Práctica 3

Blanca Cano Camarero

Curso 2020-2021

Índice

Problema de clasificación

Análisis del problema

Estamos ante un problema de clasificación.

De la página web de la que se han obtenido los datos Dataset for Sensorless Drive Diagnosis Data Set

Abstract: Las características son extraídas de la corriente de un motos. El motor puede tener componentes intactos o defectuosos. Estos resultados se encuentras en 11 clases con diferentes condiciones.

Tenemos además la siguiente información:

- Las características del data set son multivariantes.
- Los datos son tipo números reales.
- Es una tarea de clasificación.
- En número de instancias total es 58509.
- El número de atributos es de 49.
- Si faltan datos: N/A . TO-DO (¿qué hacer si faltan datos?)

Descripción del tratamiento de los datos

Lectura y tratamiento inicial de los datos

El fichero tiene extensión `.txt` de texto plano, para leerlo usaremos la función escrita `LeerDatos (nombre_fichero, separador)` que tiene como pilar básico la función `read_csv` de la biblioteca de pandas.

Nota: suponemso que la estructura de carpetas es:

```
.
|- clasificacion.py
|- datos
    |-Sensorless_drive_diagnosis.txt
```

Donde `clasificacion.py` es el nombre del ejecutable de nuestra práctica, `datos` es una carpeta y `Sensorless_drive_diagnosis.txt` es el fichero que contiene los datos.

Selección de test y entrenamiento

Comprobaremos antes si los datos están balanceados, para ello contaré el número de distintas etiquetas.

Esto lo haremos viendo el número de veces que se repite cada etiqueta, el resultado es:

Etiqueta	Número apariciones
1.0	5319
2.0	5319
3.0	5319
4.0	5319
5.0	5319
6.0	5319
7.0	5319
8.0	5319
9.0	5319
10.0	5319
11.0	5319

Como podemos ver está perfectamente balanceado.

Debemos determinar ahora qué datos usaremos para test y cuáles para entrenamiento.

El porcentaje que voy a usar será un 20% de los datos reservados para test. La elección de esta se debe a heurísticas generales usadas y porque tenemos los suficientes datos para el entrenamiento.

En cuanto a las opciones de cómo separarlos estos deben ser seleccionados de manera aleatoria con una distribución uniforme. Desconozco si además el tamaño es suficientemente grande como para serpararlos directamente sin tener que ir clase por clase tomando el mismo número, ya que al ser homogénena, si el tamaño es sufiente puedo suponer que la selección por clases será homogénea.

Para separarlos usaré la función `sklearn.model_selection.train_test_split(*arrays, test_size=None, train_size=None, random_state=None, shuffle=True, stratify=None)` de la biblioteca de scikilearn, concretamente con los siguientes parámetros:

```
ratio_test_size = 0.2
X_train, X_test, y_train, y_test = train_test_split(
    x, y,
    test_size= ratio_test_size,
    shuffle = True,
    random_state=1)
```

- `test_size` se corresponde a la proporción de los datos que usaremos para el test, está a 0.2 porque ya hemos comentado que trabajaremos con el 20%.
- `shuffle` a `True` porque queremos coger los datos al azar.
- `random_state` es una semilla para la mezcla.

Los resultados han sido:

Etiqueta	Número apariciones
1.0	1138
2.0	1097
3.0	1056
4.0	1065
5.0	1055
6.0	1029
7.0	1072
8.0	1044
9.0	1044
10.0	1043
11.0	1059

Vemos que la mayor diferencia es de $|1138 - 1029| = 109$ si recordamos que cada clase contaba con 5319 esto supone una diferencia de $\frac{109}{5319}100 = 2.0493$ es decir que en el peor de los caos estamso entrenando con dos datos más por cada cien.

Esto no me parece del todo significativo, así que continuaré sin hacerlos por clases. (TODO Hay que justificar esto, ya sea por un paper o por).

Nótese que desde ahora solo trabajaremos con los datos de entrenamiento, para no cometer ningún tipo de data snooping.

Normalización

Diferencias muy grandes entre los datos podría perjudicar al modelo, luego comprobaremos antes si es necesario si es necesario normalizar los datos.

Para ello he diseñado la función `ExploracionInicial()` que muestra la media y la varianza de los datos.

```
--------------------
Resumen de las tablas
--------------------
```

```
Media
Valor mínimo de las medias -1.5019152989937367
Valor máximo de las medias 8.416765275493

Varianza
Valor mínimo de las varianzas 3.419960283480337e-09
Valor máximo de las varianzas 752.5259323408474
--------------------
```

La variabilidad entre las medias y datos es considerable, así que vamos a normalizar.

Para ello usaremos la función `class sklearn.preprocessing.StandardScaler(*, copy=True, with_mean=True, with_std=True)` ("SrandardScaler Del Paquete sklearnPreprocessing" n.d.) Según la documentación oficial a fecha de hoy, esta función normaliza las característias eliminando la media y escalando en función de la varianza, es calculado de la siguiente manera:

$$Z = \frac{X - U}{s}$$

Donde u es la media de los dtos de entrenamiento o cero si el parámetro `with_mean=False` y s es la desviación típica de los datos del ejemplo y 1 en caso de que `with_std=False`.

No es más que una normalización del estimado (Como se hace con una distribución de normal de varianza y media... TO-DO completar).

Correlación de los datos

Veamso ahora si podemso encontrar alguna relación entre las características, para ello vamso a utilizar la matriz de correlación.

(TO-DO Añadir información sobre la correlación)

Para calcularla utilizaremos `corrcoef` de la bibliote de numpy ("CorrcoefNumpy Del Paquete Numpy" n.d.) que devuelve el el producto de los momentos de los coeficientes.

Queda recogido el código utilizado en la función `Pearson(x,umbral,traza)`.

Para un umbral de 0.9 hemos obtenido los siguientes coeficientes:

Coeficiente	Índice 1	Índice 2
0.9999999848109128	21	22
0.9999999822104457	18	19

Coeficiente	Índice 1	Índice 2
0.9999995890206894	9	10
0.9999995669836421	22	23
0.99999955603151	19	20
0.9999995050949259	21	23
0.9999994842185346	18	20
0.999998703692659	6	7
0.9999940569381244	10	11
0.9999930415927719	9	11
0.9999790106652213	7	8
0.9999755087084263	6	8
0.9999725598028012	33	34
0.999949107548143	18	23
0.9999489468171506	19	23
0.9999482678605511	18	22
0.9999480910272748	18	21
0.9999480589259971	19	22
0.9999478753629836	19	21
0.9999476614349387	20	23
0.9999462814165702	20	22
0.9999460533676524	20	21
0.9999314039884376	30	31
0.9996912953730146	42	43
0.9996506253036762	45	46
0.9996206790946303	34	35
0.9995813582717437	33	35
0.9993189379606644	31	32
0.9991906311233252	30	32
0.9970729715793113	43	44
0.9967946202246001	42	44
0.996435194391921	46	47
0.9963402918378648	45	47
0.9266535247934785	15	16
0.9105009972932715	12	13

Si además nos fijamos se cumple la propieda transitiva, esto es, si entendemos la correlación como *Si dos vectores guardan cierta correlación superior al umbral, entonces se podría decir que uno es combinación lineal del otro*

Luego podríamos aplicar la propiedad transitiva, esto es si i explica j y j explica k entonces i explica k.

Una vez explicado esto, utilizaremos este critero para reducier la dimensionalidad del

vector de características, de tal manera que pueda verse como una base linealmente independiente.

Experimentamos con los umbrales 0.9999, 0.999, 0.95, 0.9 para ver cómo se reduce la dimensión.

Estas han sido las conclusiones (recordemos que el tamaño inicial del vector de características era de 49):

umbral	tamaño tras reducción	reducción total
0.9999	38	11
0.999	34	15
0.95	32	17
0.9	30	19

Más adelante, en la validación cruzada, experimentaremos cómo afectan las reducciones.

‘

Fuentes

“CorrcoefNumpy Del Paquete Numpy.” n.d. Accessed May 21, 2021. https://numpy.org/doc/stable/reference/generated/numpy.corrcoef.html.

“SrandardScaler Del Paquete sklearnPreprocessing.” n.d. Accessed May 21, 2021. https://scikit-learn.org/stable/modules/generated/sklearn.preprocessing.StandardScaler.html.