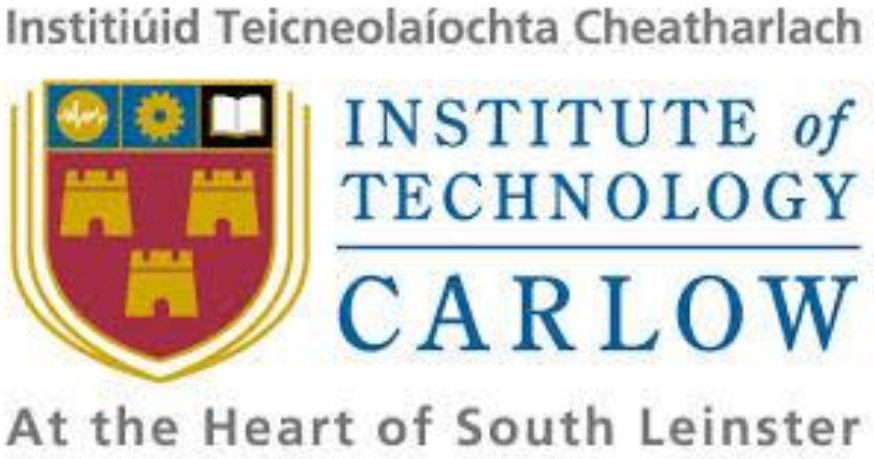


Email Spam Filter using Machine Learning

Technical Manual

Author: Hazel Murphy

Student ID: C00230058

Project Supervisor: James Egan

Date: Monday 19th April 2021

Abstract

The purpose of this project is to develop an email spam filter tool. It allows users to feel more confident about receiving emails as any spam emails will display in a separate folder to their inbox. This is achieved using machine learning algorithms such as Naïve Bayes, SVM, Random Forest, and Logistic Regression. The project used tools and technologies such as: hMailServer, Thunderbird, SQLAlchemy, Flask and python.

Table of Contents

Introduction

This document contains all the code required for this application. This application is hosted on a local server. In this project four machine learning models have been implemented which are integrated into a scraping function. This function pulls all the emails from the mail server, scores, and stores these emails for later use by the flask application. The web application files are all python files. Some initial configuration is required to set up the database. Tools such as hMailServer, Thunderbird and SQLAlchemy all need to be downloaded and installed before implementing any of the below code.

Tools and Technologies

The following tools and technologies were used to create the application:

Tools:

- MySQL Workbench (https://www.mysql.com/products/workbench/)
- hMailServer (https://www.hmailserver.com/download)
- Thunderbird (https://www.thunderbird.net/en-US/)

Technologies:

- MySQL server
- Flask
- Python
- SQLAlchemy
- Scikit-learn
- Pickle
- Matplotlib

Versions

The versions of the languages and tools used are as follows:

- MySQL Workbench Version 8.0.23
- hMailServer 5.6.7-B2425
- Python 3.9.1
- Thunderbird 78.9.0
- MySQL server 8.0.24

Database Creation

In the below screenshot the following line of code is used for initiating the database and connecting to local server using login details.

(db = SQLAlchemy(app))

```
app = Flask(__name__)
app.config['SECRET_KEY'] = '35f6d593ea98517a248e1e14f958b2f6'
app.config['SQLALCHEMY_DATABASE_URI'] = 'mysql://root:hazel9@127.0.0.1/pythonlogin'  #setting up database
db = SQLAlchemy(app) #database
migrate = Migrate(app, db)
```

This is the models file, which is used with SQLAlchemy an object relation mapper tool for database design directly from my code. My database design my created completely through Python. My database is a MySQL database and is located on my local server.

model.py

```
from datetime import datetime #datetime is the module and the attribute you are using of the module is also called datetime
from itsdangerous import TimedJSONWebSignatureSerializer as Serializer
from main import db, login_manager, app
from flask_login import UserMixin
from flask_sqlalchemy import SQLAlchemy


#login extension
@login_manager.user_loader #specifies decarater
def load_user(user_id):
    return User.query.get(int(user_id)) #gets the user with that id

#Each class has its own table in the database

#User table - all registered users stored here
class User(db.Model, UserMixin): #inherting from db.Model and UserMixin
    id = db.Column(db.Integer, primary_key=True)
    username = db.Column(db.String(20), unique=True, nullable=False)
    email = db.Column(db.String(120), unique=True, nullable=False)
    password = db.Column(db.String(60), nullable=False)
    posts = db.relationship('Post', backref='author', lazy='dynamic') #Post attribute has a relationship to the POST model (lazy load the data in one
    is_admin = db.Column(db.Boolean, default=False)

#Post table - all sent emails stored here from the flask app
class Post(db.Model):
    id = db.Column(db.Integer, primary_key=True)
    title = db.Column(db.String(100), nullable=False)
    date_posted = db.Column(db.DateTime, nullable=False, default=datetime.utcnow)
    content = db.Column(db.Text, nullable=False)
    user_id = db.Column(db.Integer, db.ForeignKey('user.id'), nullable=False) #relationship here we are ref the table name and column name

    def __repr__(self):
        return f"Post('{self.title}', {self.date_posted}')"

#Email table - all sent emails stored here from the thunderbird and each email has a spam score
class Email(db.Model):
    id = db.Column(db.String(255), primary_key=True)
    to =  db.Column(db.String(100), nullable=False)
    sender =  db.Column(db.String(100), nullable=False)
    subject =  db.Column(db.String(300), nullable=False)
    body =  db.Column(db.String(8000), nullable=False)
    date_sent =  db.Column(db.DateTime, nullable=False)
    naive_bayes_spam =  db.Column(db.Boolean, default=False)
    svm_spam = db.Column(db.Boolean, default=False)
    random_forest_spam = db.Column(db.Boolean, default=False)
    logistic_regression_spam = db.Column(db.Boolean, default=False)

```

```
    def __repr__(self):
        return f"Email('{self.to}', {self.sender}', '{self.subject}', '{self.body}', '{self.date_sent}')"
```

Training the models

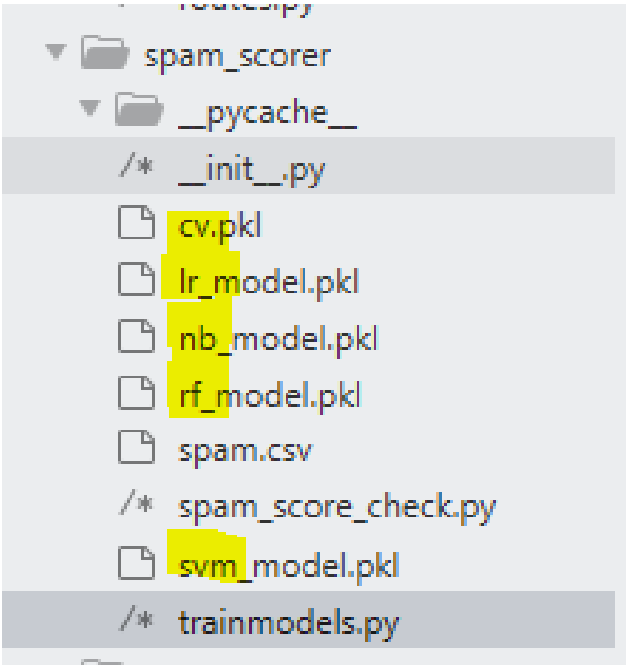
trainmodels.py

```python
import pandas as pd
from sklearn.feature_extraction.text import CountVectorizer
from sklearn.naive_bayes import MultinomialNB, GaussianNB
from sklearn import svm
from sklearn.ensemble import RandomForestClassifier
from sklearn.model_selection import GridSearchCV
from sklearn.linear_model import LogisticRegression
from sklearn.metrics import precision_score, recall_score, confusion_matrix, f1_score #does the sensisitvity need to be imported
import pickle

##Step1: Load Dataset

dataframe = pd.read_csv("spam.csv")
print(dataframe.describe())

##Step2: Split in to Training and Test Data

x = dataframe["EmailText"] #column label
y = dataframe["Label"]

x_train,y_train = x[0:4457],y[0:4457] #4457 rows to training set remaining to test set
x_test,y_test = x[4457:],y[4457:]

##Step3: Extract Features
cv = CountVectorizer()


#The CV fits the emailtext data and then trnasforms the data into vectorized features

cv.fit(x_train)
features = cv.transform(x_train)

##Step4: Initiate and build different models

tuned_parameters = {'kernel': ['rbf','linear'], 'gamma': [1e-3, 1e-4], #set of parameters for SVM
                    'C': [1, 10, 100, 1000]}

#Initzliae ski-learn
model = GridSearchCV(svm.SVC(), tuned_parameters)
model2 = MultinomialNB()
model3 = RandomForestClassifier(n_estimators=100)
model4 = LogisticRegression()

#Giving each feature the emailtext data and labels
model.fit(features,y_train)
model2.fit(features, y_train)
model3.fit(features, y_train)
model4.fit(features, y_train)
```

trainmodels.py

```python
#initiate pickle file names
cv_file = 'cv.pkl'
svm_file = 'svm_model.pkl'
nb_file = 'nb_model.pkl'
rf_file = 'rf_model.pkl'
lr_file = 'lr_model.pkl'


#dump models into pickle files for quick access
pickle.dump(cv, open(cv_file, 'wb'))
print('CV has loaded')
pickle.dump(model, open(svm_file, 'wb'))
pickle.dump(model2, open(nb_file, 'wb'))
pickle.dump(model3, open(rf_file, 'wb'))
pickle.dump(model4, open(lr_file, 'wb'))
print('models are loaded')
```

These are the pickle files in my project. They are used to access the models quicker.



Scrape function

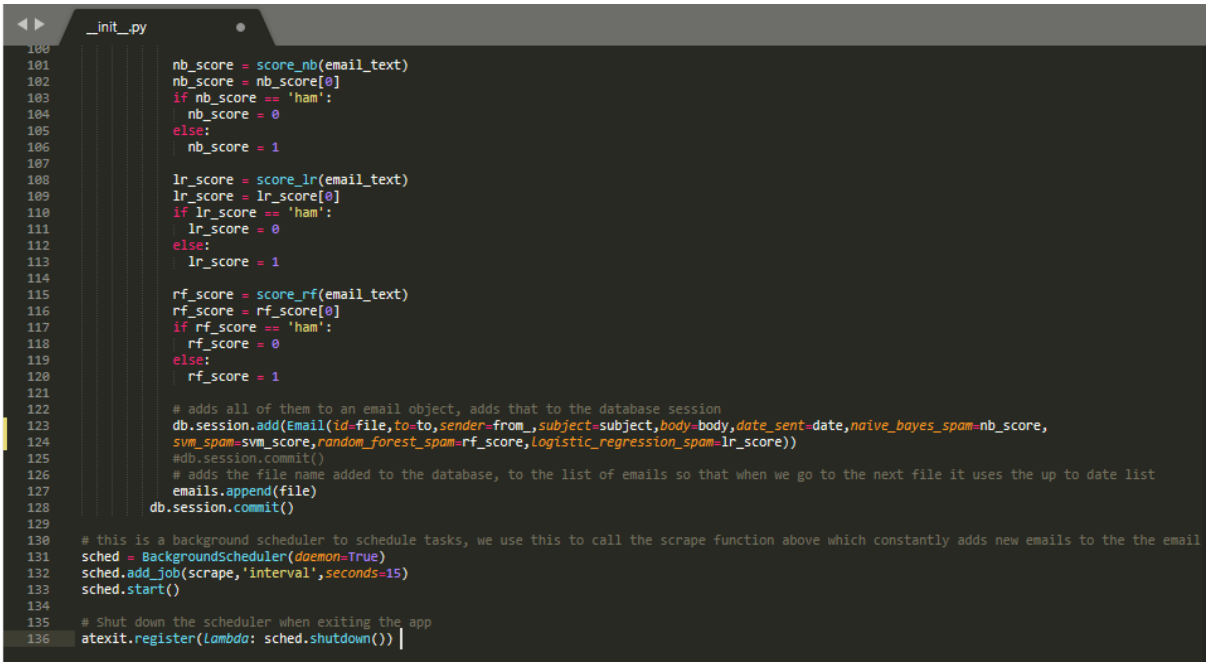
The below code is used to scrape emails for hMailServer into the database.

This code scans through the directories of the local mail server, and for each email which has not already been stored it transforms the emails data for storage and then calls upon each of the model's functions created and inserts the data into the database.

The four models were trained using the following dataset: https://www.kaggle.com/uciml/sms-spam-collection-dataset. I then split the data set into a training set and a testing set. The four models were trained using the training set.

init.py

```
from main.model import Email
from .spam_scorer import score_svm, score_nb, score_lr, score_rf

rootdir = 'C:\Program Files (x86)\hMailServer\Data'

def scrape():
  # queries email table to get all ids from the email table, the ids are the file names i the data folder
    emails = db.session.query(Email.id).all()
    # this ends the connection with the sql database
    db.session.commit()
    # this puts all of the query data into a list, so essentially puts all the file names in list
    emails = [value for (value,) in emails]
    # this is a for loop which, walks through all of the directories and sub directories of the rootdir variable above
    for subdir, dirs, files in os.walk(rootdir):
        # for each file in all of the files it is walking through
        for file in files:
          #if the file is not in the list of emails that we made above then do the following
         if file not in emails:
          # if the file name ends with eml do the following
          if str.lower(file[-3:])=="eml":
            # adds the full filepath to the a variable called file_path
            file_path = os.path.join(subdir,file)
            # gets the mail from the file_path, parses and puts it into a variable
            mail = mailparser.parse_from_file(file_path)
            # just gets each of the values we need and adds to variables
            to = mail.to[0][1]
            from_ = mail.from_[0][1]
            body = mail.body
            body = body.strip()
            subject = mail.subject
            date = mail.date
            file = str(file)
            email_text = subject + ' ' + body
            email_text = str(email_text)
            email_text = [email_text]


            svm_score = score_svm(email_text)
            svm_score = svm_score[0]
            if svm_score == 'ham':
              svm_score = 0
            else:
              svm_score = 1

```



```
_init_.py
                nb_score = score_nb(email_text)
                nb_score = nb_score[0]
                if nb_score == 'ham':
                  nb_score = 0
                else:
                  nb_score = 1

                lr_score = score_lr(email_text)
                lr_score = lr_score[0]
                if lr_score == 'ham':
                  lr_score = 0
                else:
                  lr_score = 1

                rf_score = score_rf(email_text)
                rf_score = rf_score[0]
                if rf_score == 'ham':
                  rf_score = 0
                else:
                  rf_score = 1

                # adds all of them to an email object, adds that to the database session
                db.session.add(Email(id=file,to=to,sender=from_,subject=subject,body=body,date_sent=date,naive_bayes_spam=nb_score,
                svm_spam=svm_score,random_forest_spam=rf_score,logistic_regression_spam=lr_score))
                #db.session.commit()
                # adds the file name added to the database, to the list of emails so that when we go to the next file it uses the up to date list
                emails.append(file)
            db.session.commit()

# this is a background scheduler to schedule tasks, we use this to call the scrape function above which constantly adds new emails to the the email
sched = BackgroundScheduler(daemon=True)
sched.add_job(scrape,'interval',seconds=15)
sched.start()

# Shut down the scheduler when exiting the app
atexit.register(lambda: sched.shutdown())
```

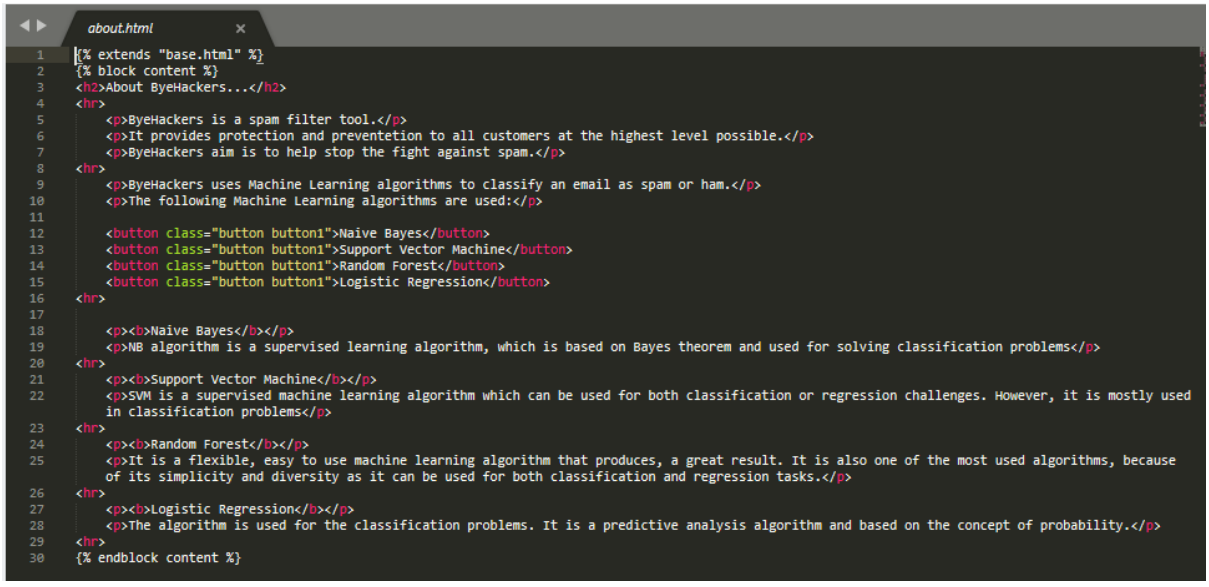
Web Application Pages

The below set of screenshots displays the code used to implement each page on the flask application.

Home.html

```
{% extends "base.html" %}
{% block content %}
      {% if current_user.is_authenticated %}
            <p></p>
      {% else %}
            <h2>Welcome to ByeHackers...</h2>
            <hr>
            <div class="text-center">
                  <img src="{{url_for('static', filename='spam_filter_home.png')}}" class="rounded"/>
            </div>
             <hr>
            <p>Find out more about ByeHackers, <button class="btn btn1-default"><a href="{{ url_for('main1.about') }}"> CLICK HERE</a></p></button>
            <hr>
            <p>Choose one of the following options to access the application:</p>
            <button class="btn btn-default"><a href="{{ url_for('users.register') }}"><span class="glyphicon glyphicon-log-in"></span> Register</a></
            button>    
            <button class="btn btn-default"><a href="{{ url_for('users.login') }}"><span class="glyphicon glyphicon-log-in"></span> Login</a></button
            >
      {% endif %}
            <hr>
            <p></p>
{% endblock content %}
```

About.html



```
{% extends "base.html" %}
{% block content %}
<h2>About ByeHackers...</h2>
<hr>
    <p>ByeHackers is a spam filter tool.</p>
    <p>It provides protection and preventetion to all customers at the highest level possible.</p>
    <p>ByeHackers aim is to help stop the fight against spam.</p>
<hr>
    <p>ByeHackers uses Machine Learning algorithms to classify an email as spam or ham.</p>
    <p>The following Machine Learning algorithms are used:</p>

    <button class="button button1">Naive Bayes</button>
    <button class="button button1">Support Vector Machine</button>
    <button class="button button1">Random Forest</button>
    <button class="button button1">Logistic Regression</button>
<hr>

    <p><b>Naive Bayes</b></p>
    <p>NB algorithm is a supervised learning algorithm, which is based on Bayes theorem and used for solving classification problems</p>
<hr>
    <p><b>Support Vector Machine</b></p>
    <p>SVM is a supervised machine learning algorithm which can be used for both classification or regression challenges. However, it is mostly used
    in classification problems</p>
<hr>
    <p><b>Random Forest</b></p>
    <p>It is a flexible, easy to use machine learning algorithm that produces, a great result. It is also one of the most used algorithms, because
    of its simplicity and diversity as it can be used for both classification and regression tasks.</p>
<hr>
    <p><b>Logistic Regression</b></p>
    <p>The algorithm is used for the classification problems. It is a predictive analysis algorithm and based on the concept of probability.</p>
<hr>
{% endblock content %}
```

Register.html

register.html

```
{% extends "base.html" %}
{% block content %}
<div class="forms">
    </div>
    <div class="content-section">
        <div class="col-sm-8 text-left">
        <form method="POST" action="">
            {{ form.hidden_tag() }} <!--Protects against XSS and CSRF attacks-->
            <fieldset class="form-group">
                <legend class="border-bottom mb-4">Register</legend>

                <!--Username Form group-->
                <div class="form-group">
                    {{ form.username.label(class="form-control-label")}} <!--Prints username label and bootstrap added to make it look nicer-->

                    <!--Print validation errors-->
                    <!--checking if the data entered is valid in the form and if not display errors to the user-->
                    {% if form.username.errors %}
                        {{ form.username(class="form-control form-control-lg is-invalid") }}
                        <div class="invalid-feedback">
                            {% for error in form.username.errors %}
                                <span>{{ error }}</span> <!--prints out the errors-->
                            {% endfor %}
                        </div>
                    {% else %}
                        {{ form.username(class="form-control form-control-lg")}}
                    {% endif %}
                </div>

                <!--Email Form group-->
                <div class="form-group">
                    {{ form.email.label(class="form-control-label")}} <!--Prints username label-->
                    <!--Print validation errors-->
                    <!--checking if the data entered is valid in the form and if not display errors to the user-->
                    {% if form.email.errors %}
                        {{ form.email(class="form-control form-control-lg is-invalid") }}
                        <div class="invalid-feedback">
                            {% for error in form.email.errors %}
                                <span>{{ error }}</span> <!--prints out the errors-->
                            {% endfor %}
                        </div>
                    {% else %}
                        {{ form.email(class="form-control form-control-lg")}}
                    {% endif %}
                </div>

                <!--Password Form group-->
                <div class="form-group">
                    {{ form.password.label(class="form-control-label")}} <!--Prints username label and bootstrap added to make it look nicer-->
                    <!--Print validation errors-->
                    <!--checking if the data entered is valid in the form and if not display errors to the user-->
                    {% if form.password.errors %}
                        {{ form.password(class="form-control form-control-lg is-invalid") }}
                        <div class="invalid-feedback">
                            {% for error in form.password.errors %}
                                <span>{{ error }}</span> <!--prints out the errors-->
                            {% endfor %}
                        </div>
                    {% else %}
                        {{ form.password(class="form-control form-control-lg")}}
                    {% endif %}
                </div>

                <!--Confirm Password Form group-->
                <div class="form-group">
                    {{ form.confirm_password.label(class="form-control-label")}} <!--Prints username label and bootstrap added to make it look nicer
                    -->
                    <!--Print validation errors-->
                    <!--checking if the data entered is valid in the form and if not display errors to the user-->
                    {% if form.confirm_password.errors %}
                        {{ form.confirm_password(class="form-control form-control-lg is-invalid") }}
                        <div class="invalid-feedback">
                            {% for error in form.confirm_password.errors %}
                                <span>{{ error }}</span> <!--prints out the errors-->
                            {% endfor %}
                        </div>
                    {% else %}
                        {{ form.confirm_password(class="form-control form-control-lg")}}
                    {% endif %}
                </div>
            </fieldset>
            <!--Submit Form group-->
            <div class="form-group">
                {{ form.submit(class="btn btn-outline-info")}} <!-- this will display as a button-->
            </div>
            <hr>
            <div class="form-group">
        <small class="text-muted"> <!--faded out text-->
            Already Have an Account? <a class="ml-2" href="{{ url_for('users.login') }}">Sign In</a> <!--link to login page, need to pass the name
            of the function not the name of the route so login is the name of the function-->
        </small>

                </div>
            </form>
        </div>
    </div>
    {% endblock content %}
```

Login.html

login.html

```
{% extends "base.html" %}
{% block content %}
<div class="forms">
    </div>
    <div class="content-section">
        <div class="col-sm-8 text-left">
        <form method="POST" action="">
            {{ form.hidden_tag() }} <!--Protects against XSS and CSRF attacks-->
            <div></div>
            <fieldset class="form-group">
                <legend class="border-bottom mb-4"> Login</legend>

                <!--Email Form group-->
                <div class="form-group">
                    {{ form.email.label(class="form-control-label")}} <!--Prints username label and bootstrap added-->
                    <!--Print validation errors-->
                    <!--checking if the data entered is valid in the form and if not display errors to the user-->
                    {% if form.email.errors %}
                        {{ form.email(class="form-control form-control-lg is-invalid") }}
                        <div class="invalid-feedback">
                            {% for error in form.email.errors %}
                                <span>{{ error }}</span> <!--prints out the errors-->
                            {% endfor %}
                        </div>
                    {% else %}
                        {{ form.email(class="form-control form-control-lg")}}
                    {% endif %}
                </div>

                <!--Password Form group-->
                <div class="form-group">
                    {{ form.password.label(class="form-control-label")}} <!--Prints username label -->
                    <!--Print validation errors-->
                    <!--checking if the data entered is valid in the form and if not display errors to the user-->
                    {% if form.password.errors %}
                        {{ form.password(class="form-control form-control-lg is-invalid") }}
                        <div class="invalid-feedback">
                            {% for error in form.password.errors %}
                                <span>{{ error }}</span> <!--prints out the errors-->
                            {% endfor %}
                        </div>
                    {% else %}
                        {{ form.password(class="form-control form-control-lg")}}
                    {% endif %}
                </div>

                    <!--Remember Me Field, this will be a bootsrap class-->
                    <div class="form-check">
                        {{ form.remember(class="form-check-input") }}
                        {{ form.remember.label(class="form-check-label") }}
                    </div>
                </fieldset>

                <!--Login-->
                <div class="form-group">
                    {{ form.submit(class="btn btn-outline-info")}}<!-- this will display as a button-->
                    <hr>
                    <!--Forgot Password-->
                    <small class="text-muted ml-2">
                        <a href="{{ url_for('users.reset_request') }}">Forgot Password?</a>
                    </small>
                </div>

                <hr>
                <div class="form-group">

                <small class="text-muted"> <!--faded out text-->
                Need an Account? <a class="ml-2" href="{{ url_for('users.register') }}">Sign Up Now</a>
            </small>
        </div>
            </form>
        </div>
    </div>
    {% endblock content %}
```

Welcome.html

welcome.html

```
{% extends "base.html" %}
{% block content %}

<h1>Welcome, {{ current_user.username }}</h1>
<p>You are now logged in and can begin using the application...</p>
{% endblock content %}
```

Create_post.html

create_post.html

```
{% extends "base.html" %}
{% block content %}
<br>
    <div class="content-section" align="left">
        <form method="POST" action="">
            {{ form.hidden_tag() }} <!--Protects against XSS and CSRF attacks-->
            <fieldset class="col-sm-8 form-group">
                <legend class="border-bottom mb-4">{{ legend }}</legend>

                <!--Recipient Form group-->
                <div class="form-group">
                    {{ form.to.label(class="form-control-label")}} <!--Prints TO label-->
                    <!--Print validation errors-->
                    <!--checking if the data entered is valid in the form and if not display errors to the user-->
                    {% if form.to.errors %}
                        {{ form.to(class="form-control form-control-lg is-invalid") }}
                        <div class="invalid-feedback">
                            {% for error in form.to.errors %}
                                <span>{{ error }}</span> <!--prints out the errors-->
                            {% endfor %}
                        </div>
                    {% else %}
                        {{ form.to(class="form-control form-control-lg")}}
                    {% endif %}
                </div>

                <!--Title Form group-->
                <div class="form-group">
                    {{ form.title.label(class="form-control-label")}} <!--Prints Subject label-->
                    <!--Print validation errors-->
                    <!--checking if the data entered is valid in the form and if not display errors to the user-->
                    {% if form.title.errors %}
                        {{ form.title(class="form-control form-control-lg is-invalid") }}
                        <div class="invalid-feedback">
                            {% for error in form.title.errors %}
                                <span>{{ error }}</span> <!--prints out the errors-->
                            {% endfor %}
                        </div>
                    {% else %}
                        {{ form.title(class="form-control form-control-lg")}}
                    {% endif %}
                </div>

```

```
                <div class="form-group">
                    {{ form.content.label(class="form-control-label")}} <!--Prints Body label-->
                    <!--Print validation errors-->
                    <!--checking if the data entered is valid in the form and if not display errors to the user-->
                    {% if form.content.errors %}
                        {{ form.content(class="form-control form-control-lg is-invalid") }}
                        <div class="invalid-feedback">
                            {% for error in form.content.errors %}
                                <span>{{ error }}</span> <!--prints out the errors-->
                            {% endfor %}
                        </div>
                    {% else %}
                        {{ form.content(class="form-control form-control-lg")}}
                    {% endif %}
                </div>
            </fieldset>

            <!--Submit Form group-->
            <div class="form-group">
                {{ form.submit(class="btn btn-outline-info")}} <!-- this will display the "Send" button-->
            </div>
        </form>
    </div>
{% endblock content %}
```

Inbox_mail.html

inbox_mail.html

```
{% extends "base.html" %}
{% block content %}

<div class="h2">Inbox</h2></div>
<hr>
<table class="myTable">
  <tr>
    <th>Date & Time</th>
    <th>Sender</th>
    <th>Subject</th>
    <th>Body</th>
  </tr>
  {% for email in query_set %}
  <tr>
    <td>{{email.date_sent}}</td>
    <td>{{email.sender}}</td>
    <td>{{email.subject}}</td>
    <td>{{email.body}}</td>
  </tr>
  {% endfor %}
</table>
{% endblock content %}
```

Sent_mail.html

sent.html

```
{% extends "base.html" %}
{% block content %}

<div class="h2">Sent</h2></div>
<hr>
<table class="myTable">
  <tr>
    <th>Date & Time</th>
    <th>Recipient</th>
    <th>Subject</th>
    <th>Body</th>
  </tr>
  {% for email in query_set %}
  <tr>
    <td>{{email.date_sent}}</td>
    <td>{{email.to}}</td>
    <td>{{email.subject}}</td>
    <td>{{email.body}}</td>
  </tr>
  {% endfor %}
</table>
{% endblock content %}
```

Spam_mail.html

spam_mail.html

```
{% extends "base.html" %}
{% block content %}

<div class="h2">Spam Mail</h2></div>
<hr>
<table class="myTable">
  <tr>
    <th>Date & Time</th>
    <th>Sender</th>
    <th>Subject</th>
    <th>Body</th>
    <th>Naive Bayes Score</th>
  </tr>
  {% for email in query_set %}
  <tr>
    <td>{{email.date_sent}}</td>
    <td>{{email.sender}}</td>
    <td>{{email.subject}}</td>
    <td>{{email.body}}</td>
    <td style="background-color:red;color:white;">{{email.naive_bayes_spam}}</td>
  </tr>
  {% endfor %}
</table>
{% endblock content %}
```

Account.html

account.html

```
{% extends "base.html" %}
{% block content %}
    <div class="content-section" align="left">
        <form method="POST" action="" enctype="multipart/form-data"> <!--add encoding type so image data is passed correct within the form-->
            {{ form.hidden_tag() }} <!--Protects against XSS and CSRF attacks-->
            <fieldset class="col-sm-8 form-group">
                <div class="h2">Account Information</h2></div>
                <hr>
                <!--Username Form group-->
                <div class="col-sm-8 form-group">
                    {{ form.username.label(class="form-control-label")}} <!--Prints username label-->
                        {{ form.username(class="form-control form-control-lg", disabled=True)}}
                </div>

                <!--Email Form group-->
                <div class="col-sm-8 form-group">
                    {{ form.email.label(class="form-control-label")}} <!--Prints email label-->
                        {{ form.email(class="form-control form-control-lg", disabled=True)}}
                </div>
            </fieldset>
        </form>
    </div>
{% endblock content %}
```

Spam_report.html

spam_report.html

```
{% extends "base.html" %}
{% block content %}


<div class="h2">Spam Report #1</h2></div>
<table class="myTable">
  <tr>
    <th>Total spam received in the last 7 days.</th>
  </tr>
  <tr>
    <td>{{query_set}}</td>
  </tr>
</table>
<hr>

<div class="h2">Spam Report #2</h2></div>
<p>Accounts who have received spam and the amount they have received.</p>
<table class="myTable">
  <tr>
    <th>Account Name</th>
    <th>Spam Total</th>
  </tr>
  {% for row in spam_by_account%}
  <tr>
    <td>{{row.to}}</td>
    <td>{{row.count}}</td>
  </tr>
  {% endfor %}
</table>

<img src="/plot.png" alt="my plot" height="400px">
<hr>
```

```
33
34    <div class="h2">Spam Report #3</h2></div>
35    <p>Emails received that have been classified as spam.</p>
36    <table class="myTable">
37      <tr>
38        <th>Date received</th>
39        <th>Subject</th>
40        <th>Body</th>
41        <th>Naive Bayes Score</th>
42      </tr>
43      {% for spam in spam_email%}
44      <tr>
45        <td>{{spam.date_sent}}</td>
46        <td>{{spam.subject}}</td>
47        <td>{{spam.body}}</td>
48        <td>{{spam.naive_bayes_spam}}</td>
49      </tr>
50      {% endfor %}
51    </table>
52    {% endblock content %}
```

Registered_user.html

registered_users.html

```
1   {% extends "base.html" %}
2   {% block content %}
3
4   <!--Displays all the registered users-->
5   <div class="h2">Registered Users</h2></div>
6   <hr>
7   <table class="myTable">
8     <tr>
9       <th>Username</th>
10      <th>Email Address</th>
11    </tr>
12    {% for user in user_set%}
13      <tr>
14      <td>{{user.username}}</td>
15      <td>{{user.email}}</td>
16    </tr>
17  {% endfor %}
18  </table>
19  {% endblock content %}
20
```

Error messages

403.html

403.html

```
{% extends "base.html" %}
{% block content %}
    <div class="content-section">
        <h1>You don't have permission to do that (403)</h1>
        <p>Please check your account and try again.</p>
    </div>
{% endblock content %}
```

404.html

404.html

```
{% extends "base.html" %}
{% block content %}
    <div class="content-section">
        <h1>Oops.. Page Not Found (404)</h1>
        <p>That page does not exist. Please try a different location</p>
    </div>
{% endblock content %}

```

500.html

500.html

```
{% extends "base.html" %}
{% block content %}
    <div class="content-section">
        <h1>Something went wrong (500)</h1>
        <p>We are experiencing some trouble on our end. Please try again in the near future.</p>
    </div>
{% endblock content %}
```

Styling for the webpages

This is the base template that was used across all page within the flask application.

Base.html

base.html

```
<!DOCTYPE html>
<html lang="en">
<head>
  <meta charset="utf-8">
  <meta name="viewport" content="width=device-width, initial-scale=1">
  <link rel="stylesheet" href="https://maxcdn.bootstrapcdn.com/bootstrap/3.4.1/css/bootstrap.min.css">
  <script src="https://ajax.googleapis.com/ajax/libs/jquery/3.5.1/jquery.min.js"></script>
  <script src="https://maxcdn.bootstrapcdn.com/bootstrap/3.4.1/js/bootstrap.min.js"></script>

  {% if title %}
    <title>ByeHackers - {{ title }}</title>
  {% else %}
    <title>ByeHackers</title>
  {% endif %}
  <style>

    /* Remove the navbar's default margin-bottom and rounded borders */
    .navbar {
      margin-bottom: 0;
      border-radius: 0;
    }

    .form-control-label {
      text-align: left;
    }

    /* Set height of the grid so .sidenav can be 100% (adjust as needed) 450px*/
    .row.content {height: 790px}

    /* Set gray background color and 100% height */
    .sidenav {
      padding-top: 18px;
      background-color: #f1f1f1;
      height: 100%;
      position: relative;
    }

    .forms {
      padding: 15px;
    }

    body {
      height: 100%;
      min-height: 100%;
      overflow: auto;
    }

```

base.html

```
47
48    /* Set black background color, white text and some padding */
49    footer {
50      background-color: #555;
51      color: white;
52      padding: 10px;
53      clear: both;
54      position: relative;
55      height: 60px;
56      margin-top: -60px;
57      width: 100%;
58    }
59
60    /* On small screens, set height to 'auto' for sidenav and grid */
61    @media screen and (max-width: 767px) {
62      .sidenav {
63        height: 100%;
64        padding: 15px;
65      }
66      .row.content {height:auto;}
67    }
68
69    .btn-default {
70      background: #000;
71      color: #fff;
72      box-shadow: 1px 2px 5px #000000;
73      width: 48%;
74
75      /*float: center;*/
76    }
77
78    .btn1-default {
79      background: #000;
80      color: #fff;
81      box-shadow: 1px 2px 5px #000000;
82      width: 15%;
83    }
84
85    .btn-default:hover {
86      background: #FFFFFF;
87      color: #000;
88    }
89
90    .btn1-default:hover {
91      background: #FFFFFF;
92      color: #000;
93    }
94
```

base.html

```
94
95      .btn-outline-info {
96          width: 35%;
97          background: #808080;
98          cursor: pointer;
99          color: #FFFFFF;
100     }
101
102     .carousel-control.right, .carousel-control.left {
103     background-image: none;
104     color: #FF0000;
105     }
106
107     .carousel-indicators li {
108       border-color: #FF0000;
109     }
110
111     .carousel-indicators li.active {
112       background-color: #FF0000;
113     }
114
115     .item h4 {
116       font-size: 19px;
117       line-height: 1.375em;
118       font-weight: 400;
119       font-style: italic;
120       margin: 50px 0;
121     }
122
123     .item span {
124       font-style: normal;
125     }
126
127     .button {
128     align-items: center;
129     color: white;
130     padding: 20px 32px;
131     text-align: center;
132     font-weight: bold;
133     display: inline-block;
134     font-size: 16px;
135     margin: 4px 2px;
136   }
137
138   .button1 {
139     background-color: #ffeeed;
140     color: black;
141     border: 4px solid #FF0000;
```

base.html

```
}

.invalid-feedback {
  color: red;
  background: #ffe0e0;
  border: 1px solid #a33a3a;
}

.myTable {
  border: 0px solid #69899F;
  width: 100%;
  }

.myTable th {

padding:15px;
    color:#fff;
    text-shadow:1px 1px 1px #568F23;
    border-bottom:3px solid #9ED929;
    background-color:#9DD929;
    background:-webkit-gradient(
        linear,
        left bottom,
        left top,
        color-stop(0.02, rgb(123,192,67)),
        color-stop(0.51, rgb(139,198,66)),
        color-stop(0.87, rgb(158,217,41))
        );
    background: -moz-linear-gradient(
        center bottom,
        rgb(123,192,67) 3%,
        rgb(139,198,66) 52%,
        rgb(158,217,41) 88%
        );
    -webkit-border-top-left-radius:5px;
    -webkit-border-top-right-radius:5px;
    -moz-border-radius:5px 5px 0px 0px;
    border-top-left-radius:5px;
    border-top-right-radius:5px;
  }

.myTable td {
    width:100px;
    padding:10px;
    text-align:center;
    vertical-align: top;
    background-color:#DEF3CA;
    border: 1px solid #BED3AB;
```

base.html

```
    -moz-border-radius:2px;
    -webkit-border-radius:2px;
    border-radius:2px;
    color: black;
    text-shadow:1px 1px 1px #fff;
}

.h2 {
  text-align: left;
  color: black;
}

.form-group {
  align-content: left;
}
</style>
</head>
<body>

<nav class="navbar navbar-inverse"> <!-- puts the black colour into the nav bar-->
  <div class="container-fluid">
      <a class="navbar-brand disabled">ByeHackers</a>
    <div class="collapse navbar-collapse" id="myNavbar">
      <ul class="nav navbar-nav">
        {% if current_user.is_authenticated %}
        <li><a href="{{ url_for('main1.welcome') }}"><span class="glyphicon glyphicon-ok"> Authenicated</a></li>
        {% else %}
        <li><a href="{{ url_for('main1.home') }}"><span class="glyphicon glyphicon-home"> Home</a></li>
        <li><a href="{{ url_for('main1.about') }}"><span class="glyphicon glyphicon-info-sign"> About</a></li>
        {% endif %}
      </ul>
      <ul class="nav navbar-nav navbar-right">
        {% if current_user.is_authenticated %}
        <li><a href="{{ url_for('users.account') }}"><span class="glyphicon glyphicon-user"></span> Account</a></li>
        <li><a href="{{ url_for('users.logout') }}"><span class="glyphicon glyphicon-log-out"></span> Logout</a></li>
        {% else %}
        <li><a href="{{ url_for('users.login') }}"><span class="glyphicon glyphicon-log-in"></span> Login</a></li>
        <li><a href="{{ url_for('users.register') }}"><span class="glyphicon glyphicon-log-in"></span> Register</a></li>
        {% endif %}
      </ul>
    </div>
  </div>
</nav>
```

base.html

```
<div class="container-fluid text-center">
  <div class="row content">
    <div class="col-sm-2 sidenav">
      {% if current_user.is_authenticated %}
      <p><a href="{{ url_for('posts.new_post') }}">Compose </a><span class="glyphicon glyphicon-pencil"></span></p> <!-- posts.post.new -->
      <p><a href="{{ url_for('main1.inbox_mail') }}">Inbox </a><span class="glyphicon glyphicon-save"></span></p>
      <p><a href="{{ url_for('main1.sent') }}">Sent </a><span class="glyphicon glyphicon-open"></span></p>
      <p><a href="{{ url_for('main1.spam_mail') }}">Spam </a><span class="glyphicon glyphicon-exclamation-sign"></span></p>
      {% if current_user.is_admin %}
      <p><a href="{{ url_for('main1.spam_report') }}">Spam Report </a><span class="glyphicon glyphicon-list-alt"></span></p>
      <p><a href="{{ url_for('main1.registered_users') }}">Registered Users </a><span class="glyphicon glyphicon-list-alt"></span></p>
      {% endif %}
      <p><a href="{{ url_for('users.account') }}">Account </a><span class="glyphicon glyphicon-user"></span></p>
      <p><a href="{{ url_for('users.logout') }}">Logout </a><span class="glyphicon glyphicon-log-out"></span></p>
      <!--ADMIN USER DISPLAY SPAM REPORT AND ALL REGISTERED USERS OF THE WEB APP-->
      {% else %}
      <!--DONT DISPLAY THE SIDENAV IF THE USER IS NOT LOGGED IN-->
      {% endif %}
    </div>

    <div class="col-sm-8 text-left">
      <div class="col-md-8">
      <!--display flash messages-->
      {% with messages = get_flashed_messages(with_categories=true) %}
      <!--return any messages from the with flash function-->
      {% if messages %} <!-- if messages is not empty then we have flash messages to display-->
        {% for category, message in messages %}
          <div class="alert alert-{{ category }}"><!-- prints out the messages-->
            {{ message }}
          </div>
        {% endfor %}
      {% endif %}
      {% endwith %} <!--close endwith code block-->
    </div>

      <div id="content">
        {% block content %}{% endblock %}
      </div>
    </div>
    <div class="col-sm-2 sidenav"> <!--right side nav-->
      {% if current_user.is_authenticated %} <!--if the user is logged in-->
      <p></p>
      {% else %}
      <div class="well">
        <p><b>STATS</p></b>
      </div>
```

```
base.html
280         <div class="well">
281           <p>&#9888; 53.95% of e-mail traffic in March 2020 was spam.</p>
282         </div>
283         <div class="well">
284           <p>&#9888; Email spam costs businesses $20.5 billion every year.</p>
285         </div>
286         <div class="well">
287           <p>&#9888; Around 107 billion spam emails are sent per day. By 2024, the figure is expected to increase to over 361.6 billion daily mails.</
              p>
288         </div>
289         <div class="well">
290           <p>&#9888; Bulk mailers can send approximately 250,000 messages an hour over a 28.8kb/s modem line.</p>
291         </div>
292       </div>
293       {% endif %}
294     </div>
295   </div>
296
297   <footer class="container-fluid text-center"> <!--footer-->
298     <p>Copyright &#169 2021 byehackers.com. All rights reserved.</p>
299   </footer>
300
301   </body>
302   </html>
303
```

Error handlers

These files handle any errors that may occur on the application.

Handlers.py

```
handlers.py
1  from flask import Blueprint, render_template
2
3  errors = Blueprint('errors', __name__) #create Blueprint
4
5
6  @errors.app_errorhandler(404)
7  def error_404(error):
8      return render_template('errors/404.html'), 404
9
10
11 @errors.app_errorhandler(403)
12 def error_403(error):
13     return render_template('errors/404.html'), 403
14
15
16 @errors.app_errorhandler(500)
17 def error_500(error):
18     return render_template('errors/404.html'), 500
19
```

Main

A route file is created in flask to inform the flask app which URL should initiate our function.

Routes.py

routes.py

```python
from flask import render_template, request, Blueprint
from flask_login import current_user, login_required
from main.model import Post, Email, User
import datetime
from sqlalchemy import func

main1 = Blueprint('main1', __name__) #create blueprint

#Home route
@main1.route("/")
@main1.route("/home")
def home():
    page = request.args.get('page', 1, type=int)
    posts = Post.query.order_by(Post.date_posted.desc()).paginate(page=page, per_page=3) #orders the posts in a specfic order and shows 5 per page
    return render_template('home.html', posts=posts)

#About route
@main1.route("/about")
def about():
    return render_template('about.html', title='About')

#Welcome route
@main1.route("/welcome")
def welcome():
    return render_template('welcome.html', title='Welcome')

#Inbox route
@main1.route("/inbox_mail")
def inbox_mail():
    email_address = current_user.email
    query_set = Email.query.filter(Email.naive_bayes_spam != 1).filter(Email.to==email_address).order_by(Email.date_sent.desc()).all()
    for email in query_set:
        print(email.date_sent)
        print(email.sender)
        print(email.subject)
        print(email.body)
    return render_template('inbox_mail.html', title='Inbox Mail', query_set=query_set)
```

```python
#Sent route
@main1.route("/sent")
def sent():
    email_address = current_user.email
    query_set = Email.query.filter(Email.naive_bayes_spam != 1 and Email.sender==email_address).order_by(Email.date_sent.desc()).all()
    for email in query_set:
        print(email.date_sent)
        print(email.to)
        print(email.subject)
        print(email.body)
    return render_template('sent.html', title='Sent Mail', query_set=query_set)

#SpamReport route
@main1.route("/spam_report")
def spam_report():

    #Report 1
    today = datetime.date.today()
    seven_days_ago = today - datetime.timedelta(days=7)
    print(seven_days_ago)
    query_set = Email.query.filter(Email.naive_bayes_spam == 1).filter(Email.date_sent > seven_days_ago).count()
    print(query_set)

    #Report 2
    spam_by_account = Email.query.with_entities(Email.to, func.count(Email.id).label('count')).filter(Email.naive_bayes_spam == 1).group_by
    (Email.to).all()
    print(spam_by_account)

    #Report 3
    today = datetime.date.today()
    two_days_ago = today - datetime.timedelta(days=2)
    spam_email = Email.query.filter(Email.naive_bayes_spam == 1).filter(Email.date_sent > two_days_ago).all()

    return render_template('spam_report.html', title='Spam Report', query_set=query_set, spam_by_account=spam_by_account, spam_email=spam_email)
```

routes.py

```
76
77  #SpamMail route
78  @main1.route("/spam_mail")
79  def spam_mail():
80      email_address = current_user.email
81      query_set = Email.query.filter(Email.to == email_address).filter(Email.naive_bayes_spam == 1).order_by(Email.date_sent.desc()).all()
82      for email in query_set:
83          print(email.date_sent)
84          print(email.sender)
85          print(email.subject)
86          print(email.body)
87          print(email.naive_bayes_spam)
88      return render_template('spam_mail.html', title='Spam Mail', query_set=query_set)
89
90  #Registered_users route
91  @main1.route("/registered_users")
92  def registered_users():
93      user_set = User.query.all() #this line outputs the first 3 columns in the USERS table
94      print(user_set)
95
96      return render_template('registered_users.html', title='Registered Users', user_set=user_set)
```

Post

Forms.py

forms.py

```
1   from flask_wtf import FlaskForm
2   from wtforms import StringField, SubmitField, TextAreaField
3   from wtforms.validators import DataRequired
4
5   #Compose Mail form
6   class PostForm(FlaskForm):
7       to = StringField('To', validators=[DataRequired()])
8       title = StringField('Subject', validators=[DataRequired()])
9       content = TextAreaField('Body', validators=[DataRequired()])
10      submit = SubmitField('Send')
11
```

Routes.py

routes.py

```
1   from flask import render_template, url_for, flash, redirect, request, abort, Blueprint
2   from flask_login import current_user, login_required
3   from main import db
4   from main.model import Post
5   from main.posts.forms import PostForm
6
7   posts = Blueprint('posts', __name__) #create blueprint
8
9   #Compose a new mail route
10  @posts.route("/post/new", methods=['GET', 'POST'])
11  @login_required
12  def new_post():
13      form = PostForm()
14      if form.validate_on_submit():
15          post = Post(title=form.title.data, content=form.content.data, author=current_user) #create email
16          db.session.add(post) #add email to database
17          db.session.commit()
18          flash('Your email has been sent!', 'success')
19          return redirect(url_for('main1.home'))
20      return render_template('create_post.html', title='Compose Mail', form=form, legend='Compose Mail')
```

Spam Score

The below code prints out the following scores of the four machine learning models that were implemented: confusion matrix, accuracy, precision & recall, f1 and sensitivity and specificity scores.

Spam_score_check.py

spam_score_check.py

```python
import pandas as pd
from sklearn.feature_extraction.text import CountVectorizer
from sklearn.naive_bayes import MultinomialNB, GaussianNB
from sklearn import svm
from sklearn.ensemble import RandomForestClassifier
from sklearn.model_selection import GridSearchCV
from sklearn.linear_model import LogisticRegression
from sklearn.metrics import precision_score, recall_score, confusion_matrix, f1_score #does the sensisitvity need to be imported
import pickle

##Step1: Load Dataset

dataframe = pd.read_csv("spam.csv")
#print(dataframe.describe())

##Step2: Split in to Training and Test Data

x = dataframe["EmailText"]
y = dataframe["Label"]

x_train,y_train = x[0:4457],y[0:4457]
x_test,y_test = x[4457:],y[4457:]
y_true = y_test.tolist()

'''
##Step3: Extract Features
cv = CountVectorizer()


#Everything below has been used to train the models and dump the models into two seperate pickle files

cv.fit(x_train)
features = cv.transform(x_train)
'''
##Step4: Initiate and build different models
'''

tuned_parameters = {'kernel': ['rbf','linear'], 'gamma': [1e-3, 1e-4],
                    'C': [1, 10, 100, 1000]}

model = GridSearchCV(svm.SVC(), tuned_parameters)
model2 = MultinomialNB()
model3 = RandomForestClassifier(n_estimators=100)
model4 = LogisticRegression()

```

spam_score_check.py

```
model.fit(features,y_train)
model2.fit(features, y_train)
model3.fit(features, y_train)
model4.fit(features, y_train)

'''

#initiate pickle file names
cv_file = 'cv.pkl'
svm_file = 'svm_model.pkl'
nb_file = 'nb_model.pkl'
rf_file = 'rf_model.pkl'
lr_file = 'lr_model.pkl'
'''

#dump models into pickle files for quick access
pickle.dump(cv, open(cv_file, 'wb'))
print('CV has loaded')
pickle.dump(model, open(svm_file, 'wb'))
pickle.dump(model2, open(nb_file, 'wb'))
pickle.dump(model3, open(rf_file, 'wb'))
pickle.dump(model4, open(lr_file, 'wb'))
print('models are loaded')

'''
cv = pickle.load(open(cv_file, 'rb'))
loaded_svm_model = pickle.load(open(svm_file, 'rb'))
loaded_nb_model = pickle.load(open(nb_file, 'rb'))
loaded_rf_model = pickle.load(open(rf_file, 'rb'))
loaded_lr_model = pickle.load(open(lr_file, 'rb'))

prediction_set = x_test.tolist()
svm_predicted_set = []
nb_predicted_set = []
rf_predicted_set = []
lr_predicted_set = []
for row in prediction_set:
    row = [row]
    transform = cv.transform(row)
    svm_result = loaded_svm_model.predict(transform)
    nb_result = loaded_nb_model.predict(transform)
    rf_result = loaded_rf_model.predict(transform)
    lr_result = loaded_lr_model.predict(transform)
    svm_predicted_set.append(svm_result)
    nb_predicted_set.append(nb_result)
    rf_predicted_set.append(rf_result)
    lr_predicted_set.append(lr_result)

```

spam_score_check.py

```
94
95  #svm confusion matrix
96
97  svm_confusion_matrix = confusion_matrix(y_true, svm_predicted_set, labels=['ham','spam'])
98  svm_sensitivity = svm_confusion_matrix[0,0]/(svm_confusion_matrix[0,0]+svm_confusion_matrix[0,1])
99  svm_specifity = svm_confusion_matrix[1,1]/(svm_confusion_matrix[1,0]+svm_confusion_matrix[1,1])
100
101 #nb confusion matrix
102
103 nb_confusion_matrix = confusion_matrix(y_true, nb_predicted_set, labels=['ham','spam'])
104 nb_sensitivity = nb_confusion_matrix[0,0]/(nb_confusion_matrix[0,0]+nb_confusion_matrix[0,1])
105 nb_specifity = nb_confusion_matrix[1,1]/(nb_confusion_matrix[1,0]+nb_confusion_matrix[1,1])
106
107 #rf confusion matrix
108
109 rf_confusion_matrix = confusion_matrix(y_true, rf_predicted_set, labels=['ham','spam'])
110 rf_sensitivity = rf_confusion_matrix[0,0]/(rf_confusion_matrix[0,0]+rf_confusion_matrix[0,1])
111 rf_specifity = rf_confusion_matrix[1,1]/(rf_confusion_matrix[1,0]+rf_confusion_matrix[1,1])
112
113 #lr confusion matrix
114
115 lr_confusion_matrix = confusion_matrix(y_true, lr_predicted_set, labels=['ham','spam'])
116 lr_sensitivity = lr_confusion_matrix[0,0]/(lr_confusion_matrix[0,0]+lr_confusion_matrix[0,1])
117 lr_specifity = lr_confusion_matrix[1,1]/(lr_confusion_matrix[1,0]+lr_confusion_matrix[1,1])
118
119
120 #Get scores for svm model
121 print('SVM')
122 print('confusion matrix for SVM')
123 print(str(svm_confusion_matrix))
124 print('accuracy of svm = ' + str(loaded_svm_model.score(cv.transform(x_test),y_test)))
125 print('precision of svm for ham and spam = ' + str(precision_score(y_true, svm_predicted_set, average=None)))
126 print('recall of svm for ham and spam = ' + str(recall_score(y_true, svm_predicted_set, average=None)))
127 print('sensitivity and specifity for svm model =' + str(svm_sensitivity) + ' ' + str(svm_specifity))
128 print('f1 score of svm for ham and spam = ' + str(f1_score(y_true, svm_predicted_set, average=None)))
129 print('**********************************************************************')
130
131 #Get scores for nb model
132 print('NAIVE BAYES')
133 print('confusion matrix for NB')
134 print(str(svm_confusion_matrix))
135 print('accuracy of nb = ' + str(loaded_nb_model.score(cv.transform(x_test),y_test)))
136 print('precision of nb for ham and spam = ' + str(precision_score(y_true, nb_predicted_set, average=None)))
137 print('recall of nb for ham and spam = ' + str(recall_score(y_true, nb_predicted_set, average=None)))
138 print('sensitivity and specifity for nb model =' + str(nb_sensitivity) + ' ' + str(nb_specifity))
139 print('f1 score of nb for ham and spam = ' + str(f1_score(y_true, nb_predicted_set, average=None)))
140 print('**********************************************************************')
141
```

spam_score_check.py

```
141
142 #Get scores for rf model
143 print('RANDOM FOREST')
144 print('confusion matrix for RF')
145 print(str(svm_confusion_matrix))
146 print('accuracy of rf = ' + str(loaded_rf_model.score(cv.transform(x_test),y_test)))
147 print('precision of rf for ham and spam = ' + str(precision_score(y_true, rf_predicted_set, average=None)))
148 print('recall of rf for ham and spam = ' + str(recall_score(y_true, rf_predicted_set, average=None)))
149 print('sensitivity and specifity for rf model =' + str(rf_sensitivity) + ' ' + str(rf_specifity))
150 print('f1 score of rf for ham and spam = ' + str(f1_score(y_true, rf_predicted_set, average=None)))
151 print('**********************************************************************')
152
153 #Get scores for lr model
154 print('LOGISTIC REGRESSION')
155 print('confusion matrix for LR')
156 print(str(svm_confusion_matrix))
157 print('accuracy of lr = ' + str(loaded_lr_model.score(cv.transform(x_test),y_test)))
158 print('precision of lr for ham and spam = ' + str(precision_score(y_true, lr_predicted_set, average=None)))
159 print('recall of lr for ham and spam = ' + str(recall_score(y_true, lr_predicted_set, average=None)))
160 print('sensitivity and specifity for lr model =' + str(lr_sensitivity) + ' ' + str(lr_specifity))
161 print('f1 score of lr for ham and spam = ' + str(f1_score(y_true, lr_predicted_set, average=None)))
162 print('**********************************************************************')
163
164 #print(str(precision_recall_fscore_support(y_true, y_pred, average='macro')))
165
166 #this is where we insert single string and get svm + nb predictions
167 '''
168 email_text = ['You are a winner, you have won a free phone. Please signup at the following link. SALE SALE SALE FRIDAY FRIDAY. Text 50777']
169 transform = cv.transform(email_text)
170 result = loaded_svm_model.predict(transform)
171 result2 = loaded_nb_model.predict(transform)
172 result3 = loaded_rf_model.predict(transform)
173 result4 = loaded_lr_model.predict(transform)
174 print(result)
175 print(result2)
176 print(result3)
177 print(result4)
178 '''
```

This is the output for each score for the four machine learning models that were implemented.

```
(venv) C:\Users\C00230058\Desktop\HazelsProject\main\spam_scorer>py spam_score_check.py
SVM
confusion matrix for SVM
[[969   1]
 [ 13 132]]
accuracy of svm = 0.9874439461883409
precision of svm for ham and spam = [0.98676171 0.9924812 ]
recall of svm for ham and spam = [0.99896907 0.91034483]
sensitivity and specifity for svm model =0.9989690721649485 0.9103448275862069
f1 score of svm for ham and spam = [0.99282787 0.94964029]
**********************************************************************
NAIVE BAYES
confusion matrix for NB
[[962   8]
 [  8 137]]
accuracy of nb = 0.9856502242152466
precision of nb for ham and spam = [0.99175258 0.94482759]
recall of nb for ham and spam = [0.99175258 0.94482759]
sensitivity and specifity for nb model =0.9917525773195877 0.9448275862068966
f1 score of nb for ham and spam = [0.99175258 0.94482759]
**********************************************************************
RANDOM FOREST
confusion matrix for RF
[[968   2]
 [ 25 120]]
accuracy of rf = 0.9757847533632287
precision of rf for ham and spam = [0.97482377 0.98360656]
recall of rf for ham and spam = [0.99793814 0.82758621]
sensitivity and specifity for rf model =0.9979381443298969 0.8275862068965517
f1 score of rf for ham and spam = [0.98624554 0.8988764 ]
**********************************************************************
LOGISTIC REGRESSION
confusion matrix for LR
[[967   3]
 [ 15 130]]
accuracy of lr = 0.9838565022421525
precision of lr for ham and spam = [0.98472505 0.97744361]
recall of lr for ham and spam = [0.99690722 0.89655172]
sensitivity and specifity for lr model =0.9969072164948454 0.896551724137931
f1 score of lr for ham and spam = [0.99077869 0.9352518 ]
**********************************************************************
```

Users

All forms displayed on the flask application are coded in the forms.py file.

Forms.py

forms.py

```python
from flask_wtf import FlaskForm
from flask_wtf.file import FileField, FileAllowed
from wtforms import StringField, PasswordField, SubmitField, BooleanField, TextAreaField
from wtforms.validators import DataRequired, Length, Email, EqualTo, ValidationError
from flask_login import current_user
from main.model import User


# Registration Form
class RegistrationForm(FlaskForm):
    username = StringField('Username', validators=[DataRequired(), Length(min=2, max=20)])
    email = StringField('Email', validators=[DataRequired(), Email()])
    password = PasswordField('Password', validators=[DataRequired()])
    confirm_password = PasswordField('Confirm Password', validators=[DataRequired(), EqualTo('password')])
    submit = SubmitField('Sign Up')

    #Create a customize validation for the register form for username
    def validate_username(self, username):
        user = User.query.filter_by(username=username.data).first() #this filters to see if the username is already in the database
        if user:
            raise ValidationError('That username is taken. Please choose a different username')

    #Create a customize validation for the register form for email
    def validate_email(self, email):
        user = User.query.filter_by(email=email.data).first()
        if user:
            raise ValidationError('That email is taken. Please choose a different email')

#Login Form
class LoginForm(FlaskForm):
    email = StringField('Email', validators=[DataRequired(), Email()])
    password = PasswordField('Password', validators=[DataRequired()])
    remember = BooleanField('Remember Me')
    submit = SubmitField('Login')


#Account Form
class UpdateAccountForm(FlaskForm):
    username = StringField('Username', validators=[DataRequired(), Length(min=2, max=20)])
    email = StringField('Email', validators=[DataRequired(), Email()])

```

Routes.py

routes.py

```python
from flask import render_template, url_for, flash, redirect, request, Blueprint
from flask_login import login_user, current_user, logout_user, login_required
from main import db, bcrypt
from main.model import User, Post
from main.users.forms import RegistrationForm, LoginForm, UpdateAccountForm, RequestResetForm, ResetPasswordForm, PostForm
#from main.users.utils import save_picture, send_reset_email

users = Blueprint('users', __name__) #create blueprint

#Register route
@users.route("/register", methods=['GET', 'POST']) #adding methods to allow for POST request
def register():
    if current_user.is_authenticated: #if current user is logged in redirect them to the home page
        return redirect(url_for('main1.home'))
    form = RegistrationForm()
    #Check if the data entered is validate for the form
    if form.validate_on_submit():#CHECKING IF FORM IS VALIDATE ON SUBMIT
        #Hash Password
        hashed_password = bcrypt.generate_password_hash(form.password.data).decode('utf-8')
        user = User(username=form.username.data, email=form.email.data, password=hashed_password) #creates a new user
        db.session.add(user) #add this user to the changes in our db
        db.session.commit() #commit these changes this will add the user to the database
        flash('Your account has now been created! You are now able to log in', 'success') #flash one time alert message, f string is used as we are
        #redirect to the home page after a successful creation
        return redirect(url_for('users.login')) #login is the name of the function of the route, user is returned to login page
    return render_template('register.html', title='Register', form=form)


#Login route
@users.route("/login", methods=['GET', 'POST'])
def login():
    if current_user.is_authenticated: #if current user is logged in redirect them to the home page
        return redirect(url_for('main1.home'))
    form = LoginForm()
    if form.validate_on_submit():
        user = User.query.filter_by(email=form.email.data).first() #this will filter the emails entered into the form to see if this email exist alr
        if user and bcrypt.check_password_hash(user.password, form.password.data): #this will check the hash password with the password they enter i
            login_user(user, remember=form.remember.data) #this will login the user
            next_page = request.args.get('next') #get next parameter from url if user exists (get method)
            return redirect(next_page) if next_page else redirect(url_for('main1.welcome')) #redirect to the next page if the next page exists if it
        else:
            flash('Login unsuccessful. Please check email and password', 'danger') #danger will display a red alert
    return render_template('login.html', title='Login', form=form) #return login page
```

routes.py

```python
#Logout route
@users.route("/logout")
def logout():
    logout_user()
    return redirect(url_for('main1.home'))


#Account route
#This is used if a user clicks on a certain link and they need to be logged in order to view this data
@users.route("/account", methods=['GET', 'POST']) #allowing GET and POST requests
@login_required #this means we need to be logged in to access the account route
def account():
    form = UpdateAccountForm()
    #if form.validate_on_submit(): #validate form
        #if form.picture.data: #check for picture data
            #set users profile pic (create new function for this above in the code)
        #    picture_file = save_picture(form.picture.data)
        #    current_user.image_file = picture_file
        #current_user.username = form.username.data #we can update our username and email
        #current_user.email = form.email.data
        #db.session.commit()
        #flash('Your account has been updated!', 'success')
        #return redirect(url_for('users.account'))
    if request.method == 'GET': #populate form when user visits account page
        form.username.data = current_user.username
        form.email.data = current_user.email
        #form.password.data = current_user.password
    #image_file = url_for('static', filename='profile_pics/' + current_user.image_file) #set imagefile to static dir
    return render_template('account.html', title='Account', form=form) #image_file=image_file
```