
Práctica 1

Inteligencia de Negocio

Amin Kasrou Aouam



**UNIVERSIDAD
DE GRANADA**

2020-11-10

Índice

Práctica 1	3
Introducción	3
Procesado de datos	3
Valores nulos	5
Valores no numéricos	6
Separación de datos	7
Configuración de algoritmos	7
Resultados obtenidos	10
Naives Bayes	10
Linear SVC	10
KNN	10
Árbol de decisión	11
Perceptrón multicapa	11
Análisis de resultados	12

Práctica 1

Introducción

En esta práctica, usaremos distintos algoritmos de aprendizaje automático para resolver un problema de clasificación.

Procesado de datos

Antes de proceder con el entrenamiento de los distintos modelos, debemos realizar un preprocesado de los datos, para asegurarnos que nuestros modelos aprenden de un *dataset* congruente.

La integridad de la lógica del preprocesado se encuentra en el archivo *preprocessing.py*, cuyo contenido mostramos aquí:

```
1 from pandas import read_csv
2 from sklearn.preprocessing import LabelEncoder
3 from sklearn.model_selection import KFold
4
5
6 def replace_values(df):
7     columns = ["BI-RADS", "Margin", "Density", "Age"]
8     for column in columns:
9         df[column].fillna(value=df[column].mean(), inplace=True)
10    return df
11
12
13 def process_na(df, action):
14     if action == "drop":
15         return df.dropna()
16     elif action == "fill":
17         return replace_values(df)
18     else:
19         print("Unknown action selected. The choices are: ")
20         print("fill: fills the na values with the mean")
21         print("drop: drops the na values")
22         exit()
23
24
25 def encode_columns(df):
26     label_encoder = LabelEncoder()
27     encoded_df = df.copy()
28     encoded_df["Shape"] = label_encoder.fit_transform(df["Shape"])
29     encoded_df["Severity"] = label_encoder.fit_transform(df["Severity"])
30    return encoded_df
31
32
33 def split_train_target(df):
34     train_data = df.drop(columns=["Severity"])
35     target_data = df["Severity"]
36    return train_data, target_data
37
38
39 def split_k_sets(df):
40     k_fold = KFold(shuffle=True, random_state=42)
41    return k_fold.split(df)
42
43
44 def parse_data(source, action):
45     df = read_csv(filepath_or_buffer=source, na_values="?")
46     processed_df = process_na(df=df, action=action)
47     encoded_df = encode_columns(df=processed_df)
48     test_data, target_data = split_train_target(df=encoded_df)
49    return test_data, target_data
```

A continuación, mostraremos cada uno de los pasos que realizamos para obtener el *dataset* final:

Valores nulos

Nuestro *dataset* contiene valores nulos, representados mediante un signo de interrogación (?). Optamos por evaluar 2 estrategias:

1. Eliminar los valores nulos

```
1 df = read_csv(filepath_or_buffer="../data/mamografia.csv",
2               na_values="?")
3 processed_df = process_na(df=df, action="drop")
4 print("DataFrame sin preprocesamiento: ")
5 print(df.describe())
6 print("DataFrame sin preprocesamiento: ")
7 print(processed_df.describe())
```

```
1 DataFrame sin preprocesamiento:
2      BI-RADS      Age      Margin      Density
3 count  959.000000  956.000000  913.000000  885.000000
4 mean    4.296142  55.487448   2.796276   2.910734
5 std     0.706291  14.480131   1.566546   0.380444
6 min     0.000000  18.000000   1.000000   1.000000
7 25%     4.000000  45.000000   1.000000   3.000000
8 50%     4.000000  57.000000   3.000000   3.000000
9 75%     5.000000  66.000000   4.000000   3.000000
10 max    6.000000  96.000000   5.000000   4.000000
11 DataFrame sin preprocesamiento:
12     BI-RADS      Age      Margin      Density
13 count  847.000000  847.000000  847.000000  847.000000
14 mean    4.322314  55.842975   2.833530   2.909091
15 std     0.703762  14.603754   1.564049   0.370292
16 min     0.000000  18.000000   1.000000   1.000000
17 25%     4.000000  46.000000   1.000000   3.000000
18 50%     4.000000  57.000000   3.000000   3.000000
19 75%     5.000000  66.000000   4.000000   3.000000
20 max    6.000000  96.000000   5.000000   4.000000
```

Observamos que el número de instancias disminuye considerablemente, hasta un máximo de 112, en el caso del *BI-RADS*. Aún así, los valores de la media y desviación estándar no se ven afectados de forma considerable.

2. Imputar su valor con la media

```
1 df = read_csv(filepath_or_buffer="../data/mamografia.csv",
  na_values="?")
2 processed_df = process_na(df=df, action="fill")
3 print("DataFrame sin preprocesamiento: ")
4 print(df.describe())
5 print("DataFrame sin preprocesamiento: ")
6 print(processed_df.describe())
```

```
1 DataFrame sin preprocesamiento:
2      BI-RADS      Age      Margin      Density
3 count  961.000000  961.000000  961.000000  961.000000
4 mean    4.296142   55.487448   2.796276   2.910734
5 std     0.705555   14.442373   1.526880   0.365074
6 min     0.000000   18.000000   1.000000   1.000000
7 25%     4.000000   45.000000   1.000000   3.000000
8 50%     4.000000   57.000000   3.000000   3.000000
9 75%     5.000000   66.000000   4.000000   3.000000
10 max    6.000000   96.000000   5.000000   4.000000
11 DataFrame sin preprocesamiento:
12      BI-RADS      Age      Margin      Density
13 count  961.000000  961.000000  961.000000  961.000000
14 mean    4.296142   55.487448   2.796276   2.910734
15 std     0.705555   14.442373   1.526880   0.365074
16 min     0.000000   18.000000   1.000000   1.000000
17 25%     4.000000   45.000000   1.000000   3.000000
18 50%     4.000000   57.000000   3.000000   3.000000
19 75%     5.000000   66.000000   4.000000   3.000000
20 max    6.000000   96.000000   5.000000   4.000000
```

Esta alternativa nos permite mantener el número de instancias en todas las columnas, sin alterar la media ni la desviación típica.

Valores no numéricos

La mayoría de algoritmos de aprendizaje automática trabaja con datos numéricos, desafortunadamente nuestro *dataset* contiene dos columnas con datos descriptivos.

Procedemos a convertirlos en valores numéricos mediante un *LabelEncoder*:

```
1 encoded_df = encode_columns(df=processed_df)
2 print(encoded_df.head())
```

	BI-RADS	Age	Shape	Margin	Density	Severity	
1	0	5.0	67.0	1	5.0	3.000000	1
2	1	4.0	43.0	4	1.0	2.910734	1
3	2	5.0	58.0	0	5.0	3.000000	1
4	3	4.0	28.0	4	1.0	3.000000	0
5	4	5.0	74.0	4	5.0	2.910734	1

Vemos como las columnas **Shape** y **Severity** se componen ahora únicamente de valores numéricos.

Separación de datos

Como último paso, separamos la columna objetivo de los demás datos.

```
1 test_data, target_data = split_train_target(df=encoded_df)
2 print("Datos de entrenamiento: ")
3 print(test_data.head())
4 print("Datos objetivo: ")
5 print(target_data.head())
```

```
1 Datos de entrenamiento:
2   BI-RADS  Age  Shape  Margin  Density
3  0     5.0  67.0     1     5.0  3.000000
4  1     4.0  43.0     4     1.0  2.910734
5  2     5.0  58.0     0     5.0  3.000000
6  3     4.0  28.0     4     1.0  3.000000
7  4     5.0  74.0     4     5.0  2.910734
8 Datos objetivo:
9  0     1
10 1     1
11 2     1
12 3     0
13 4     1
14 Name: Severity, dtype: int64
```

Configuración de algoritmos

Elegimos 5 algoritmos distintos:

1. Naive Bayes
2. Linear Support Vector Classification
3. K Nearest Neighbors
4. Árbol de decisión
5. Perceptrón multicapa (red neuronal)

Procedemos a evaluar el rendimiento de cada algoritmo, usando las siguientes métricas:

- Accuracy score
- Matriz de confusión
- Cross validation score
- Area under the curve (AUC)

Vamos a realizar 2 ejecuciones por algoritmo, para evaluar las diferencias que obtenemos según el preprocesado utilizado (eliminación de valores nulos o imputación).

La implementación se encuentra en el archivo *processing.py*, cuyo contenido mostramos a continuación:


```

1  from numpy import mean
2  from sklearn.metrics import confusion_matrix, accuracy_score,
   roc_auc_score
3  from sklearn.model_selection import cross_val_score
4  from sklearn.naive_bayes import GaussianNB
5  from sklearn.neural_network import MLPClassifier
6  from sklearn.neighbors import KNeighborsClassifier
7  from sklearn.preprocessing import scale
8  from sklearn.svm import LinearSVC
9  from sklearn.tree import DecisionTreeClassifier
10
11 from sys import argv
12
13 from preprocessing import parse_data, split_k_sets
14
15
16 def choose_model(model):
17     if model == "gnb":
18         return GaussianNB()
19     elif model == "svc":
20         return LinearSVC(random_state=42)
21     elif model == "knn":
22         return KNeighborsClassifier(n_neighbors=10)
23     elif model == "tree":
24         return DecisionTreeClassifier(random_state=42)
25     elif model == "neuralnet":
26         return MLPClassifier(hidden_layer_sizes=10)
27     else:
28         print("Unknown model selected. The choices are: ")
29         print("gnb: Gaussian Naive Bayes")
30         print("svc: Linear Support Vector Classification")
31         print("knn: K-neighbors")
32         print("tree: Decision tree")
33         print("neuralnet: MLP Classifier")
34         exit()
35
36
37 def predict_data(data, target, model):
38     model = choose_model(model)
39     if model == "knn":
40         data = scale(data)
41     accuracy_scores = []
42     confusion_matrices = []
43     auc = []
44     for train_index, test_index in split_k_sets(data):
45         model.fit(data.iloc[train_index], target.iloc[train_index])
46         prediction = model.predict(data.iloc[test_index])
47         accuracy_scores.append(accuracy_score(target.iloc[test_index],
48         prediction))
49         confusion_matrices.append(confusion_matrix(target.iloc[
49         test_index], prediction))
50         auc.append(roc_auc_score(target.iloc[test_index], prediction))
51     cv_score = cross_val_score(model, data, target, cv=10)
52     evaluate_performance(
53         confusion_matrix=mean(confusion_matrices, axis=0),
54         accuracy=mean(accuracy_scores),
55         cv_score=mean(cv_score),
56         auc=mean(auc),
57     )
58

```

Resultados obtenidos

Naives Bayes

Los resultados que obtenemos son los siguientes:

```
P1 master python src/processing.py drop gnb
Accuracy Score: 0.8182178907065785
Confusion matrix:
[[68.2 19.2]
 [11.6 70.4]]
Cross validation score: 0.8098459383753502
AUC: 0.8189853261106881

P1 master python src/processing.py fill gnb
Accuracy Score: 0.80330310880629
Confusion matrix:
[[79.6 23.6]
 [14.2 74.8]]
Cross validation score: 0.8085481099656358
AUC: 0.8062409534117073
```

Figura 1: Naive Bayes

Linear SVC

Los resultados que obtenemos son los siguientes:

```
Accuracy Score: 0.8028611207796729
Confusion matrix:
[[73.4 14. ]
 [19.4 62.6]]
Cross validation score: 0.774593837535014
AUC: 0.8025099198242861
```

Figura 2: Linear SVC con eliminación

```
Accuracy Score: 0.7762359671848014
Confusion matrix:
[[70.6 32.6]
 [18.4 78.6]]
Cross validation score: 0.7137242268041237
AUC: 0.775259773350704
```

Figura 3: Linear SVC con imputación

KNN

Antes de ejecutar este algoritmo, normalizamos los datos dado que el *KNN* es un algoritmo basado en distancia.

Los resultados que obtenemos son los siguientes:

```
P1 master python src/processing.py drop knn
Accuracy Score: 0.8029028889662374
Confusion matrix:
[[70.2 17.2]
 [16.2 65.8]]
Cross validation score: 0.7945378151260505
AUC: 0.8015792049700469

P1 master python src/processing.py fill knn
Accuracy Score: 0.7918609671848014
Confusion matrix:
[[84.2 19. ]
 [21.  68. ]]
Cross validation score: 0.803382731958763
AUC: 0.7910466994240934
```

Figura 4: KNN

Árbol de decisión

Los resultados que obtenemos son los siguientes:

```
P1 master python src/processing.py drop tree
Accuracy Score: 0.761545422902889
Confusion matrix:
[[71.8 15.6]
 [24.8 57.2]]
Cross validation score: 0.763795518207283
AUC: 0.7583995659538703

P1 master python src/processing.py fill tree
Accuracy Score: 0.7700129533670757
Confusion matrix:
[[84.4 18.8]
 [25.4 63.6]]
Cross validation score: 0.7606743986254295
AUC: 0.7649776842324315
```

Figura 5: Árbol de decisión

Perceptrón multicapa

Los resultados que obtenemos son los siguientes:

```
Accuracy Score: 0.7663557257222415
Confusion matrix:
[[66.  21.4]
 [18.2 63.8]]
Cross validation score: 0.7863725490196077
AUC: 0.7625865698185432
```

Figura 6: Perceptrón multicapa con eliminación

```
Accuracy Score: 0.7814604922279702
Confusion matrix:
[[77.  26.2]
 [15.8 73.2]]
Cross validation score: 0.7846327319587629
AUC: 0.7842896451911407
```

Figura 7: Perceptrón multicapa con imputación

Análisis de resultados

Tras ejecutar todos los algoritmos elegidos, podemos analizar los resultados obtenidos.

Observamos que la imputación de la media en los valores nulos no es una estrategia que aporte demasiados beneficios, y según ciertas métricas puede ser hasta dañino para nuestro modelo.

Vemos que nuestros modelos obtienen unos resultados bastante correctos, con un *accuracy score*, *cross validation score* y *AUC* en el intervalo [0,7, 0,82].

Por lo tanto, concluimos que nuestro problema de clasificación es suficientemente genérico para ser resuelto por distintos modelos de aprendizaje, sin variaciones significantes en el rendimientos de éstos.