practica3

February 15, 2021

1 Práctica 3: Clasificación con scikit-learn

1.1 Importar módulos

[152]:
```
%matplotlib inline

import pandas as pd
import numpy as np
import matplotlib
import matplotlib.pyplot as plt

import sklearn
from sklearn import preprocessing, model_selection, neighbors, svm, tree
```

1.2 1. Seleccionar datasets

Los conjuntos de datos seleccionados se encuentran en el directorio `datasets`.

[215]:
```
accent_df = pd.read_csv('datasets/accent.csv', header=0)
avila_df = pd.read_csv('datasets/avila.csv', header=0)
cancer_df = pd.read_csv('datasets/cancer.csv', header=0, na_values='?')
digits_df = pd.read_csv('datasets/digits.csv', header=None)
fertility_df = pd.read_csv('datasets/fertility.csv', header=0)
glass_df = pd.read_csv('datasets/glass.csv', header=0)
iris_df = pd.read_csv('datasets/iris.csv', header=None)
column_df = pd.read_csv('datasets/column.csv', header=0)
phishing_df = pd.read_csv('datasets/phishing.csv', header=None)
wine_df = pd.read_csv('datasets/wine.csv', header=0)
```

1.2.1 Preprocesamiento

[216]:
```
accent_df = accent_df[accent_df.columns.to_list()[1:]+[accent_df.columns.
↪to_list()[0]]]
```

[217]:
```
cancer_df = cancer_df.drop('id', axis=1)
```

[218]:
```
cancer_df.bare_nuclei = cancer_df.bare_nuclei.fillna(np.mean(cancer_df.
↪bare_nuclei)).astype('int')
```

[219]:
```
dfs = [accent_df, avila_df, cancer_df, digits_df, fertility_df, glass_df,␣
↪iris_df, column_df, phishing_df, wine_df]
df_names = ['Accents', 'Avila', 'Cancer', 'Digits', 'Fertility', 'Glass',␣
↪'Iris', 'Vertebral Column', 'Phishing', 'Wine']
```

[220]:
```
for df in dfs:
    df.iloc[:,-1] = df.iloc[:,-1].astype('category')
```

[221]:
```
X = [df.iloc[:, :-1].values for df in dfs]
y = [df.iloc[:, -1].values for df in dfs]
```

1.3 2. Seleccionar clasificadores

Usaremos los clasificadores siguientes: - `sklearn.neighbors.KNeighborsClassifier` - `sklearn.tree.DecisionTreeClassifier` - `sklearn.svm.SVC`

[247]:
```
knn = neighbors.KNeighborsClassifier()
dtree = tree.DecisionTreeClassifier()
svc = svm.SVC()
```

1.4 3. Entrenar los modelos

Usaremos el método *hold out* con porcentajes 70% entrenamiento y 30% test. Realizaremos 5 entrenamientos distintos con cada dataset.

[262]:
```
knn_scores_train = np.zeros([len(dfs),5])
knn_scores_test = np.zeros([len(dfs),5])

dtree_scores_train = np.zeros([len(dfs),5])
dtree_scores_test = np.zeros([len(dfs),5])

svc_scores_train = np.zeros([len(dfs),5])
svc_scores_test = np.zeros([len(dfs),5])



for i in range(len(dfs)):
    for j in range(5):
        X_train, X_test, y_train, y_test = model_selection.
↪train_test_split(X[i], y[i], test_size=.3)
```

```
        sc = preprocessing.MinMaxScaler()
        sc.fit_transform(X_train)
        sc.transform(X_test)

        knn.fit(X_train, y_train)
        dtree.fit(X_train, y_train)
        svc.fit(X_train, y_train)

        knn_scores_train[i][j] = knn.score(X_train, y_train)*100
        knn_scores_test[i][j] = knn.score(X_test, y_test)*100

        dtree_scores_train[i][j] = dtree.score(X_train, y_train)*100
        dtree_scores_test[i][j] = dtree.score(X_test, y_test)*100

        svc_scores_train[i][j] = svc.score(X_train, y_train)*100
        svc_scores_test[i][j] = svc.score(X_test, y_test)*100
```

[263]:

```
print('KNN Scores')
print('==========')
for i in range(len(dfs)):
    print(f'{df_names[i]}:')
    print(f'\tTrain -> {np.mean(knn_scores_train[i]):.2f} with std. {np.
 ↪std(knn_scores_train[i]):.2f}')
    print(f'\tTest -> {np.mean(knn_scores_test[i]):.2f} with std. {np.
 ↪std(knn_scores_test[i]):.2f}')
```

```
KNN Scores
==========
Accents:
        Train -> 88.43 with std. 1.52
        Test -> 77.58 with std. 3.40
Avila:
        Train -> 81.24 with std. 0.41
        Test -> 70.82 with std. 0.56
Cancer:
        Train -> 97.71 with std. 0.40
        Test -> 96.48 with std. 0.65
Digits:
        Train -> 99.09 with std. 0.09
        Test -> 98.45 with std. 0.14
Fertility:
        Train -> 88.00 with std. 1.46
        Test -> 85.33 with std. 3.40
```

```
Glass:
	Train -> 76.11 with std. 1.38
	Test -> 66.77 with std. 2.50
Iris:
	Train -> 97.33 with std. 1.11
	Test -> 96.00 with std. 0.89
Vertebral Column:
	Train -> 88.57 with std. 1.47
	Test -> 85.38 with std. 3.09
Phishing:
	Train -> 91.30 with std. 0.81
	Test -> 85.71 with std. 1.82
Wine:
	Train -> 65.06 with std. 1.27
	Test -> 49.33 with std. 2.02
```

[264]:
```python
print('Decision Tree Scores')
print('====================')
for i in range(len(dfs)):
    print(f'{df_names[i]}:')
    print(f'\tTrain -> {np.mean(dtree_scores_train[i]):.2f} with std. {np.
↪std(dtree_scores_train[i]):.2f}')
    print(f'\tTest -> {np.mean(dtree_scores_test[i]):.2f} with std. {np.
↪std(dtree_scores_test[i]):.2f}')
```

```
Decision Tree Scores
====================
Accents:
	Train -> 100.00 with std. 0.00
	Test -> 61.82 with std. 5.51
Avila:
	Train -> 100.00 with std. 0.00
	Test -> 93.81 with std. 0.71
Cancer:
	Train -> 100.00 with std. 0.00
	Test -> 94.67 with std. 0.76
Digits:
	Train -> 100.00 with std. 0.00
	Test -> 88.96 with std. 1.02
Fertility:
	Train -> 99.43 with std. 0.70
	Test -> 82.00 with std. 3.40
Glass:
	Train -> 100.00 with std. 0.00
	Test -> 65.85 with std. 7.93
Iris:
	Train -> 100.00 with std. 0.00
	Test -> 95.56 with std. 2.81
```

```
Vertebral Column:
	Train -> 100.00 with std. 0.00
	Test -> 79.57 with std. 2.63
Phishing:
	Train -> 96.60 with std. 0.26
	Test -> 87.24 with std. 1.83
Wine:
	Train -> 100.00 with std. 0.00
	Test -> 58.92 with std. 1.11
```

[265]:
```python
print('SVM Scores')
print('==========')
for i in range(len(dfs)):
    print(f'{df_names[i]}:')
    print(f'\tTrain -> {np.mean(svc_scores_train[i]):.2f} with std. {np.
↪std(svc_scores_train[i]):.2f}')
    print(f'\tTest -> {np.mean(svc_scores_test[i]):.2f} with std. {np.
↪std(svc_scores_test[i]):.2f}')
```

```
SVM Scores
==========
Accents:
	Train -> 68.17 with std. 2.45
	Test -> 61.62 with std. 3.44
Avila:
	Train -> 72.85 with std. 0.08
	Test -> 70.24 with std. 0.31
Cancer:
	Train -> 97.42 with std. 0.38
	Test -> 96.76 with std. 0.92
Digits:
	Train -> 99.44 with std. 0.08
	Test -> 98.80 with std. 0.28
Fertility:
	Train -> 87.14 with std. 1.28
	Test -> 90.00 with std. 2.98
Glass:
	Train -> 37.05 with std. 2.30
	Test -> 33.54 with std. 4.90
Iris:
	Train -> 97.33 with std. 0.71
	Test -> 98.22 with std. 0.89
Vertebral Column:
	Train -> 85.25 with std. 2.29
	Test -> 84.73 with std. 2.19
Phishing:
	Train -> 89.31 with std. 1.03
	Test -> 85.96 with std. 0.93
```

```
Wine:
        Train -> 51.01 with std. 1.00
        Test -> 49.29 with std. 2.06
```

[268]:

```
average_knn_scores = [np.mean(x) for x in knn_scores_test]
knn_score_std = [np.std(x) for x in knn_scores_test]

average_dtree_scores = [np.mean(x) for x in dtree_scores_test]
dtree_score_std = [np.std(x) for x in dtree_scores_test]

average_svc_scores = [np.mean(x) for x in svc_scores_test]
svc_score_std = [np.std(x) for x in svc_scores_test]
```

[282]:

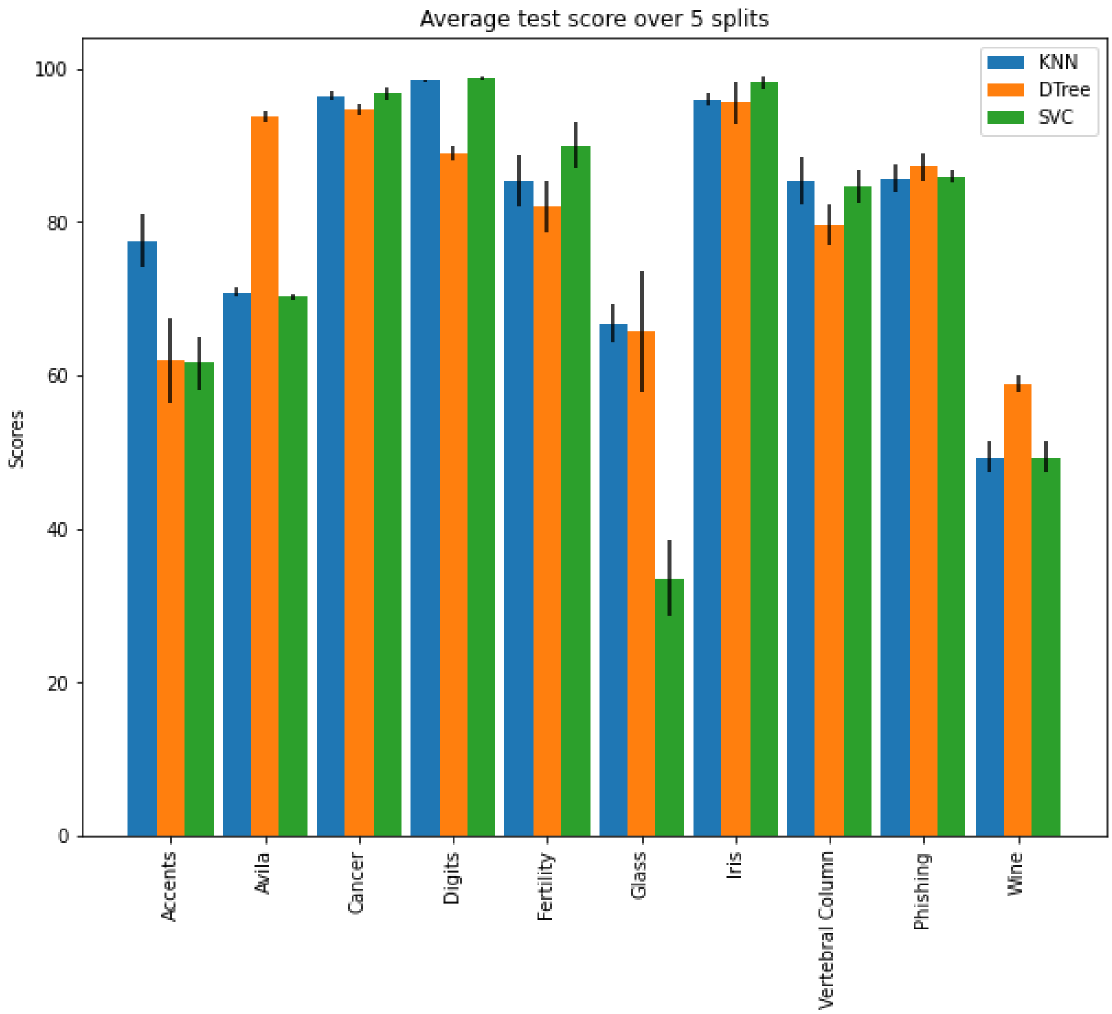
```
x = np.arange(len(dfs)*2, step=2)  # the label locations
width = 0.6  # the width of the bars

fig, ax = plt.subplots(figsize=(10,8))

ax.bar(x - width, average_knn_scores, width, label='KNN', yerr=knn_score_std)
ax.bar(x, average_dtree_scores, width, label='DTree', yerr=dtree_score_std)
ax.bar(x + width, average_svc_scores, width, label='SVC', yerr=svc_score_std)

# Add some text for labels, title and custom x-axis tick labels, etc.
ax.set_ylabel('Scores')
ax.set_title('Average test score over 5 splits')
ax.set_xticks(x)
ax.set_xticklabels(df_names, rotation='vertical')
ax.legend()

plt.show()
```



1.5 4. Métodos de comparación y evaluación de algoritmos

Aplicaremos el test de Wilcoxon a los algoritmos KNN y SVM.

[315]:
```
winner = []
difference = []

for i in range(len(dfs)):
    diff = average_knn_scores[i]-average_dtree_scores[i]
    difference.append(
        (abs(diff), i)
    )
    w = 1 if diff > 0 else -1
    winner.append(w)
```

```
difference.sort(key=lambda x: x[0])

r_plus = 0
r_minus = 0
for i, d in enumerate(difference):
    if winner[d[1]] == 1:
        r_plus += i+1
    else:
        r_minus += i+1
```

Mirando en la tabla del test de Wilcoxon, para $N = 10$ datasets y $\alpha = 0.05$, el menor de R^+ y R^- debe ser menor o igual que 8 para que la diferencia entre los clasificadores sea significativa.

[316]:
```
print(min(r_plus, r_minus))
```

```
21
```

Viendo que no lo es, podemos afirmar que no hay una diferencia de rendimiento entre ellos.

1.6 6. Aplicar GridSearch a un clasificador

Lo haremos con KNN, variando el número de vecinos K, el peso de los votos entre uniforme y basado en distancia, y la métrica de distancia (euclídea o manhattan)

[330]:
```
knn_gc_scores_train = np.zeros(len(dfs))
knn_gc_scores_test = np.zeros(len(dfs))

for i in range(len(dfs)):
        X_train, X_test, y_train, y_test = model_selection.
 ↪train_test_split(X[i], y[i], test_size=.3)
        sc = preprocessing.MinMaxScaler()
        sc.fit_transform(X_train)
        sc.transform(X_test)

        Ks = np.arange(start=3, stop=12, step=2)
        weigths = ['uniform', 'distance']
        ps = [1, 2]

        optimal = model_selection.GridSearchCV(
            estimator=neighbors.KNeighborsClassifier(),
            param_grid=dict(n_neighbors=Ks, weights=weigths, p=ps),
            cv=3)

        optimal.fit(X_train, y_train)


        knn_gc_scores_train[i] = optimal.score(X_train, y_train)*100
```

```
        knn_gc_scores_test[i] = optimal.score(X_test, y_test)*100
```

[346]:
```
print('KNN with GridSearch Scores')
print('==========================')
for i in range(len(dfs)):
    print(f'{df_names[i]}:')
    print(f'\tTrain -> {knn_gc_scores_train[i]:.2f}')
    print(f'\tTest -> {knn_gc_scores_test[i]:.2f}')
    print(f'\t{knn_gc_scores_test[i] - average_knn_scores[i]:.2f} variation wrt␣
↪to default parameters')
print(f'\nAverage variation = {np.mean(knn_gc_scores_test - average_knn_scores):
↪.2f}')
```

```
KNN with GridSearch Scores
==========================
Accents:
        Train -> 100.00
        Test -> 83.84
        6.26 variation wrt to default parameters
Avila:
        Train -> 100.00
        Test -> 82.18
        11.36 variation wrt to default parameters
Cancer:
        Train -> 100.00
        Test -> 97.62
        1.14 variation wrt to default parameters
Digits:
        Train -> 100.00
        Test -> 99.05
        0.60 variation wrt to default parameters
Fertility:
        Train -> 88.57
        Test -> 83.33
        -2.00 variation wrt to default parameters
Glass:
        Train -> 100.00
        Test -> 64.62
        -2.15 variation wrt to default parameters
Iris:
        Train -> 100.00
        Test -> 93.33
        -2.67 variation wrt to default parameters
Vertebral Column:
        Train -> 100.00
        Test -> 89.25
        3.87 variation wrt to default parameters
```

```
Phishing:
        Train -> 97.36
        Test -> 87.93
        2.22 variation wrt to default parameters
Wine:
        Train -> 100.00
        Test -> 62.29
        12.96 variation wrt to default parameters

Average variation = 3.16
```

Vemos que en general, los resultados son mucho mejores, con un incremento medio del CCR de un 3% frente a los parámetros por defecto, que llega hasta el 13% en algunos casos.