Faculty of Engineering & Technology
Electrical & Computer Engineering Department

Classifying emails

Prepared by: Oday Ziq 1201168

Date: 15/07/2023

Table Of Contents

Formalization

The project focuses on classifying emails as spam or not-spam using the provided spambase.csv dataset. The dataset consists of 4601 examples, with each example represented by 58 numerical attributes. The last number in each row denotes whether the email was considered spam (1) or not (0). The first 57 numbers represent the attributes of the emails. Detailed information about the attributes can be found at https://archive.ics.uci.edu/dataset/94/spambase.

Output & Design Description

Procedure

Imports:

```
import numpy as np
import pandas as pd
import seaborn as sns
import sys
from sklearn.model_selection import train_test_split
from sklearn.neural_network import MLPClassifier
from sklearn.preprocessing import StandardScaler
from sklearn.metrics import accuracy_score, precision_score, recall_score,
f1_score
from sklearn.metrics import confusion_matrix
from scipy.spatial import distance
import matplotlib.pyplot as plt
```

The script imports Python libraries for scientific computing (numpy, pandas), data visualization (seaborn, matplotlib), machine learning (sklearn modules for dataset splitting, neural network creation, data preprocessing, and model evaluation), and computation of distances between points (scipy.spatial). The sys module is also imported for interpreter-specific operations.

Class NN:

```
class NN:
    def __init__(self, trainingFeatures, trainingLabels) -> None:
        self.trainingFeatures = trainingFeatures
        self.trainingLabels = trainingLabels

    def predict(self, features, k):
        try:
            predictions = []
            for x in features:
                distances = np.array([distance.euclidean(x, xi) for xi in
self.trainingFeatures])
                k_nearest_indices = distances.argsort()[:k]
                k_nearest_labels = [self.trainingLabels[i] for i in
k_nearest_indices]
                predictions.append(max(set(k_nearest_labels), key =
k_nearest_labels.count))
        except NotImplementedError as error:
            print(error)
        return predictions
```

The provided class NN constructs a k-nearest neighbor's model. On initialization, it stores training data. The predict method calculates Euclidean distances between a new point and existing points, identifies the 'k' closest points, and assigns the most frequent label among these as the new point's label. Errors in this process are caught and printed.

Load data:

```
def load_data(filename):
    try:
        df = pd.read_csv(filename, header=None)
        labels = df.iloc[:, -1]
        features = df.iloc[:, :-1]
    except NotImplementedError as error:
        print(error)
    return features.values, labels.values
```

This function load_data(filename) reads a CSV file into a pandas DataFrame, separates the last column as labels and the rest as features, and returns these as numpy arrays. If an error occurs during this process, it is caught and printed to the console.

Preprocess:

```
def preprocess(features):
    try:

        scaler = StandardScaler()
        features = scaler.fit_transform(features)
    except NotImplementedError as error:
        print(error)
    return features
```

The function preprocess (features) standardizes the input features using StandardScaler from scikit-learn, transforming them to have a mean of 0 and a standard deviation of 1. Any errors during the process are caught and printed out.

Train MLP model:

```
def train_mlp_model(features, labels):
    try:
        model = MLPClassifier(hidden_layer_sizes=(10,5),
activation='logistic', max_iter=5000)
        model.fit(features, labels)
```

```
    except NotImplementedError as error:
        print(error)
    return model
```

The function train_mlp_model(features, labels) trains a Multilayer Perceptron (MLP) model with two hidden layers (10 neurons and 5 neurons respectively), using logistic activation. It catches and prints any occurring errors during this process.

Evaluate:

```
def evaluate(labels, predictions):
    try:

        accuracy = accuracy_score(labels, predictions)
        precision = precision_score(labels, predictions)
        recall = recall_score(labels, predictions)
        f1 = f1_score(labels, predictions)
    except NotImplementedError as error:
        print(error)
    return accuracy, precision, recall, f1
```

The function **evaluate(labels, predictions)** computes the accuracy, precision, recall, and F1-score of the predicted labels against the true labels. It prints any error that occurs during this evaluation.

Main:

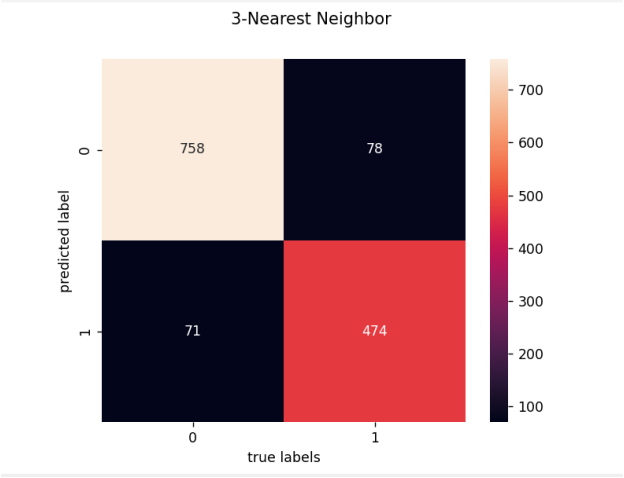
```
try:
    csv_path = 'D:\\d\\spam\\spambase.csv'

    features, labels = load_data(csv_path)
    features = preprocess(features)
    X_train, X_test, y_train, y_test = train_test_split(
        features, labels, test_size=TEST_SIZE)

    model_nn = NN(X_train, y_train)
    predictions = model_nn.predict(X_test, K)
    accuracy, precision, recall, f1 = evaluate(y_test, predictions)

    print("**** 3-Nearest Neighbor Results ****")
    print("Accuracy: ", accuracy)
    print("Precision: ", precision)
    print("Recall: ", recall)
    print("F1: ", f1)
    mat = confusion_matrix(y_test, predictions)
    sns.heatmap(mat, square=True, annot=True, fmt="d")
    plt.xlabel("true labels")
```

```
    plt.ylabel("predicted label")
    plt.suptitle("3-Nearest Neighbor")
    plt.show()

    model = train_mlp_model(X_train, y_train)
    predictions = model.predict(X_test)
    accuracy, precision, recall, f1 = evaluate(y_test, predictions)


    print("**** MLP Results ****")
    print("Accuracy: ", accuracy)
    print("Precision: ", precision)
    print("Recall: ", recall)
    print("F1: ", f1)
    mat = confusion_matrix(y_test, predictions)
    sns.heatmap(mat, square=True, annot=True, fmt="d")
    plt.xlabel("true labels")
    plt.ylabel("predicted label")
    plt.suptitle(" MLP ")
    plt.show()
except FileNotFoundError as error:
    print(error)
```

This script reads data from a CSV file, preprocesses features, splits the data into training and testing sets, and then trains a k-nearest neighbor's model and a Multilayer Perceptron (MLP) model. It makes predictions with each model and evaluates those using accuracy, precision, recall, and F1-score metrics, displaying results and confusion matrices. It handles File Not Found errors and prints the associated error message.

Output:

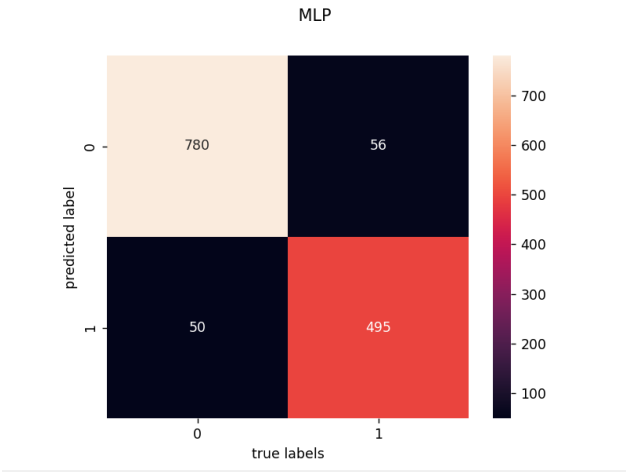
For the 3 nearest neighbor this is the visualize for the data:



Here is the output of the 3 nearest neighbor:

```
**** 3-Nearest Neighbor Results ****
Accuracy:  0.8921071687183201
Precision:  0.8586956521739131
Recall:  0.8697247706422019
F1:  0.8641750227894258
```

For the MLP this is the visualize for the data:



Here is the output of the MLP:

```
**** MLP Results ****
Accuracy:  0.9232440260680667
Precision:  0.8983666061705989
Recall:  0.908256880733945
F1:  0.9032846715328466
```