

НАЦИОНАЛЬНЫЙ ИССЛЕДОВАТЕЛЬСКИЙ ЯДЕРНЫЙ УНИВЕРСИТЕТ
«МИФИ»

Экзаменационная работа
по курсу: «Специальные технологии баз данных»

Выполнил: студент группы С20-702

Нуритдинходжаева А.

(подпись)

(Фамилия И.О.)

Проверил:

Манаенкова Т.А.

(оценка)

(подпись)

(Фамилия И.О.)

Условия заданий

Вариант 1

Задания

1. Загрузите с сайта <https://sci2s.ugr.es/keel/datasets.php> набор статистических данных, указанный в вашем варианте. Разберитесь, какие данные приведены в наборе и какой атрибут является меткой класса.
2. На основе загруженного файла создайте Pandas DataFrame, подобрав правильные типы данных столбцов.
3. Выполните стандартизацию полученного дата фрейма.
4. Разделите дата фрейм на обучающую, тестовую и валидационную выборки в соотношении 5 / 3 / 2.
5. На основе обучающей и тестовой выборки постройте MLP-классификатор. Рассчитайте параметры его эффективности (Accuracy, Precision, Recall, ROC-AUC). Меняя значение параметров классификатора обосновано подберите наиболее удачную MLP-сеть.
6. На основе обучающей и тестовой выборки постройте дерево решений, рассчитайте параметры эффективности классификатора (Accuracy, Precision, Recall, ROC-AUC). Меняя значение параметра альфа ([0.005, 0.01, 0.015, 0.02, 0.025, 0.03, 0.035, 0.2, 0.8]) и критерий классификации ([Entropy, Gini]) обосновано подберите наиболее удачное дерево классификации для подготовленных выборок.
7. На основе обучающей и тестовой выборки постройте SVM-классификатор, рассчитайте параметры эффективности классификатора (Accuracy, Precision, Recall, ROC-AUC). Меняя значение параметров kernel, gamma, coef0, degree, C (на основе вариантов, представленных в лекции 1 второго семестра) обосновано подберите наиболее удачное дерево классификации для подготовленных выборок.
8. Используя валидационную выборку рассчитайте для лучшего MLP-классификатора, дерева решений и лучшего SVM-классификатора параметры эффективности (Accuracy, Precision, Recall, ROC-AUC). Обоснуйте какой из двух классификаторов и когда лучше.

Решение

```
import pandas as pd
from sklearn import preprocessing
from sklearn.model_selection import train_test_split
import graphviz
import os
from sklearn import tree
import numpy as np
```

```
import matplotlib.pyplot as plt
from sklearn.metrics import accuracy_score, precision_score, recall_score,
roc_auc_score
from sklearn import svm
import sklearn
!pip install ydata-profiling
!pip install pandas-profiling
from ydata_profiling import ProfileReport
from sklearn.model_selection import ParameterGrid
from sklearn.metrics import roc_curve, auc, roc_auc_score
from sklearn import metrics, tree
from sklearn.preprocessing import label_binarize
from sklearn.multiclass import OneVsRestClassifier
```

№2

```
f = open('winequality-red.dat', 'r')
skip_rows = 0
COLUMNS = []
for line in f:
    if line[0] == '@':
        string = line.split(' ')
        if string[0] == '@attribute':
            COLUMNS.append(string[1])
            skip_rows += 1
COLUMNS = ['FixedAcidity', 'VolatileAcidity', 'CitricAcid', 'ResidualSugar',
'Chlorides', 'FreeSulfurDioxide', 'TotalSulfurDioxide',
'Density', 'PH', 'Sulphates', 'Alcohol', 'Quality']
f.close()

df = pd.read_table('winequality-red.dat', skiprows=skip_rows, sep=',',
names=COLUMNS)

#числовой тип данных
numeric_columns = ['FixedAcidity', 'VolatileAcidity', 'CitricAcid',
'ResidualSugar', 'Chlorides', 'FreeSulfurDioxide', 'TotalSulfurDioxide',
'Density', 'PH', 'Sulphates', 'Alcohol', 'Quality']
df[numeric_columns] = df[numeric_columns].apply(pd.to_numeric)

df = pd.read_table('winequality-red.dat', skiprows=skip_rows, sep=',',
names=COLUMNS)

pd.set_option('display.max_columns', 2000)
pd.set_option('display.width', 20000)

print(df)
```

№3

```
scaler_std = preprocessing.StandardScaler()
```

```
x = scaler_std.fit_transform(df[['FixedAcidity', 'VolatileAcidity',
                                'CitricAcid', 'ResidualSugar', 'Chlorides', 'FreeSulfurDioxide',
                                'TotalSulfurDioxide', 'Density', 'PH',
                                'Sulphates', 'Alcohol']])
df[['FixedAcidity', 'VolatileAcidity', 'CitricAcid', 'ResidualSugar',
    'Chlorides', 'FreeSulfurDioxide', 'TotalSulfurDioxide', 'Density', 'PH',
    'Sulphates', 'Alcohol']] = x

print(df)
```

№4

```
train, temp = train_test_split(df, test_size=0.5, random_state=42)
test, val = train_test_split(temp, test_size=0.4, random_state=42)
train.reset_index()
test = test.reset_index()

train_x = train.drop(['Quality'], axis = 1)
train_y = train['Quality']
test_x = test.drop(['index', 'Quality'], axis = 1)
test_y = test['Quality']
val_x = val.drop(['Quality'], axis = 1)
val_y = val['Quality']

print("Обучающая выборка:", '\n', train_x, '\n', train_y, '\n')
print("Тестовая выборка:", '\n', test_x, '\n', test_y, '\n')
print("Валидационная выборка:", '\n', val_x, '\n', val_y)
```

№5

```
#MLP
from sklearn.neural_network import MLPClassifier

parameters={'max_iter':[300, 100], "activation":['identity', "logistic",
"tanh", "relu"],"solver":["lbfgs", "sgd", "adam"],
            "alpha":[0.0001,0.00001,0.001]}
mlp = MLPClassifier()
param_grid = ParameterGrid(parameters)
results=[]
for params in param_grid:
    mlp.set_params(**params)
    mlp.fit(train_x, train_y)
    y_pred = mlp.predict(test_x)

    accuracy, recall, precision = par_metrics(test_y, y_pred)
    results.append({'params': params, 'accuracy': accuracy, 'recall':
recall, 'precision': precision})
```

```
results
```

```
results_mlp_df=pd.DataFrame(data=results)
```

```
results_mlp_df
```

```
results_mlp_df=pd.DataFrame(data=results)
for index, row in results_mlp_df.iterrows():
    results_mlp_df.at[index, 'max_iter'] = row['params']["max_iter"]
    results_mlp_df.at[index, 'activation'] = row['params']["activation"]
    results_mlp_df.at[index, 'solver'] = row['params']["solver"]
    results_mlp_df.at[index, 'alpha'] = row['params']["alpha"]
    results_mlp_df.at[index, 'total'] =
row['accuracy']+row["recall"]+row['recall']
results_mlp_df.drop('params',axis=1,inplace=True)
results_mlp_df
```

```
results_mlp_df.loc[results_mlp_df.total==max(results_mlp_df.total)]
```

```
#ROC-AUC
```

```
max_=results_mlp_df.iloc[results_mlp_df.total.idxmax()]

y_train_bin = label_binarize(train_y, classes=mlp.classes_)
y_test_bin = label_binarize(test_y, classes=mlp.classes_)
classifier = OneVsRestClassifier(mlp.set_params(max_iter=max_.max_iter,
activation=max_.activation, solver=max_.solver, alpha=max_.alpha))
y_score = classifier.fit(train_x, y_train_bin).decision_function(test_x)

roc_auc_ovr()
```

№6

```
clf = tree.DecisionTreeClassifier(criterion= 'gini', random_state = 42,
ccp_alpha = 0.01)
clf.fit(train_x, train_y)

tree.plot_tree(clf, filled = True, rounded = True)
plt.show()

gini_0005 = clf.score(test_x, test_y)
gini_0005*100
clf = tree.DecisionTreeClassifier(criterion= 'entropy', random_state = 42,
ccp_alpha = 0.03)
clf.fit(train_x, train_y)

tree.plot_tree(clf, filled = True, rounded = True)
plt.show()

gini_0005 = clf.score(test_x, test_y)
gini_0005*100
#параметры эффективности
parameters = {'criterion':('gini', 'entropy'), 'ccp_alpha':[0,0.005, 0.01,
0.015, 0.02, 0.025, 0.03, 0.035, 0.2, 0.8]}
param_grid = ParameterGrid(parameters)
```

```

results = []

def par_metrics(t,p):
    accuracy = metrics.accuracy_score(t, p)
    recall = metrics.recall_score(t, p, average='weighted')
    precision = metrics.precision_score(t, p, average='weighted')
    return accuracy, recall, precision

for params in param_grid:
    clf.set_params(**params)
    clf.fit(train_x, train_y)
    y_pred = clf.predict(test_x)

    accuracy, recall, precision = par_metrics(test_y, y_pred)
    results.append({'params': params, 'accuracy': accuracy, 'recall':
recall, 'precision': precision})

```

```

results_df=pd.DataFrame(data=results)
for index, row in results_df.iterrows():
    results_df.at[index, 'criterion'] = row['params']['criterion']
    results_df.at[index, 'ccp_alpha'] = row['params']['ccp_alpha']
    results_df.at[index, 'total'] =
row['accuracy']+row['recall']+row['precision']
results_df.drop('params',axis=1,inplace=True)
results_df

```

```

results_df.loc[results_df.total==max(results_df.total)]

```

```

def roc_auc_ovr():
    fpr = dict()
    tpr = dict()
    roc_auc = dict()
    for i in range(len(y_train_bin[1])):
        fpr[i], tpr[i], _ = roc_curve(y_test_bin[:, i], y_score[:, i])
        roc_auc[i] = auc(fpr[i], tpr[i])
        x = [0, 1]
        y = [0, 1]
        plt.plot(x, y, color='red', linestyle='--', label='Diagonal Line')
        plt.plot(fpr[i], tpr[i], label=f"AUC ({params}): {roc_auc[i]:.2f}")
        plt.xlabel('False Positive Rate')
        plt.ylabel('True Positive Rate')
        plt.title(f'ROC Curve for class {clf.classes_[i]}')
        plt.show()

    # Считаем микросреднее TPR и FPR
    fpr["micro"], tpr["micro"], _ = roc_curve(y_test_bin.ravel(),
y_score.ravel())
    # Простое среднее AUC
    roc_auc["avg"]=sum(roc_auc.values()) / len(roc_auc)
    # Первый способ микросреднего AUC

```

```

roc_auc["micro"] = auc(fpr["micro"], tpr["micro"])
#Второй способ микросреднего
roc_auc["micro_aux"] = roc_auc_score(y_test_bin, y_score, average='micro',
multi_class='ovr')
print(f'Микроустредненный AUC первым методом {roc_auc["micro"]}, м-й AUC
вторым методом {roc_auc["micro_aux"]} и средний AUC {roc_auc["avg"]}')
return roc_auc

```

```

#ROC-AUC
max_=results_df.iloc[results_df.total.idxmax()]

y_train_bin = label_binarize(train_y, classes=clf.classes_)
y_test_bin = label_binarize(test_y, classes=clf.classes_)
classifier =
OneVsRestClassifier(clf.set_params(criterion=max_.criterion,ccp_alpha=max_.c
cp_alpha))
y_score = classifier.fit(train_x, y_train_bin).predict_proba(test_x)

roc_auc_ovr()

```

№7

```

#SVM
parameters = {'kernel':['rbf'], 'gamma':['auto'],'coef0':[0, 1,
2], 'degree':[2], 'C':[0.1, 0.5, 1]}

cl = svm.SVC()
param_grid = ParameterGrid(parameters)
results = []

for params in param_grid:
    cl.set_params(**params)
    cl.fit(train_x, train_y)
    y_pred = cl.predict(test_x)

    accuracy, recall, precision = par_metrics(test_y, y_pred)
    results.append({'params': params, 'accuracy': accuracy, 'recall':
recall, 'precision': precision})

```

```

results_svm_df=pd.DataFrame(data=results)
for index, row in results_svm_df.iterrows():
    results_svm_df.at[index, 'kernel'] = row['params']["kernel"]
    results_svm_df.at[index, 'gamma'] = row['params']["gamma"]
    results_svm_df.at[index, 'coef0'] = row['params']["coef0"]
    results_svm_df.at[index, 'degree'] = row['params']["degree"]
    results_svm_df.at[index, 'C'] = row['params']["C"]
    results_svm_df.at[index, 'total'] =
row['accuracy']+row['recall']+row['precision']
results_svm_df['degree']=results_svm_df['degree'].astype(int)
results_svm_df.drop('params',axis=1,inplace=True)

```

```
results_svm_df
```

```
results_svm_df.loc[results_svm_df.total==max(results_svm_df.total)]
```

```
#ROC-AUC
```

```
max_=results_svm_df.iloc[results_svm_df.total.idxmax()]
```

```
y_train_bin = label_binarize(train_y, classes=cl.classes_)
```

```
y_test_bin = label_binarize(test_y, classes=cl.classes_)
```

```
classifier =
```

```
OneVsRestClassifier(cl.set_params(kernel=max_.kernel,gamma=max_.gamma,coef0=
max_.coef0,degree=max_.degree,C=max_.C))
```

```
y_score = classifier.fit(train_x, y_train_bin).decision_function(test_x)
```

```
roc_auc_ovr()
```

№8

```
max_tree=results_df.iloc[results_df.total.idxmax()]
```

```
clf.set_params(criterion=max_tree.criterion,ccp_alpha=max_tree.ccp_alpha)
```

```
clf.fit(train_x, train_y)
```

```
y_pred_tree = clf.predict(val_x)
```

```
#SVM
```

```
max_svm=results_svm_df.iloc[results_svm_df.total.idxmax()]
```

```
cl.set_params(kernel=max_svm.kernel,gamma=max_svm.gamma,coef0=max_svm.coef0,
degree=max_svm.degree,C=max_svm.C)
```

```
cl.fit(train_x, train_y)
```

```
y_pred_svm = cl.predict(val_x)
```

```
#MLP
```

```
max_mlp=results_mlp_df.iloc[results_mlp_df.total.idxmax()]
```

```
cl.set_params(max_iter=max_.max_iter, activation=max_.activation,
solver=max_.solver, alpha=max_.alpha)
```

```
cl.fit(train_x, train_y)
```

```
y_pred_mlp = cl.predict(val_x)
```

```
bests=[]
```

```
for pred in [y_pred_tree,y_pred_svm]:
```

```
    accuracy, recall, precision = par_metrics(val_y, pred)
```

```
    bests.append({'accuracy': accuracy, 'recall': recall, 'precision':
precision})
```

```
bests_df=pd.DataFrame(data=bests)
```

```
for index, row in bests_df.iterrows():
```

```
#ROC-AUC
```

```
y_train_bin = label_binarize(train_y, classes=val_y.unique())
```

```
y_test_bin = label_binarize(val_y, classes=val_y.unique())
```

```
roc_auc_best=[]
```



```
classifier =  
OneVsRestClassifier(clf.set_params(criterion=max_tree.criterion, ccp_alpha=ma  
x_tree.ccp_alpha))  
y_score = classifier.fit(train_x, y_train_bin).predict_proba(val_x)  
roc_auc_best.append(roc_auc_ovr())  
classifier =  
OneVsRestClassifier(cl.set_params(kernel=max_svm.kernel, gamma=max_svm.gamma,  
coef0=max_svm.coef0, degree=max_svm.degree, C=max_svm.C))  
y_score = classifier.fit(train_x, y_train_bin).decision_function(val_x)  
roc_auc_best.append(roc_auc_ovr())  
  
bests_df.at[index, 'total'] = row['accuracy']+row["recall"]+row['recall']  
bests_df
```

```
auc_best_df=pd.DataFrame(data=roc_auc_best)  
auc_best_df
```