramientas necesarias), el número de valores distintos, los rangos de los datos (valores mínimos y máximos para cada atributo), si existe algún tipo de correlación entre las variables, etc.

Vamos a comenzar estudiando primero las características más simples, para adentrarnos luego en el estudio de la correlación. Empecemos mirando información del conjunto training[1]:

```
In [4]: # Clase para mostrar información de DataFrames resumida
        from pandas_summary import DataFrameSummary
        from IPython.core.display import display

        # Crear información resumida sobre los datos de training
        train_df = pd.DataFrame(columns=column_names,
                                data=np.c_[X_train, y_train])

        # Crear un DataFrame de resumen y mostrarlo
        train_sum = DataFrameSummary(train_df).summary()
        display(train_sum)
```

[1]El módulo que se ha utilizado para obtener la información resumida no viene instalado por defecto en el entorno de *conda* y no se puede instalar en éste. Se puede instalar mediante pip, pero como tal, no se encuentra disponible para *conda*. Por tanto, al no poder utilizarlo en Spyder, no se incluirá esta funcionalidad en el código entregado.

	Frequency	Angle of attack	Chord length	Free-stream velocity	SSD thickness	Sound Pressure
count	1202	1202	1202	1202	1202	1202
mean	2966.26	6.73569	0.136403	50.941	0.0111494	124.852
std	3246.55	5.9433	0.0934083	15.605	0.0132485	6.9909
min	200	0	0.0254	31.7	0.000400682	103.38
25%	800	2	0.0508	39.6	0.00251435	120.1
50%	2000	5.3	0.1016	39.6	0.00495741	125.821
75%	4000	10.875	0.2286	71.3	0.0150478	130.071
max	20000	22.2	0.3048	71.3	0.0584113	140.987
counts	1202	1202	1202	1202	1202	1202
uniques	21	27	6	4	105	1169
missing	0	0	0	0	0	0
missing_perc	0%	0%	0%	0%	0%	0%
types	numeric	numeric	numeric	numeric	numeric	numeric

Figura 2: Tabla que contiene el resumen de los datos de training.

Aquí podemos ver que para ninguna de las variables faltan datos, lo cuál nos ahorra tiempo extra de procesado en el que tendríamos que insertar valores a partir de algún valor estadístico (valores medios, por ejemplo).

También podemos ver información sobre como varían los datos, tanto los de entrada como los de salida. Vemos que, por ejemplo, **Frequency** es una característica que varía mucho, ya que tiene unos valores mínimos y máximos muy dispares, además de tener una desviación típica muy elevada. Posiblemente este atributo contenga *outliers*, pero al no disponer de demasiados datos, y al ser tan pocos los posibles valores anómalos, no merece la pena intentar eliminarlos. Observando el resto de características, nos encontramos con unos valores que varían menos y cuyos rangos de valores más pequeño. Lo sorprendente es que, para los datos de entrada, tenemos que hay muy pocos valores únicos (no repetidos). Esto se puede deber a que no se han medido los valores con suficiente precisión o a que no exista una verdadera variabilidad entre ellos. Para los datos de salida, en cambio, nos encontramos que hay un montón de valores distintos. Esto es normal, ya que, al ser valores reales, hay muchos posibles valores. De aquí podemos concluir que, a pesar de que nos encontremos ante un problema con variable reales, parece que los valores que toman las variables de entrada están discretizados, es decir, que no son exactamente contínuos.

Pasemos ahora a analizar el conjunto de datos de entrenamiento. Para obtener suficiente información, vamos a fijarnos solo en valores únicos y si faltan datos, teniendo en cuenta que nunca debemos obtener información completa sobre los datos de test, ya que se supone que nunca serán conocidos y que nunca deberíamos verlos. A continuación, podemos ver esta información:

```
In [5]: # Crear DataFrame con los datos de test
        test_df = pd.DataFrame(columns=column_names,
                               data=np.c_[X_test, y_test])

        # Crear un DataFrame resumen y mostrar algunos
        # valores estadísticos
        test_sum = DataFrameSummary(test_df).columns_stats
        display(test_sum)
```

	Frequency	Angle of attack	Chord length	Free-stream velocity	SSD thickness	Sound Pressure
counts	301	301	301	301	301	301
uniques	20	27	6	4	97	300
missing	0	0	0	0	0	0
missing_perc	0%	0%	0%	0%	0%	0%
types	numeric	numeric	numeric	numeric	numeric	numeric

Figura 3: Tabla que contiene el resumen de los datos de test.

Como se puede ver, el número de valores únicos, para las variables de entrada, son muy próximos a los que teníamos anteriormente. En el caso de los valores de salida, podemos observar que hay mucha diversidad. Y, finalmente, como punto positivo, podemos ver que en ninguna de las muestras faltan datos.

Una vez hecho este pequeño análisis, pasemos a observar ahora la correlación entre las variables. Vamos a intentar obtener, para cada una de las variables (tanto las de entrada como las de salida) el coeficiente de correlación de Pearson. El resultado se puede ver a continuación:

```
In [6]: # Obtener gráfica de correlación de Pearson
        corr = train_df.corr()
        corr.style.background_gradient(cmap='Spectral')
```

	Frequency	Angle of attack	Chord length	Free-stream velocity	SSD thickness	Sound Pressure
Frequency	1	-0.270796	-0.018266	0.122803	-0.233431	-0.395423
Angle of attack	-0.270796	1	-0.498267	0.0703666	0.754145	-0.159278
Chord length	-0.018266	-0.498267	1	-0.0241411	-0.22103	-0.23498
Free-stream velocity	0.122803	0.0703666	-0.0241411	1	-0.00863181	0.13866
SSD thickness	-0.233431	0.754145	-0.22103	-0.00863181	1	-0.314796
Sound Pressure	-0.395423	-0.159278	-0.23498	0.13866	-0.314796	1

Figura 4: Tabla con los coeficientes de Pearson para cada par de variables.

Se puede ver que, en general, no existe una correlación entre la mayoría de las características. Sin embargo, sí que destacan dos casos, uno más que el otro. El primer caso es la relación que existe entre la característica **SSD thickness** y la característica **Angle of attack**. Estas dos características tienen una coeficiente de correlación de Pearson de 0.75, valor que es muy próximo a 1. Por tanto, podemos decir que existe cierta correlación entre ellas, ya que el crecimiento de una influirá en el crecimiento de la otra. Sin embargo, como el valor del coeficiente de Pearson no es 1, no podríamos asegurar con absoluta confianza que las 2 características estén correlacionadas, y que por tanto, sería necesario eliminar una de ellas. El segundo caso es **Chord length** y **Angle of attack**. Aquí lo que sucede es que el coeficiente de correlación de Pearson tiene un valor de aproximadamente -0.5. Con lo cuál, a pesar de que existe cierta correlación negativa entre las dos variables (cuando crezca una, la otra decrecerá), no podríamos afirmar con un 100 % de confianza que estén totalmente correlacionadas, ya que el valor del coeficiente está en un punto medio.

Para tener una mejor visión de todo lo que se acaba a discutir, vamos a ver un conjunto de gráficas en las que se pueden ver los valores de todas las variables dos a dos (es decir, se muestran gráficas para mostrar como cambia cada par de variables, que pueden ser tanto las características como la etiqueta de salida). Esto se puede ver a continuación:

```
In [7]: # Módulo avanzado para dibujar gráficas
        import seaborn as sns

        # Crear pares de plots para cada 2 atributos
        # También se incluyen las etiquetas
        sns.set_style("whitegrid")
        sns.pairplot(train_df)

        # Mostrar el plot
        plt.show()
```

2. Problema de Clasificación

Referencias

[1] Texto referencia
`https://url.referencia.com`