НАЦИОНАЛЬНЫЙ ИССЛЕДОВАТЕЛЬСКИЙ ЯДЕРНЫЙ УНИВЕРСИТЕТ «МИФИ»

Отчет по лабораторной работе №2

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 1

Задание 1

1. Зарегистрируйтесь на сайте https://www.kaggle.com/datasets и загрузите с него набор статистических данных, указанный в вашем варианте. Разберитесь, какие данные приведены в наборе и каой атрибут является меткой класса.
2. На основе загруженного CSV-файла создайте Pandas DataFrame, подобрав правильные типы данных столбцов.
3. Выполните стандартизацию полученного дата фрейма.
4. Разделите дата фрейм на обучающую и тестовую выборки в соотношении 6 к 4.
5. 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.

Задание 2

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 CSV-файла создайте Pandas DataFrame, подобрав правильные типы данных столбцов.
3. Выполните стандартизацию полученного дата фрейма.
4. Разделите дата фрейм на обучающую и тестовую выборки в соотношении 6 к 4.
5. Подготовьте на основе полученного дата фрейма ещё два: первый – данные после обработки методом главных компонент с критерий 95% значимости, второй – данные после обработки методом главных компонет с критерием Критерий Кайзера.
6. Для каждого из трёх вариантов постройте дерево решений и проанализируйте его эффективность.
7. Обосновано выберите наилучший вариант классификатора.

Решение

Задание 1

```
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 import svm
from sklearn import preprocessing
from sklearn.metrics import classification_report

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

df = pd.read_csv('glass.csv', sep = ',')

numeric_columns = ['RI', 'Na', 'Mg', 'Al', 'Si', 'K', 'Ca', 'Ba', 'Fe', 'Type']
df[numeric_columns] = df[numeric_columns].apply(pd.to_numeric)

print(df)
print(df['Type'].value_counts())

scaler_std = preprocessing.StandardScaler()
x = scaler_std.fit_transform(df[['RI','Na','Mg','Al','Si','K','Ca','Ba','Fe']])
df[['RI','Na','Mg','Al','Si','K','Ca','Ba','Fe']] = x

train_table, test_table = train_test_split(df, test_size = 0.4, random_state = 44, stratify = df['Type'])
test_table = test_table.reset_index()
train_table = train_table.reset_index()
train_x = train_table[['RI','Na','Mg','Al','Si','K','Ca','Ba','Fe']]
train_y = train_table['Type']
test_x = test_table[['RI','Na','Mg','Al','Si','K','Ca','Ba','Fe']]
test_y = test_table['Type']

print(train_y)
print(test_y)

clf =  tree.DecisionTreeClassifier(criterion= 'gini', random_state = 44, 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 = 44, ccp_alpha = 0.01)
clf.fit(train_x, train_y)

tree.plot_tree(clf, filled = True, rounded = True)
```

```
plt.show()

gini_001 = clf.score(test_x, test_y)
gini_001*100

clf =  tree.DecisionTreeClassifier(criterion= 'gini', random_state = 44, ccp_alpha = 0.015)
clf.fit(train_x, train_y)

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

gini_0015 = clf.score(test_x, test_y)
gini_0015*100

clf =  tree.DecisionTreeClassifier(criterion= 'gini', random_state = 44, ccp_alpha = 0.02)
clf.fit(train_x, train_y)

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

gini_002 = clf.score(test_x, test_y)
gini_002*100

clf =  tree.DecisionTreeClassifier(criterion= 'gini', random_state = 44, ccp_alpha = 0.025)
clf.fit(train_x, train_y)

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

gini_0025 = clf.score(test_x, test_y)
gini_0025*100

clf =  tree.DecisionTreeClassifier(criterion= 'gini', random_state = 44, ccp_alpha = 0.03)
clf.fit(train_x, train_y)

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

gini_003 = clf.score(test_x, test_y)
gini_003*100

clf =  tree.DecisionTreeClassifier(criterion= 'gini', random_state = 44, ccp_alpha = 0.035)
clf.fit(train_x, train_y)

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

gini_0035 = clf.score(test_x, test_y)
gini_0035*100

clf =  tree.DecisionTreeClassifier(criterion= 'gini', random_state = 44, ccp_alpha = 0.2)
```

```
clf.fit(train_x, train_y)

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

gini_02 = clf.score(test_x, test_y)
gini_02*100

clf =  tree.DecisionTreeClassifier(criterion= 'gini', random_state = 44, ccp_alpha = 0.8)
clf.fit(train_x, train_y)

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

gini_08 = clf.score(test_x, test_y)
gini_08*100

clf =  tree.DecisionTreeClassifier(criterion= 'entropy', random_state = 44, ccp_alpha = 0.005)
clf.fit(train_x, train_y)

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

entropy_0005 = clf.score(test_x, test_y)
entropy_0005*100

clf =  tree.DecisionTreeClassifier(criterion= 'entropy', random_state = 44, ccp_alpha = 0.01)
clf.fit(train_x, train_y)

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

entropy_001 = clf.score(test_x, test_y)
entropy_001*100

clf =  tree.DecisionTreeClassifier(criterion= 'entropy', random_state = 44, ccp_alpha = 0.015)
clf.fit(train_x, train_y)

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

entropy_0015 = clf.score(test_x, test_y)
entropy_0015*100

clf =  tree.DecisionTreeClassifier(criterion= 'entropy', random_state = 44, ccp_alpha = 0.02)
clf.fit(train_x, train_y)

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

entropy_002 = clf.score(test_x, test_y)
```

```
entropy_002*100

clf =  tree.DecisionTreeClassifier(criterion= 'entropy', random_state = 44, ccp_alpha = 0.025)
clf.fit(train_x, train_y)

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

entropy_0025 = clf.score(test_x, test_y)
entropy_0025*100

clf =  tree.DecisionTreeClassifier(criterion= 'entropy', random_state = 44, ccp_alpha = 0.03)
clf.fit(train_x, train_y)

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

entropy_003 = clf.score(test_x, test_y)
entropy_003*100

clf =  tree.DecisionTreeClassifier(criterion= 'entropy', random_state = 44, ccp_alpha = 0.035)
clf.fit(train_x, train_y)

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

entropy_0035 = clf.score(test_x, test_y)
entropy_0035*100

clf =  tree.DecisionTreeClassifier(criterion= 'entropy', random_state = 44, ccp_alpha = 0.2)
clf.fit(train_x, train_y)

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

entropy_02 = clf.score(test_x, test_y)
entropy_02*100

clf =  tree.DecisionTreeClassifier(criterion= 'entropy', random_state = 44, ccp_alpha = 0.8)
clf.fit(train_x, train_y)

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

entropy_08 = clf.score(test_x, test_y)
entropy_08*100
```

Задание 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()

df1 = 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']
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)
print(df1['Quality'].value_counts())

scaler_std = preprocessing.StandardScaler()
x = scaler_std.fit_transform(df1[['FixedAcidity', 'VolatileAcidity', 'CitricAcid', 'ResidualSugar',
'Chlorides', 'FreeSulfurDioxide', 'TotalSulfurDioxide',
          'Density', 'PH', 'Sulphates', 'Alcohol']])
df1[['FixedAcidity', 'VolatileAcidity', 'CitricAcid', 'ResidualSugar', 'Chlorides', 'FreeSulfurDioxide',
'TotalSulfurDioxide',
          'Density', 'PH', 'Sulphates', 'Alcohol']] = x

print(df1)

train_table, test_table = train_test_split(df, test_size = 0.4, random_state = 44, stratify = df['Type'])
test_table = test_table.reset_index()
train_table = train_table.reset_index()
train_x = train_table[['FixedAcidity', 'VolatileAcidity', 'CitricAcid', 'ResidualSugar', 'Chlorides',
'FreeSulfurDioxide', 'TotalSulfurDioxide',
'Density', 'PH', 'Sulphates', 'Alcohol']]
train_y = train_table['Quality']
test_x = test_table[['FixedAcidity', 'VolatileAcidity', 'CitricAcid', 'ResidualSugar', 'Chlorides',
'FreeSulfurDioxide', 'TotalSulfurDioxide',
'Density', 'PH', 'Sulphates', 'Alcohol']]
test_y = test_table['Quality']

print(train_y)
print(test_y)
```

```
from sklearn.decomposition import PCA

pca95 = PCA(n_components=0.95)
pca95_train_x = pca95.fit_transform(train_x)
explained_variance_ratio = pca95.explained_variance_ratio_
print(explained_variance_ratio)

explained_variance = pca95.explained_variance_
print(explained_variance)

pca95 = PCA(n_components=0.95)
pca95_test_x = pca95.fit_transform(test_x)
explained_variance_ratio = pca95.explained_variance_ratio_
print(explained_variance_ratio)

explained_variance = pca95.explained_variance_
print(explained_variance)

pca_k = PCA().fit(train_x)
components_k = sum(pca_k.explained_variance_ > 1)
pca_k_final = PCA(n_components=components_k)
train_x_pca_k = pca_k_final.fit_transform(train_x)
test_x_pca_k = pca_k_final.transform(test_x)

from sklearn.tree import DecisionTreeClassifier
from sklearn.metrics import accuracy_score

def train_evaluate(train_x, test_x, train_y, test_y):
    clf = DecisionTreeClassifier(random_state=42)
    clf.fit(train_x, train_y)
    y_pred = clf.predict(test_x)
    return accuracy_score(test_y, y_pred)

accuracy_original = train_evaluate(train_x, test_x, train_y, test_y)
accuracy_pca95 = train_evaluate(pca95_train_x, pca95_test_x, train_y, test_y)
accuracy_pca_k = train_evaluate(train_x_pca_k, test_x_pca_k, train_y, test_y)

print(accuracy_original)
print(accuracy_pca95)
print(accuracy_pca_k)

clf =  tree.DecisionTreeClassifier(criterion= 'gini', random_state = 42)
clf.fit(pca95_train_x, train_y)

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

clf =  tree.DecisionTreeClassifier(criterion= 'gini', random_state = 42)
clf.fit(pca95_train_x, train_y)

tree.plot_tree(clf, filled = True, rounded = True)
```

```
plt.show()

clf =  tree.DecisionTreeClassifier(criterion= 'gini', random_state = 42)
clf.fit(pca95_test_x, test_y)

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