Electrical and Computer Engineering Department

ENCS3340 Artificial Intelligence, First Semester, 2022-2023

Project 2

Name: Sondos Hisham   ID:1200166
Name: Shahd Qatre   ID:1200431

Instructor: Dr.Yazan Abu Farha
section: 4

Data :14/7/2023

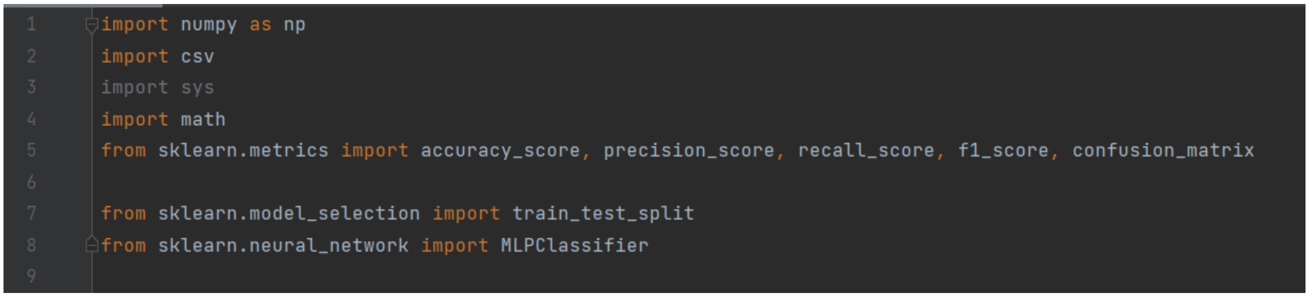
Contents

project task:

On the given dataset, we must test a k-NN and a multi-layer perceptron classifier. we must train each classifier on the training set, and then on the test set, we must report its accuracy, precision, recall, and F1-score.

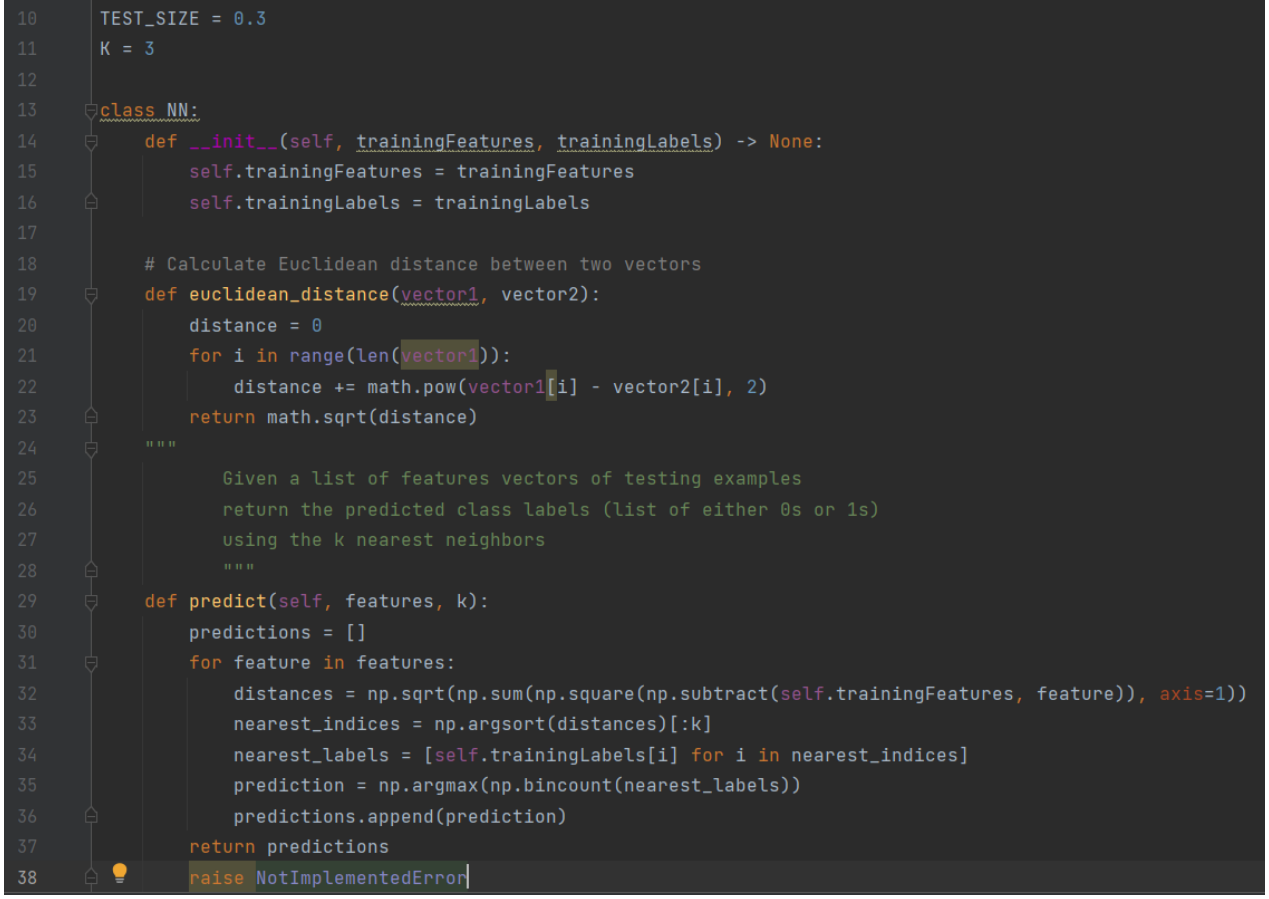
Code:

Importing the important libraries required for the project



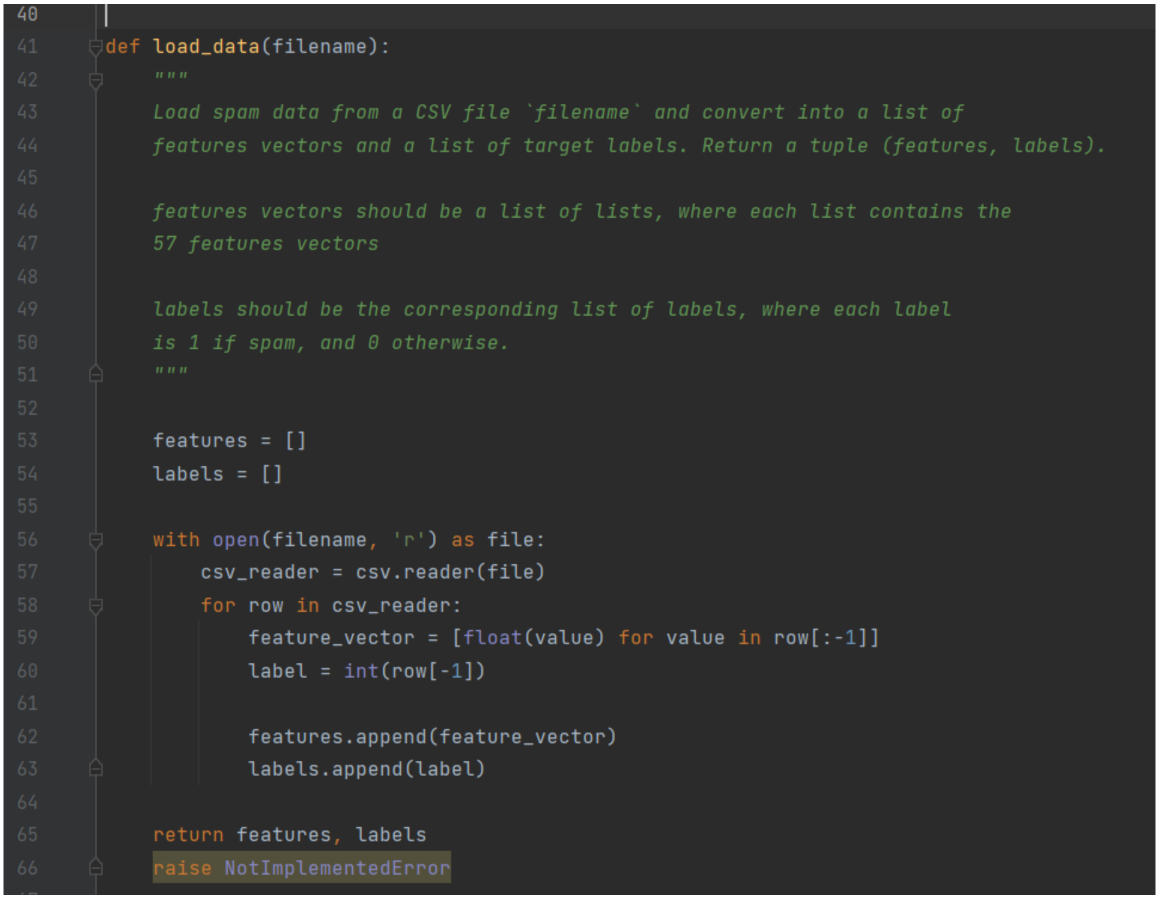
```
import numpy as np
import csv
import sys
import math
from sklearn.metrics import accuracy_score, precision_score, recall_score, f1_score, confusion_matrix

from sklearn.model_selection import train_test_split
from sklearn.neural_network import MLPClassifier

```

Implementing the knn classifier:

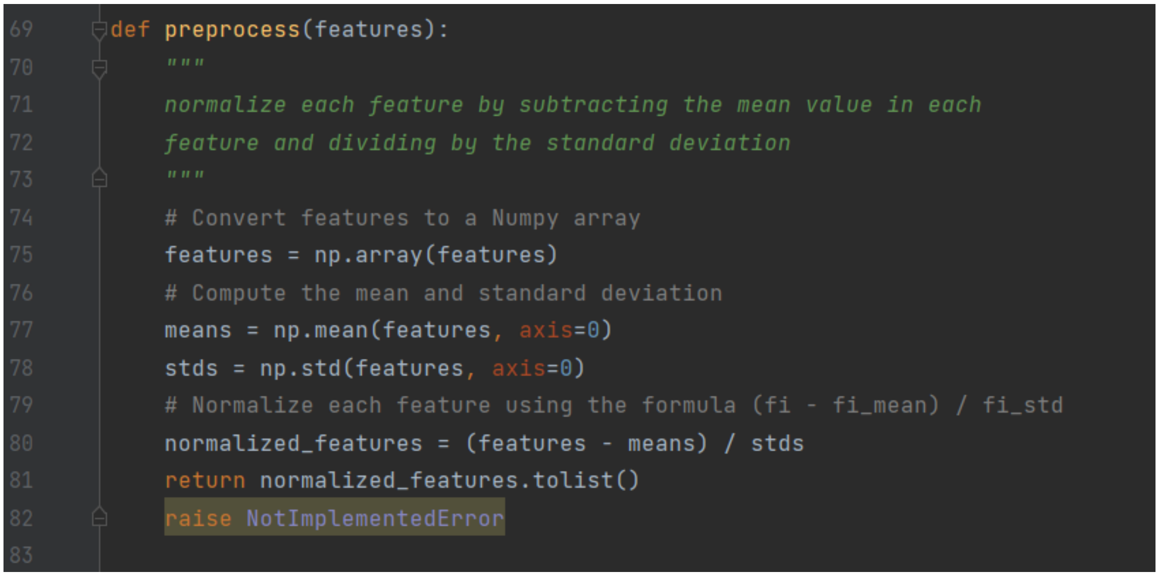


```
TEST_SIZE = 0.3
K = 3

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

    # Calculate Euclidean distance between two vectors
    def euclidean_distance(vector1, vector2):
        distance = 0
        for i in range(len(vector1)):
            distance += math.pow(vector1[i] - vector2[i], 2)
        return math.sqrt(distance)
    """
           Given a list of features vectors of testing examples
           return the predicted class labels (list of either 0s or 1s)
           using the k nearest neighbors
           """
    def predict(self, features, k):
        predictions = []
        for feature in features:
            distances = np.sqrt(np.sum(np.square(np.subtract(self.trainingFeatures, feature)), axis=1))
            nearest_indices = np.argsort(distances)[:k]
            nearest_labels = [self.trainingLabels[i] for i in nearest_indices]
            prediction = np.argmax(np.bincount(nearest_labels))
            predictions.append(prediction)
        return predictions
        raise NotImplementedError
```

Load the data from the csv file:

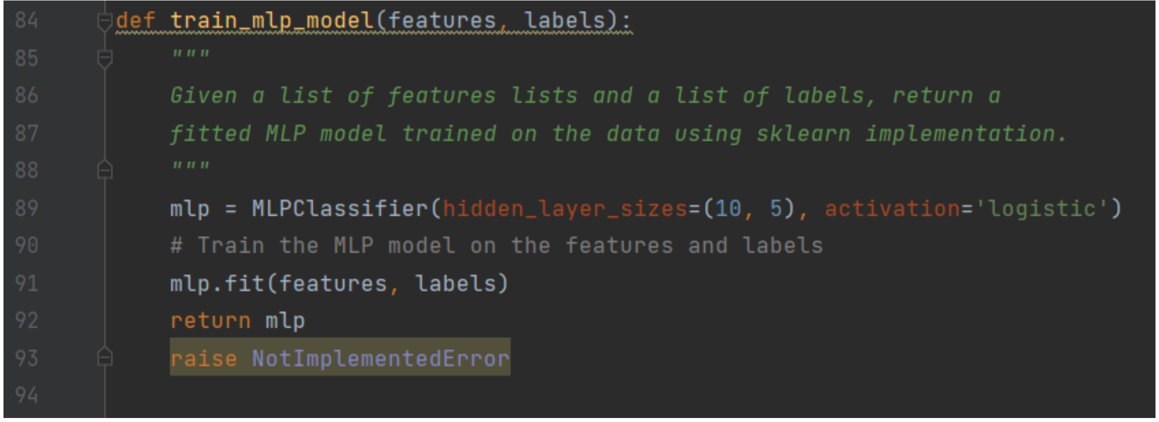


```python

def load_data(filename):
    """
    Load spam data from a CSV file `filename` and convert into a list of
    features vectors and a list of target labels. Return a tuple (features, labels).

    features vectors should be a list of lists, where each list contains the
    57 features vectors

    labels should be the corresponding list of labels, where each label
    is 1 if spam, and 0 otherwise.
    """

    features = []
    labels = []

    with open(filename, 'r') as file:
        csv_reader = csv.reader(file)
        for row in csv_reader:
            feature_vector = [float(value) for value in row[:-1]]
            label = int(row[-1])

            features.append(feature_vector)
            labels.append(label)

    return features, labels
    raise NotImplementedError
```



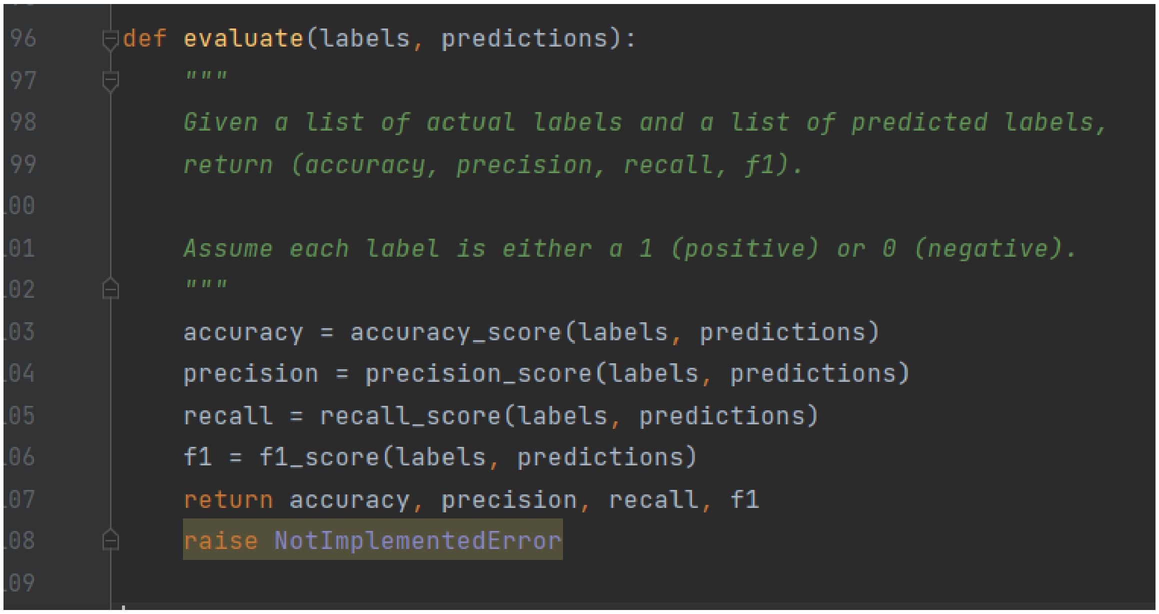
```python
def preprocess(features):
    """
    normalize each feature by subtracting the mean value in each
    feature and dividing by the standard deviation
    """
    # Convert features to a Numpy array
    features = np.array(features)
    # Compute the mean and standard deviation
    means = np.mean(features, axis=0)
    stds = np.std(features, axis=0)
    # Normalize each feature using the formula (fi - fi_mean) / fi_std
    normalized_features = (features - means) / stds
    return normalized_features.tolist()
    raise NotImplementedError

```

Train the mlp model:

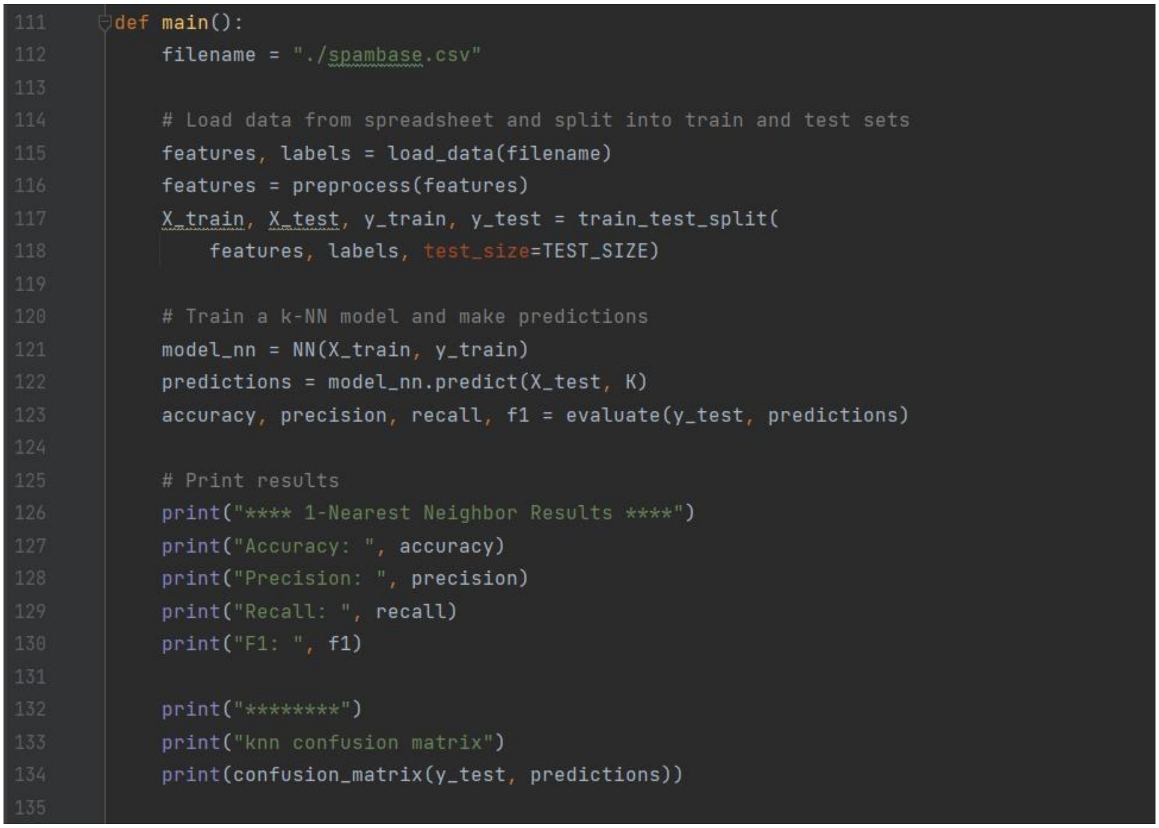


```
def train_mlp_model(features, labels):
    """
    Given a list of features lists and a list of labels, return a
    fitted MLP model trained on the data using sklearn implementation.
    """
    mlp = MLPClassifier(hidden_layer_sizes=(10, 5), activation='logistic')
    # Train the MLP model on the features and labels
    mlp.fit(features, labels)
    return mlp
    raise NotImplementedError

```

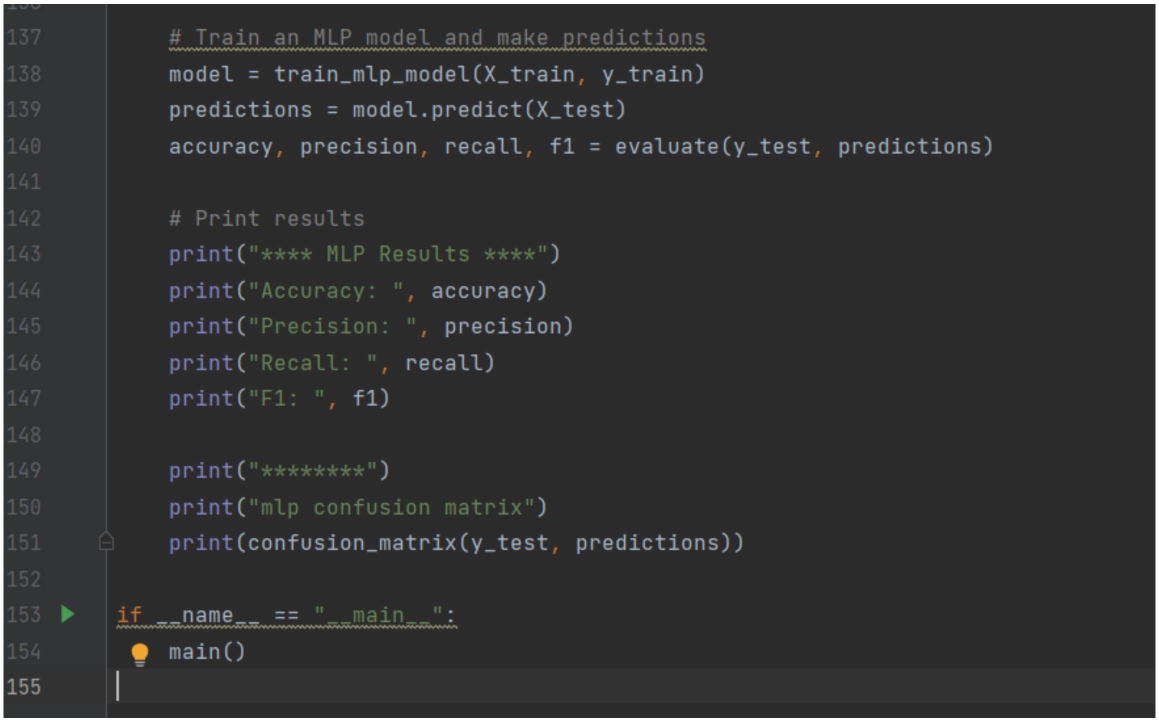
Evaluation function:



```
def evaluate(labels, predictions):
    """
    Given a list of actual labels and a list of predicted labels,
    return (accuracy, precision, recall, f1).

    Assume each label is either a 1 (positive) or 0 (negative).
    """
    accuracy = accuracy_score(labels, predictions)
    precision = precision_score(labels, predictions)
    recall = recall_score(labels, predictions)
    f1 = f1_score(labels, predictions)
    return accuracy, precision, recall, f1
    raise NotImplementedError

```

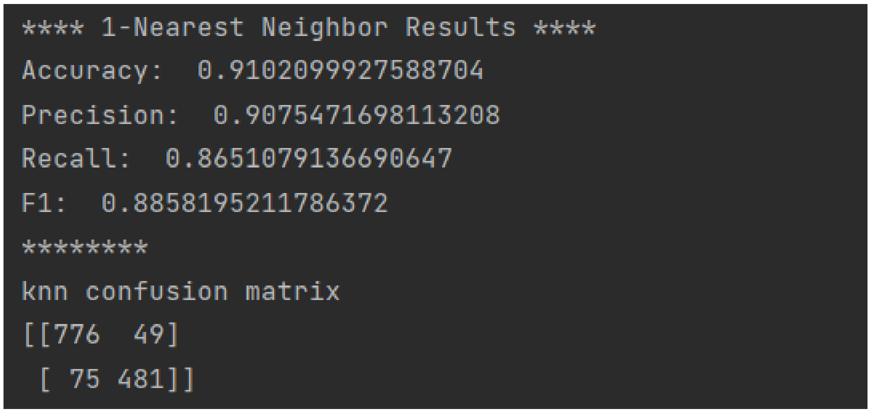


```python
def main():
    filename = "./spambase.csv"

    # Load data from spreadsheet and split into train and test sets
    features, labels = load_data(filename)
    features = preprocess(features)
    X_train, X_test, y_train, y_test = train_test_split(
        features, labels, test_size=TEST_SIZE)

    # Train a k-NN model and make predictions
    model_nn = NN(X_train, y_train)
    predictions = model_nn.predict(X_test, K)
    accuracy, precision, recall, f1 = evaluate(y_test, predictions)

    # Print results
    print("**** 1-Nearest Neighbor Results ****")
    print("Accuracy: ", accuracy)
    print("Precision: ", precision)
    print("Recall: ", recall)
    print("F1: ", f1)

    print("********")
    print("knn confusion matrix")
    print(confusion_matrix(y_test, predictions))

```

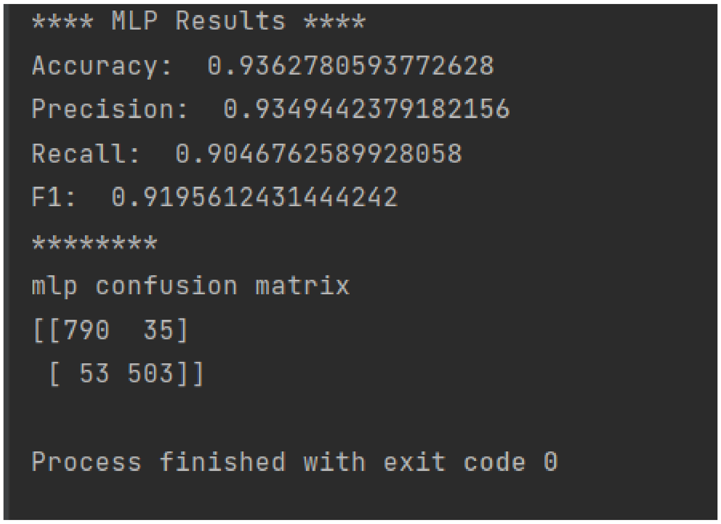


```python
    # Train an MLP model and make predictions
    model = train_mlp_model(X_train, y_train)
    predictions = model.predict(X_test)
    accuracy, precision, recall, f1 = evaluate(y_test, predictions)

    # Print results
    print("**** MLP Results ****")
    print("Accuracy: ", accuracy)
    print("Precision: ", precision)
    print("Recall: ", recall)
    print("F1: ", f1)

    print("********")
    print("mlp confusion matrix")
    print(confusion_matrix(y_test, predictions))

if __name__ == "__main__":
    main()

```

Output:



```
**** 1-Nearest Neighbor Results ****
Accuracy:  0.9102099927588704
Precision:  0.9075471698113208
Recall:  0.8651079136690647
F1:  0.8858195211786372
********
knn confusion matrix
[[776  49]
 [ 75 481]]
```



```
**** MLP Results ****
Accuracy:  0.9362780593772628
Precision:  0.9349442379182156
Recall:  0.9046762589928058
F1:  0.9195612431444242
********
mlp confusion matrix
[[790  35]
 [ 53 503]]

Process finished with exit code 0
```

Improvements:

To improve the performance of the knn model, we can increase the value of k, but not make it too large. We can also perform feature selection techniques to identify and select the most relevant features for the classification task. This can help reduce noise and improve the model's ability to generalize.