YILDIZ TECHNICAL UNIVERSITY

FACULTY OF ELECTRICAL AND ELECTRONICS

COMPUTER ENGINEERING DEPARTMENT

BLM3120 – Information Retrieval and Search Engines
Gr:1
Prof. Dr. Mehmet Sıddık Aktaş

Project Report

Students	Number
Bilal MÜFTÜOĞLU	20011007
Selahattin Yasin ÇAYCI	20011099

How It Works?

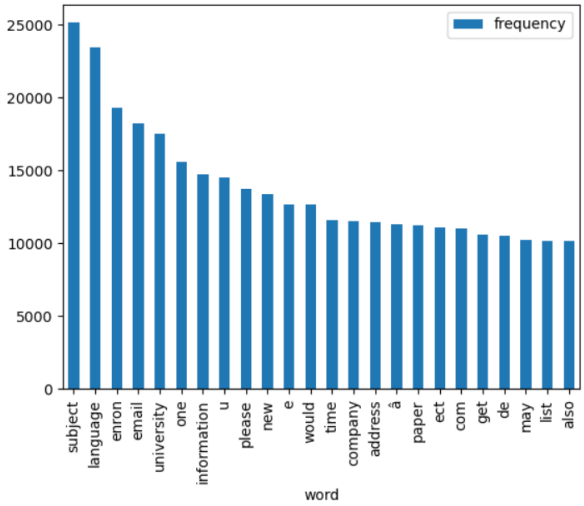
To achieve more robust results within the scope of the project, three different datasets were utilized. Firstly, null values in the datasets were removed. Then, column values in these datasets were standardized, and the three datasets were merged into a single dataset, resulting in a total of 18,650 elements.

```
enronSpam = pd.read_csv("./enronSpamSubset.csv")
lingSpam = pd.read_csv("./lingSpam.csv")
completeSpam = pd.read_csv("./completeSpamAssassin.csv")

completeSpam.dropna(inplace=True)
enronSpam.drop(["Unnamed: 0.1", "Unnamed: 0"], axis=1, inplace=True)
lingSpam.drop(["Unnamed: 0"], axis=1, inplace=True)
completeSpam.drop(["Unnamed: 0"], axis=1, inplace=True)
dataset = pd.concat([enronSpam,lingSpam,completeSpam], axis=0, ignore_index=True)
```

```
def preprocess(dataset = None, flag = 1):
    if flag == 0: # dataset includes all inputs
        df = dataset.copy()
        body = df['Body']

        # Model can be confused due to the links. We will extract and remove links with regex(regular expressions). There is also
        alternatives like BeatifulSoup(for HTML labels), linkify etc.
        for i in range(len(body)):
                link_pattern = re.compile(r'http[s]?://(?:[a-zA-Z]|[0-9]|[$-_@.&+]|[!*\\(\\),]|(?:%[0-9a-fA-F][0-9a-fA-F]))+', re.
                IGNORECASE)
                df.loc[i,"Body"] = re.sub(link_pattern, '', df.loc[i,"Body"])
                # it fails if link is like that: "http : / / www . tiscali . co . uk / travel / flights /"

        # convert upper cases to lower cases:
        df['Body'] = df['Body'].apply(lambda line: " ".join(text.lower() for text in line.split()))

        # removing punctuations:
        df['Body'] = df['Body'].replace("[^\w\s]","", regex=True)

        # removing numbers:
        df['Body'] = df['Body'].replace("\d","",regex=True)

        # removing stopwords:
        sw = stopwords.words("english")
        df['Body'] = df['Body'].apply(lambda line: " ".join(text for text in line.split() if text not in sw))

        # to find root of a word, we can use stemming and lemmatization. Stemming is faster but has lower accuracy so we'll use
        lemmatization
        df['Body'] = df['Body'].apply(lambda line: " ".join([Word(text).lemmatize() for text in line.split()]))

        return df
```

The data needs preprocessing. Since model can be confused due to the links, sections containing links in emails were removed. Then, all characters were converted to lowercase, punctuation marks and numbers were removed. After removing stopwords, the roots of words were found with lemmatization method, non-root parts were removed, and the preprocessing step was completed.

After the preprocessing step, the list of the most common words in the dataset was as follows:



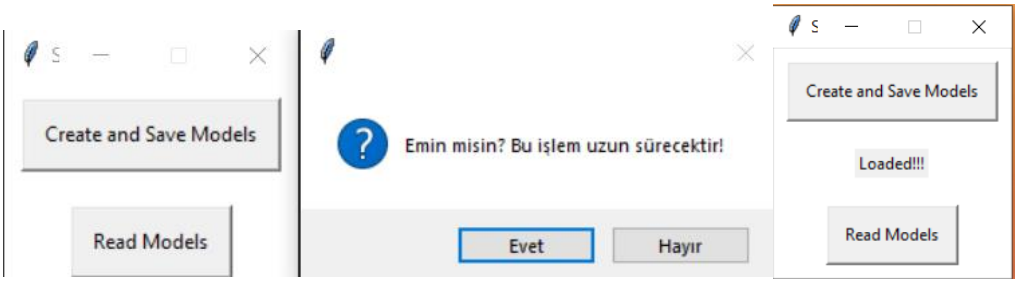
After applying tf-idf vectorization to the dataset, the data was split into 75% training and 25% testing sets, and the necessary steps for model creation were completed. Models were built with Logistic Regression, Random Forest, Gradient Boosting Machines, K-Nearest Neighbors and Multilayer Perceptron (MLP) algorithms. Additionally, metrics such as confusion matrix, accuracy, precision, recall, and f1 score were calculated for each model. Since there is no need to retrain the models every time the application opens or closes, the created models are saved into the local storage.

```python
def eval_metrics(model, X_test, Y_test):
    predict = model.predict(X_test)
    confusion = confusion_matrix(Y_test,predict)
    accuracy = accuracy_score(Y_test,predict)
    precision = precision_score(Y_test, predict)
    recall = recall_score(Y_test, predict)
    f1 = f1_score(Y_test,predict)
    print("accuracy_score:",str(accuracy),"\n",
          "precision_score:",str(precision),"\n",
          "recall_score:",str(recall),"\n",
          "f1_score:",str(f1),"\n",
          "confusion_matrix:","\n",str(confusion),"\n",
          model.get_params())
    return confusion, accuracy, precision, recall, f1
```

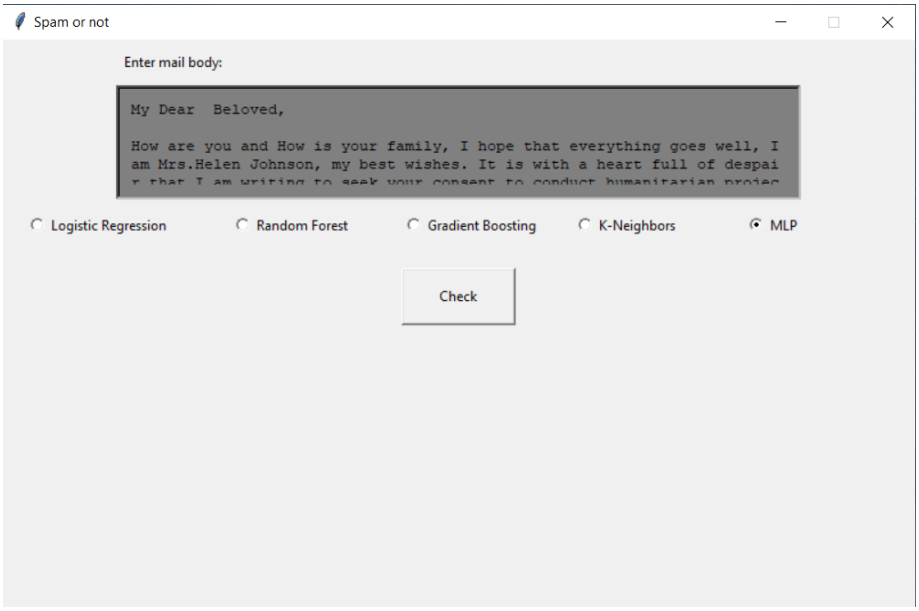
```python
df_preprocessed = preprocess(dataset,0)

X, Y = df_preprocessed['Body'], df_preprocessed['Label']
tf_idf_vectorizer = TfidfVectorizer()
tf_idf_vectorizer.fit(X)
X= tf_idf_vectorizer.transform(X)
X_train,X_test,Y_train,Y_test = train_test_split(X, Y, random_state=42)

model_logistic = linear_model.LogisticRegression().fit(X_train, Y_train)
model_rf = ensemble.RandomForestClassifier().fit(X_train, Y_train)
model_gbm = ensemble.GradientBoostingClassifier().fit(X_train, Y_train)
model_knn = KNeighborsClassifier(n_neighbors=38).fit(X_train, Y_train) # best n_neighbors value is 38
"""...
model_mlp = MLPClassifier().fit(X_train, Y_train)

confusion, accuracy, precision, recall, f1 = eval_metrics(model_logistic, X_test=X_test, Y_test=Y_test)
metrics['Logistic Regression'] = [confusion, accuracy, precision, recall, f1]

confusion, accuracy, precision, recall, f1 =  eval_metrics(model_rf, X_test=X_test, Y_test=Y_test)
metrics['Random Forest'] = [confusion, accuracy, precision, recall, f1]

confusion, accuracy, precision, recall, f1 = eval_metrics(model_gbm, X_test=X_test, Y_test=Y_test)
metrics['Gradient Boosting'] = [confusion, accuracy, precision, recall, f1]

confusion, accuracy, precision, recall, f1 = eval_metrics(model_knn, X_test=X_test, Y_test=Y_test)
metrics['K-Neighbors'] = [confusion, accuracy, precision, recall, f1]

confusion, accuracy, precision, recall, f1 =  eval_metrics(model_mlp, X_test=X_test, Y_test=Y_test)
metrics['MLP'] = [confusion, accuracy, precision, recall, f1]
```

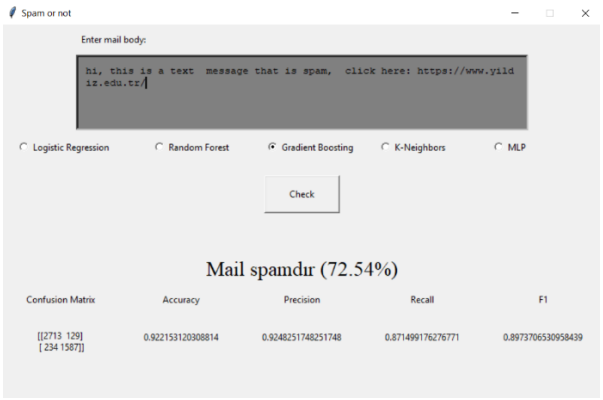
Application

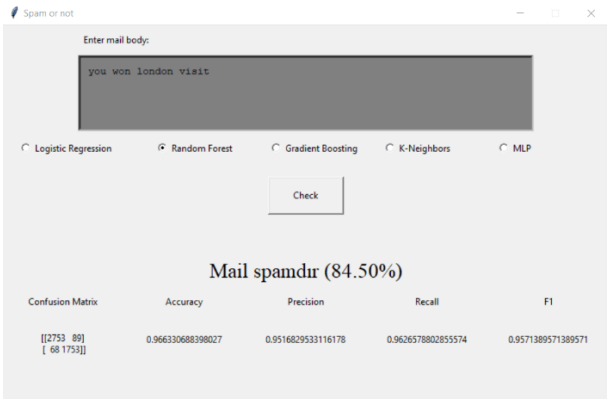
Initially, two buttons appear on the screen: one for creating models and the other for reading models. With the creation button, five different models are built from scratch. With the reading button, locally saved models are loaded.



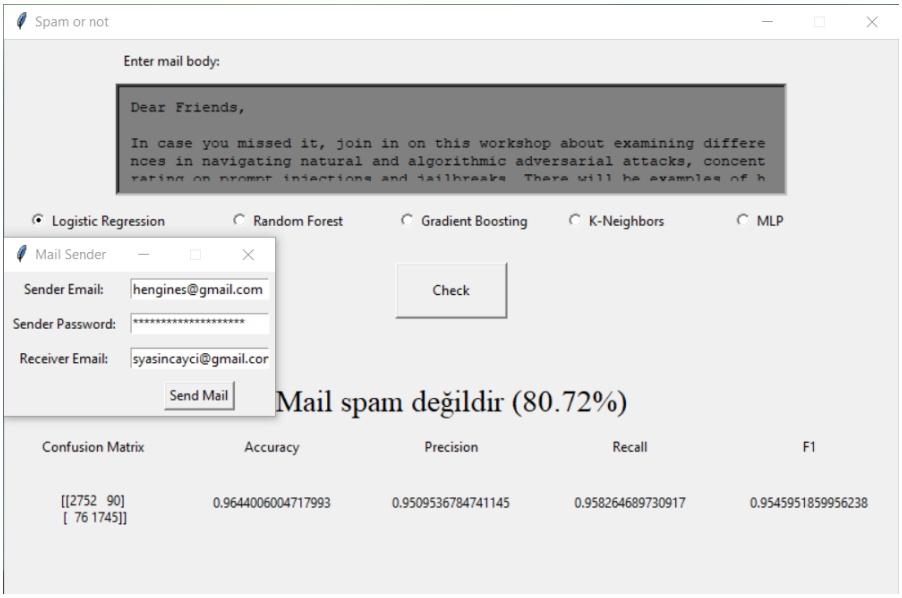
Then, on the opened page, the user is prompted to enter a text. Additionally, one of the five different algorithms is selected. Accordingly, metrics for that model and whether the entered text is spam or not are displayed on the screen with confidence.

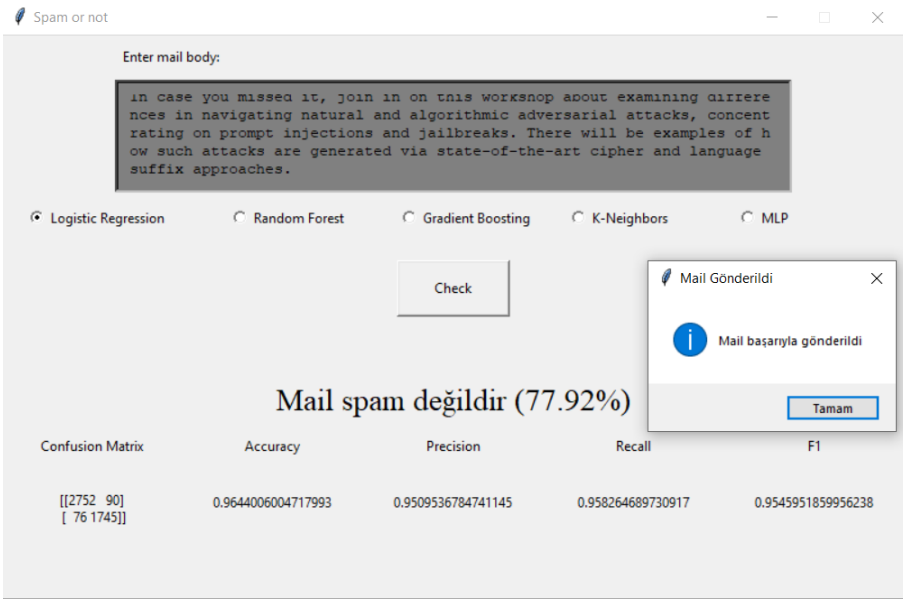






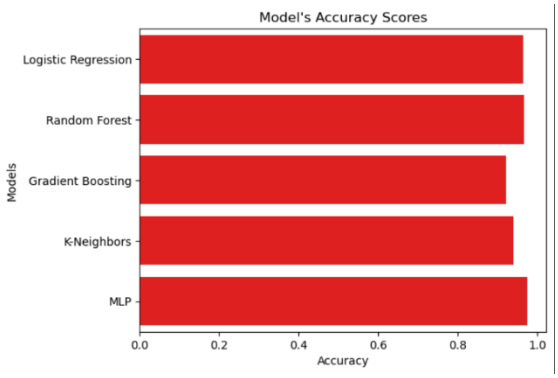
If the text is not spam, a pop-up appears, and information is requested on which email address the email should be sent from and to. After entering the information, the text is sent from one email address to another.

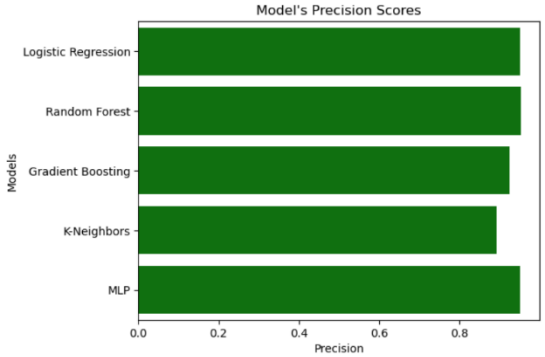


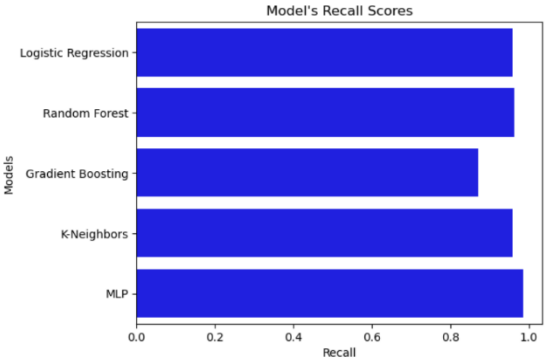


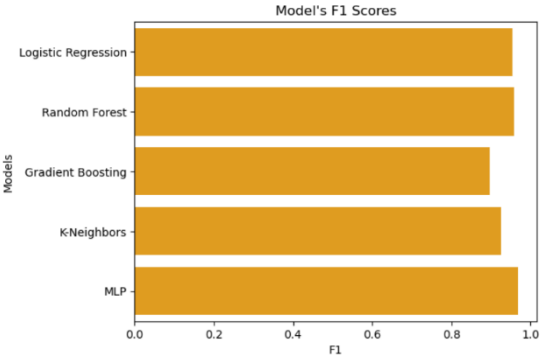
Performance Analysis

Model Name Metric	Logistic Regression	Random Forest	GBM	KNN (tuned)	MLP
Accuracy	0.9644	0.9663	0.9221	0.9388	0.9744
Precision	0.9509	0.9516	0.9248	0.8926	0.9507
Recall	0.9582	0.9626	0.8714	0.9588	0.9857
F1	0.9545	0.9571	0.8973	0.9245	0.9679









KNN initially had the following metric values: Accuracy: 0.5127 Precision: 0.4448 Recall: 0.9989 F1 Score: 0.6155.

After the model tuning process, the number of neighbors was set to 38, and the new metrics are as per above.

The logistic regression model demonstrates a highly successful performance with high accuracy, precision, recall, and F1 score. The model has successfully distinguished between spam and non-spam classes.

The Random Forest model also performs well, similar to logistic regression. Both accuracy and precision, along with recall values, are considerably high and balanced.

On the other hand, the GBM model has lower performance compared to other models. Especially, the low recall value indicates difficulties in detecting the spam class.

Despite having high accuracy, the KNN model shows low precision and F1 score values. The model faces challenges, especially in accurately identifying the spam class. The KNN model is more successful on small data sets. We may have encountered such a result because the dataset is not small.

The MLP model stands out with the highest performance. With high accuracy, precision, recall, and F1 score, it successfully distinguishes between spam and non-spam classes.