НАЦИОНАЛЬНЫЙ ИССЛЕДОВАТЕЛЬСКИЙ ЯДЕРНЫЙ УНИВЕРСИТЕТ «МИФИ»

Отчет по лабораторной работе №3

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.)

Москва 2024 г

Условие задания

Вариант 3

Задание 1

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 каой 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 с применением стратификации.
5. На основе обучающей и тестовой выборки постройте дерево решений. 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]) подберите наиболее удачное по макро усреднённому параметру ROC-AUC дерево классификации для подготовленных выборок.
6. На основе обучающей и тестовой выборки постройте SVM-классификатор. Меняя значение параметров kernel, gamma, coef0, degree, C (на основе вариантов, представленных в лекции 1 второго семестра) обосновано подберите наиболее удачное дерево по макро усреднённому параметру ROC-AUC классификации для подготовленных выборок.
7. На основе обучающей и тестовой выборки постройте Random Forest-классификатор. Меняя значение параметра критерий классификации ([Entropy, Gini]), а также число генерируемых деревьев и число используемых полей подберите наиболее удачный по макро усреднённому параметру ROC-AUC лес для подготовленных выборок.
8. Выполните обогащение выборки и повторите шаги 5, 6, 7. Сравните с помощью ROC-AUC-критерия и валидационной выборки, полученные 6 классификаторов и выберите лучший.

Решение

```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
from sklearn.model_selection import train_test_split
from sklearn import tree
from sklearn.model_selection import ParameterGrid
from sklearn.metrics import roc_curve, auc, roc_auc_score
from sklearn import svm
from sklearn import preprocessing
from sklearn.metrics import classification_report
from sklearn.ensemble import RandomForestClassifier
from sklearn.datasets import make_classification
import re
import seaborn as sns
from imblearn.over_sampling import RandomOverSampler

f = open('wisconsin.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 = ['ClumpThickness', 'CellSize', 'CellShape', ' MarginalAdhesion', 'EpithelialSize', 'BareNuclei',
'BlandChromatin',
         'NormalNucleoli', 'Mitoses', 'Class']
f.close()

df1 = pd.read_table('wisconsin.dat', skiprows=skip_rows, sep=',', names=COLUMNS)

#числовой тип данных
numeric_columns = ['ClumpThickness', 'CellSize', 'CellShape', ' MarginalAdhesion', 'EpithelialSize',
'BareNuclei', 'BlandChromatin',
         'NormalNucleoli', 'Mitoses']
df1[numeric_columns] = df1[numeric_columns].apply(pd.to_numeric)

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

print(df1)

scaler_std = preprocessing.StandardScaler()
x = scaler_std.fit_transform(df1[['ClumpThickness', 'CellSize', 'CellShape', ' MarginalAdhesion',
'EpithelialSize', 'BareNuclei',
                                 'BlandChromatin', 'NormalNucleoli', 'Mitoses']])
df1[['ClumpThickness', 'CellSize', 'CellShape', ' MarginalAdhesion', 'EpithelialSize', 'BareNuclei',
'BlandChromatin', 'NormalNucleoli',
     'Mitoses']] = x

print(df1)

train, temp = train_test_split(df1, test_size=0.5, random_state=2222)
test, val = train_test_split(temp, test_size=0.4, random_state=2222)
```

```
train.reset_index()
test = test.reset_index()

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

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)

clf =  tree.DecisionTreeClassifier(criterion= 'gini', random_state = 42, ccp_alpha = 0.005)
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= '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= 'gini', random_state = 42, ccp_alpha = 0.015)
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= 'gini', random_state = 42, ccp_alpha = 0.02)
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= 'gini', random_state = 42, ccp_alpha = 0.025)
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= 'gini', 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

clf =  tree.DecisionTreeClassifier(criterion= 'gini', random_state = 42, ccp_alpha = 0.035)
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= 'gini', random_state = 42, ccp_alpha = 0.2)
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= 'gini', random_state = 42, ccp_alpha = 0.8)
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 = 0.005)
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 = 0.01)
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 = 0.015)
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 = 0.02)
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 = 0.025)
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

clf =  tree.DecisionTreeClassifier(criterion= 'entropy', random_state = 42, ccp_alpha = 0.035)
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 = 0.2)
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 = 0.8)
```

```
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

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]}
tr = tree.DecisionTreeClassifier()
param_grid = ParameterGrid(parameters)
results = []

train_y_ = [1 if label != ' negative' else 0 for label in train_y]
test_y_ = [1 if label != ' negative' else 0 for label in test_y]
val_y_ = [1 if label != ' negative' else 0 for label in val_y]

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

    auc = roc_auc_score(test_y_, y_pred, average='macro')
    results.append({'params': params, 'auc': auc})

results_df = pd.DataFrame(data=results)

# Find the model with the highest AUC score
best_tree_model = results_df.loc[results_df['auc'].idxmax()]

# Print the best model's parameters and AUC score
print("наиболее удачное:")
print(best_tree_model['params'])
print("наиболее удачное по макро усреднённому параметру ROC-AUC :", best_tree_model['auc'])

parameters = {'kernel':['rbf'], 'gamma':['auto'],'coef0':[0, 1, 2, 5, 10, 0.1, 0.01],'degree':[2],'C':[0.001, 0.01,
0.1, 0.5, 1, 2, 10, 20]}

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

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

    auc = roc_auc_score(test_y_, y_pred, average='macro')
    results.append({'params': params, 'auc': auc})

results_svm_df = pd.DataFrame(data=results)

best_model = results_svm_df.loc[results_svm_df['auc'].idxmax()]

print(best_model['params'])
```

```
print("наиболее удачное дерево по макро усреднённому параметру ROC-AUC :", best_model['auc'])

parameters = {'criterion':('gini', 'entropy'), 'max_depth':[1,2,3],
'n_estimators':[50,100,500,1000],"bootstrap":[True],"max_features":["sqrt", "log2", None]}
forest = RandomForestClassifier()
param_grid = ParameterGrid(parameters)
results = []

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

    auc = roc_auc_score(test_y_, y_pred, average='macro')
    results.append({'params': params, 'auc': auc})

results__forest_df = pd.DataFrame(data=results)

# Find the model with the highest AUC score
best_model = results__forest_df.loc[results__forest_df['auc'].idxmax()]

# Print the best model's parameters and AUC score
print("Best Model Parameters:")
print(best_model['params'])
print("наиболее удачный по макро усреднённому параметру ROC-AUC лес:", best_model['auc'])

OverS = RandomOverSampler()
train_x_o, train_y_o = OverS.fit_resample(train_x, train_y_)

tmp=train_x_o
tmp['Class']=train_y_o

plt.subplots(figsize=(30, 30))
c=1
for i in ['ClumpThickness', 'CellSize', 'CellShape', ' MarginalAdhesion', 'EpithelialSize', 'BareNuclei',
'BlandChromatin',
        'NormalNucleoli', 'Mitoses']:
  for j in ['ClumpThickness', 'CellSize', 'CellShape', ' MarginalAdhesion', 'EpithelialSize', 'BareNuclei',
'BlandChromatin',
        'NormalNucleoli', 'Mitoses']:
    plt.subplot(9,9,c)
    sns.scatterplot(data=tmp, x=i, y=j, hue="Class",palette="magma")
    c+=1
plt.show()

train_x_o.drop('Class',axis=1,inplace=True)

#SVM
clf = svm.SVC()
clf.set_params(C=2, coef0= 0, degree=2, gamma= 'auto', kernel='rbf')
clf.fit(train_x_o, train_y_o)
y_pred = clf.predict(val_x)
auc = roc_auc_score(val_y_, y_pred, average='macro')
print("Best Model AUC Score:", auc)

#лес
```

```
forest = RandomForestClassifier()
forest.set_params(bootstrap= True, criterion='gini', max_depth=5, max_features='sqrt', n_estimators= 50)
forest.fit(train_x_o, train_y_o)
y_pred = forest.predict(val_x)
auc = roc_auc_score(val_y_, y_pred, average='macro')
print("Best Model AUC Score:", auc)
```