**CSE422\_Assignment6**

from google.colab import files

load\_data = files.upload()

import sklearn

import numpy as np

import pandas as pd

import seaborn as sns

import matplotlib.pyplot as plt

from sklearn.preprocessing import LabelEncoder

from sklearn.preprocessing import MinMaxScaler

from sklearn.model\_selection import train\_test\_split

from sklearn.neighbors import KNeighborsClassifier

from sklearn.linear\_model import LogisticRegression

from sklearn.metrics import classification\_report

from sklearn.metrics import accuracy\_score

from sklearn.tree import DecisionTreeClassifier

from sklearn.svm import SVC

from sklearn.ensemble import RandomForestClassifier

from sklearn.neural\_network import MLPClassifier

from sklearn.decomposition import PCA

#loading data

dataset = pd.read\_csv('mushroom edibility classification dataset.csv')

# missing values

dataset.drop(dataset.columns[dataset.columns.str.contains('unnamed',case = False)],axis = 1, inplace = True)

dataset.isnull().sum()

dataset = dataset.dropna(axis = 0, subset = ['cap-shape', 'cap-color'])

print('unique values in class ' + str(dataset['class'].unique()))

print('unique values in bruises' + str(dataset['bruises'].unique()))

#encoding categorical features

enc = LabelEncoder()

dataset['class'] = enc.fit\_transform(dataset['class'])

dataset['bruises'] = enc.fit\_transform(dataset['bruises'])

dataset[['class', 'bruises']].head()

scaler = MinMaxScaler()

scaler.fit(dataset)

dataset\_train\_scaled = scaler.transform(dataset)

print('per-feature minimum before scaling:\n{}'.format(dataset.min(axis = 0)))

print('per-feature maximum before scaling:\n{}'.format(dataset.max(axis = 0)))

print('per-feature minimum after scaling:\n{}'.format(dataset\_train\_scaled.min(axis = 0)))

print('per-feature maximum after scaling:\n{}'.format(dataset\_train\_scaled.max(axis = 0)))

#splitting the dataset into features and labels

features = dataset[['cap-shape', 'cap-surface', 'cap-color', 'bruises', 'odor', 'stalk-shape', 'stalk-root', 'stalk-surface-above-ring', 'stalk-surface-below-ring', 'stalk-color-above-ring', 'stalk-color-below-ring', 'veil-type', 'veil-color', 'ring-number', 'ring-type', 'spore-print-color', 'population', 'habitat']]

label = dataset[['class']]

stratified = pd.DataFrame(label)

#8:2 train-test split

xTrain, xTest, yTrain, yTest = train\_test\_split(features, label, test\_size = 0.20, stratify = stratified, random\_state = 0)

#using the logistic regression

perform\_logisticRegression = LogisticRegression()

perform\_logisticRegression.fit(xTrain, yTrain)

predictions = perform\_logisticRegression.predict(xTest)

accuracy\_of\_LogisticRegression = accuracy\_score(yTest, predictions)

print(accuracy\_of\_LogisticRegression)

#using the decision tree

perform\_decisionTree = DecisionTreeClassifier(criterion='entropy',random\_state=1)

perform\_decisionTree.fit(xTrain,yTrain)

yPred = perform\_decisionTree.predict(xTest)

accuracy\_of\_DecisionTreeClassifier = accuracy\_score(yPred, yTest)

print(accuracy\_of\_DecisionTreeClassifier)

#using Support Vector Machine

perform\_supportVectorClassification = SVC(kernel="linear")

perform\_supportVectorClassification.fit(xTrain, yTrain)

prediction\_of\_SupportVectorClassification = perform\_supportVectorClassification.predict(xTest)

accuracy\_of\_SupportVectorMachine = accuracy\_score(prediction\_of\_SupportVectorClassification, yTest)

print(accuracy\_of\_SupportVectorMachine)

#using Random Forest Classification

perform\_randomForestClassifier = RandomForestClassifier(n\_estimators = 50)

perform\_randomForestClassifier.fit(xTrain, yTrain)

prediction\_of\_RandomForestClassifier = perform\_randomForestClassifier.predict(xTest)

accuracy\_of\_RandomForestClassifier = accuracy\_score(prediction\_of\_RandomForestClassifier, yTest)

print(accuracy\_of\_RandomForestClassifier)

#using Neural Network

perform\_neuralNetwork = MLPClassifier(hidden\_layer\_sizes=(7), activation="relu", max\_iter=1000)

perform\_neuralNetwork.fit(xTrain, yTrain)

prediction\_of\_NeuralNetwork =perform\_neuralNetwork.predict(xTest)

accuracy\_of\_NeuralNetwork = accuracy\_score(prediction\_of\_NeuralNetwork, yTest)

print(accuracy\_of\_NeuralNetwork)

plt.bar(['Logistic\nRegression', 'Decision\nTree\nClassification', 'Support\nVector\nMachine', 'Random\nForest\nClassification', 'Neural\nNetwork'],[accuracy\_of\_LogisticRegression, accuracy\_of\_DecisionTreeClassifier, accuracy\_of\_SupportVectorMachine, accuracy\_of\_RandomForestClassifier, accuracy\_of\_NeuralNetwork])

plt.title('Accuracy Comparison among Algorithms ')

plt.show()

#using pca and reducing the number of feature vectors into half

columnCount = int(len(features.columns.values.tolist())/2)

pca = PCA(n\_components = columnCount)

principal\_components = pca.fit\_transform(features)

principalColummns = ["Principal Component" + str(i+1) for i in range(columnCount)]

principal\_df = pd.DataFrame(data=principal\_components, columns = principalColummns)

main\_df=pd.concat([principal\_df, dataset[["class"]]], axis=1)

main\_df = main\_df.dropna(how = 'any', axis = 0)

pcaFeature = main\_df.drop('class', axis = 1)

pcaLabel = main\_df['class']

pcaXTrain, pcaXTest, pcaYTrain, pcaYTest = train\_test\_split(pcaFeature , pcaLabel, test\_size=0.2, stratify = pcaLabel, random\_state=0)

pcaLabelName = ['class']

pcaFeatureName = principalColummns #list(features.columns.values.tolist())

scaler = MinMaxScaler()

pcaXTrain = pd.DataFrame(scaler.fit\_transform(pcaXTrain), columns = pcaFeatureName)

pcaXTest = pd.DataFrame(scaler.fit\_transform(pcaXTest), columns = pcaFeatureName)

#using Logistic Regression again

perform\_logisitceRegressionModel = LogisticRegression()

perform\_logisitceRegressionModel.fit(pcaXTrain, pcaYTrain)

prediction\_of\_LogisticRegressionModel = perform\_logisitceRegressionModel.predict(pcaXTest)

pcaAccuracy\_of\_LogisticRegression = accuracy\_score(pcaYTest, prediction\_of\_LogisticRegressionModel)

print(pcaAccuracy\_of\_LogisticRegression)

#using Decision Tree again

perform\_decisionTreeClassification = DecisionTreeClassifier(criterion='entropy',random\_state=1)

perform\_decisionTreeClassification.fit(pcaXTrain,pcaYTrain)

prediction\_of\_DecisionTreeClassification = perform\_decisionTreeClassification.predict(pcaXTest)

pcaAccuracy\_of\_DecisionTreeClassifier = accuracy\_score(prediction\_of\_DecisionTreeClassification, pcaYTest)

print(pcaAccuracy\_of\_DecisionTreeClassifier)

#using Support Vector Machine again

perform\_supportVectorClassification = SVC(kernel="linear")

perform\_supportVectorClassification.fit(pcaXTrain, pcaYTrain)

prediction\_of\_SupportVectorClassification = perform\_supportVectorClassification.predict(pcaXTest)

pcaAccuracy\_of\_SupportVectorMachine = accuracy\_score(prediction\_of\_SupportVectorClassification, pcaYTest)

print(pcaAccuracy\_of\_SupportVectorMachine)

#using Random Forest Classification again

perform\_randomForestClassifier = RandomForestClassifier(n\_estimators = 50)

perform\_randomForestClassifier.fit(pcaXTrain, pcaYTrain)

prediction\_of\_RandomForestClassifier = perform\_randomForestClassifier.predict(pcaXTest)

pcaAccuracy\_of\_RandomForestClassifier = accuracy\_score(prediction\_of\_RandomForestClassifier, pcaYTest)

print(pcaAccuracy\_of\_RandomForestClassifier)

#using Neural Network again

perform\_neuralNetwork = MLPClassifier(hidden\_layer\_sizes=(7), activation="relu", max\_iter=100000)

perform\_neuralNetwork.fit(pcaXTrain, pcaYTrain)

prediction\_of\_NeuralNetwork = perform\_neuralNetwork.predict(pcaXTest)

pcaAccuracy\_of\_NeuralNetwork = accuracy\_score(prediction\_of\_NeuralNetwork, pcaYTest)

print(pcaAccuracy\_of\_NeuralNetwork)

plt.bar(['Logistic\nRegression', 'Decision\nTree\nClassification', 'Support\nVector\nMachine', 'Random\nForest\nClassification', 'Neural\nNetwork'],[pcaAccuracy\_of\_LogisticRegression, pcaAccuracy\_of\_DecisionTreeClassifier, pcaAccuracy\_of\_SupportVectorMachine, pcaAccuracy\_of\_RandomForestClassifier, pcaAccuracy\_of\_NeuralNetwork])

plt.title('Accuracy Comparison among Algorithms ')

plt.show()

plt.bar(['Before PCA', 'After PCA'],[accuracy\_of\_LogisticRegression, pcaAccuracy\_of\_LogisticRegression])

plt.title('Accuracy Comparison of Logistic Regression ')

plt.show()

plt.bar(['Before PCA', 'After PCA'],[accuracy\_of\_SupportVectorMachine, pcaAccuracy\_of\_SupportVectorMachine])

plt.title('Accuracy Comparison of Support Vector Machine')

plt.show()

plt.bar(['Before PCA', 'After PCA'],[accuracy\_of\_RandomForestClassifier, pcaAccuracy\_of\_RandomForestClassifier])

plt.title('Accuracy Comparison of Random Forest Classifier ')

plt.show()

plt.bar(['Before PCA', 'After PCA'],[accuracy\_of\_NeuralNetwork, pcaAccuracy\_of\_NeuralNetwork])

plt.title('Accuracy Comparison of Neural Network ')

plt.show()