

Delhi Technological University

Department of Information Technology



THEORY OF COMPUTATION (IT301)

SPAM-TEXT-MESSAGE-CLASSIFICATION

PROJECT REPORT

Submitted to:

Ms. Sunakshi Mehra

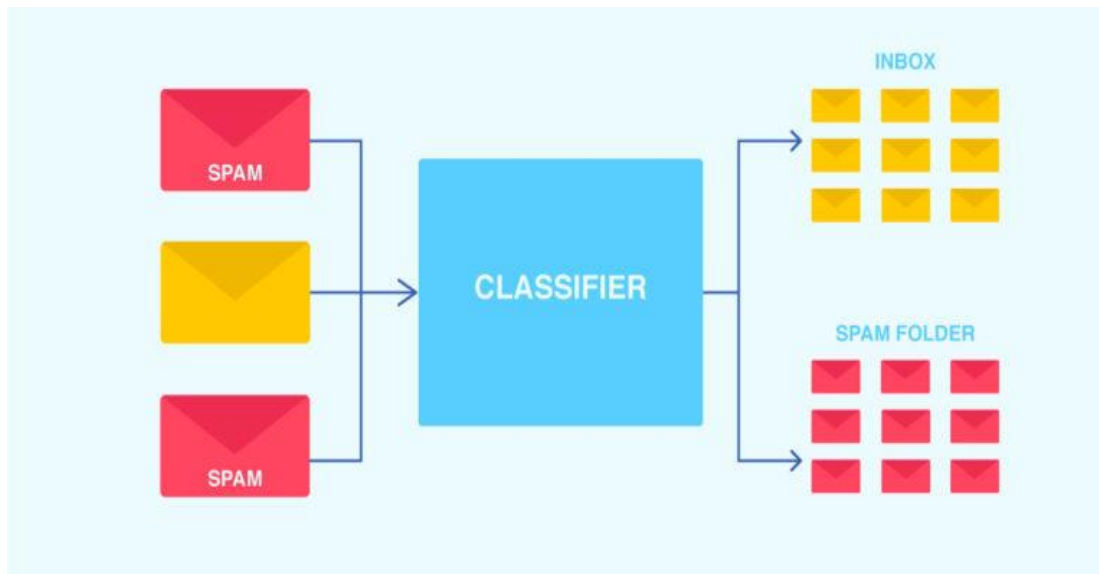
Delhi Technological University

Submitted by:

N.Sweety(2K18/IT/074)

Palak Meena(2K18/IT/080)

1. INTRODUCTION



Email has been an extremely important medium of communication for quite some time now, allowing almost instant reachability to any part of the world with internet connectivity. The fact that though more than 270 billion emails are exchanged daily, approximately 57% of these are just spam emails. Users worldwide are constantly bombarded by various types of email attacks, such as email spoofing, phishing, and variants of phishing, such as spear phishing, clone phishing, whaling, covert redirect etc. Email spoofing often involves forging the email header (The From part) so that the message appears to have been sent from a legitimate user. Email spoofing is a ploy used in spam campaigns because people tend to open an email when they think it is sent by someone they recognise. Email phishing is a form of spoofing which deceives the user with messages which appear legitimate. Malicious attackers have also attempted to hide the text behind images to defeat anti-spam programs. It is an obfuscation method by which the text of the message is stored as a JPEG or GIF image and displayed in the email. This prevents text-based spam filters from detecting and blocking spam messages.

Today, internet and social media have become the fastest and easiest ways to get information. In this age, reviews, opinions, feedbacks, messages and recommendations have become significant source of information. Thanks to advancement in technologies, we are now able to extract meaningful

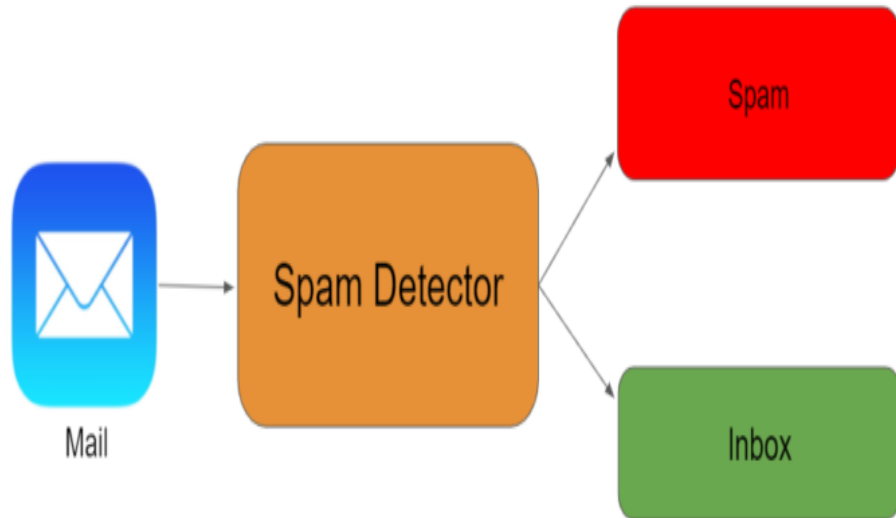
information from such data using various Natural Language Processing (NLP) techniques. NLP, a branch of Artificial Intelligence (AI), makes use of computers and human natural language to output valuable information. NLP is commonly used in text classification task such as spam detection and sentiment analysis, text generation, language translations and document classification.

Short Message Services (SMS) is far more than just a technology for a chat. SMS technology evolved out of the global system for mobile communications standard, an internationally accepted[1]. Spam is the abuse of electronic messaging systems to send unsolicited messages in bulk indiscriminately [2]. While the most widely recognized form of spam is email spam, the term is applied to similar abuses in other media and mediums. SMS Spam in the context is very similar to email spams, typically, unsolicited bulk messaging with some business interest. SMS spam is used for commercial advertising and spreading phishing links. Commercial spammers use malware to send SMS spam because sending SMS spam is illegal in most countries. Sending spam from a compromised machine reduces the risk to the spammer because it obscures the provenance of the spam. SMS can have a limited number of characters, which includes alphabets, numbers, and a few symbols. A look through the messages shows a clear pattern. Almost all of the spam messages ask the users to call a number, reply by SMS, or visit some URL. This pattern is observable by the results obtained by a simple SQL query on the spam corpus[3]. The low price and the high bandwidth of the SMS network have attracted a large amount of SMS spam

Natural Language Processing (NLP), which processes text into useful insights that can be applied to future data. In the field of artificial intelligence, NLP is one of the most complex areas of research due to the fact that text data is contextual. It needs modification to make it machine-interpretable, and requires multiple stages of processing for feature extraction.

Classification problems can be broadly split into two categories: binary classification problems, and multi-class classification problems. Binary classification means there are only two possible label classes, e.g. a patient's condition is cancerous or it isn't, or a financial transaction is fraudulent or it is not. Multi-class classification refers to cases where there are more than two label classes. An example of this is classifying the sentiment of a movie review into positive, negative, or neutral.

2. PROBLEM DESCRIPTION



The aim of this project is to identify whether the message/ Email is a spam messages or not in an email. Regular Expressions (regex) are strings used to detect patterns in data. They are often used to detect and block various forms of malware, including spam and network-based attacks.

The reason to do this is simple: by detecting unsolicited and unwanted emails, we can prevent spam messages from creeping into the user's inbox, thereby improving user experience.

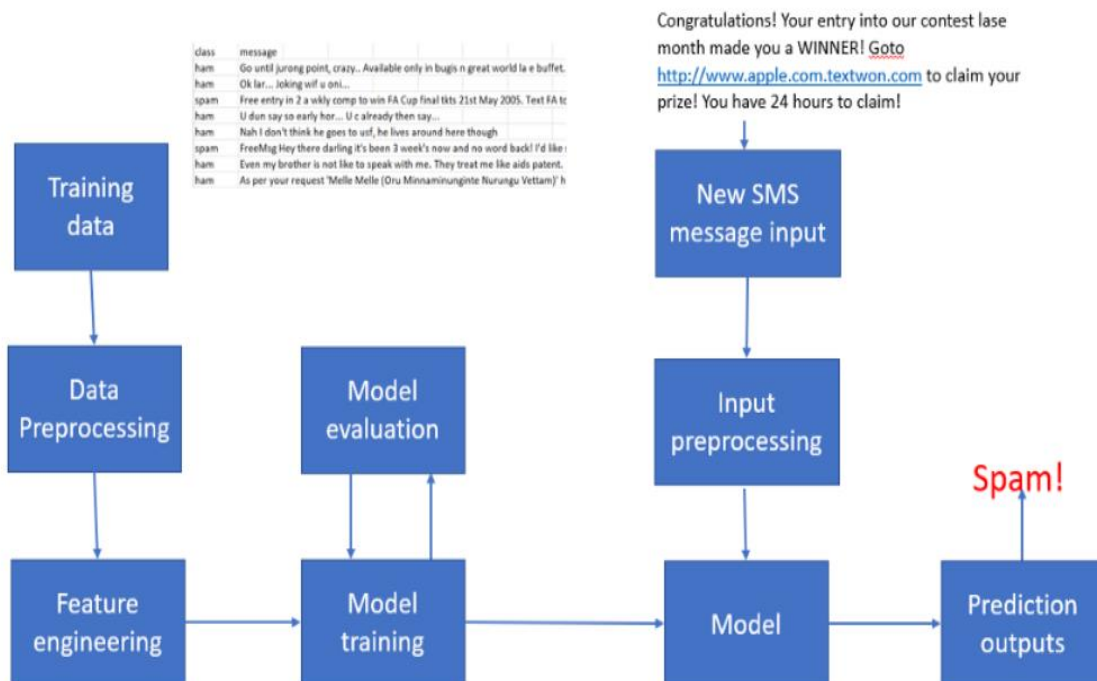
We have build a machine learning model for spam SMS message classification, then create an API for the model, using Flask, the Python micro framework for building web applications. This API allows us to utilize the predictive capabilities through HTTP requests.

3. PROPOSED APPROACH

3.1 Proposed Algorithm

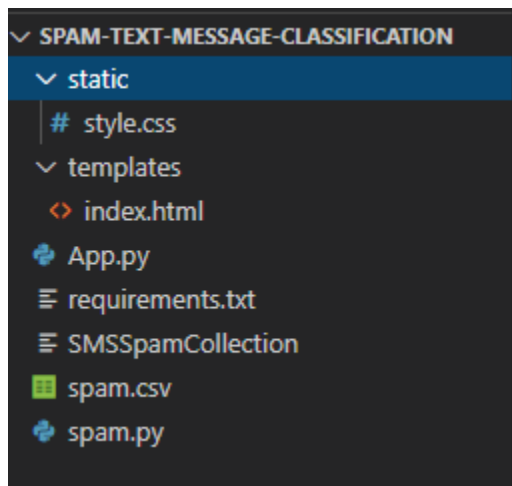
1. Import Necessary Libraries
2. Load the dataset of SMS messages
3. Preprocess the Data
4. Convert class labels to binary values, 0 = ham and 1 = spam
5. Store the SMS message data
6. Use regular expressions to replace email addresses, URLs, phone numbers, other numbers
7. Generating Features
8. Create bag-of-words
9. Split the data into training and testing datasets
10. Define models to train
11. Ensemble methods - Voting classifier
12. Make class label prediction for testing set
13. Print a confusion matrix
14. Print a bar graph of Accuracy of Models

3.2 Block Diagram



3.3 Working

The data is a collection of SMS messages tagged as spam or ham . The data set we will be using comes from the UCI Machine Learning Repository. It contains over 5000 SMS labeled messages that have been collected for mobile phone spam research and email spam detection. It can be downloaded from the following URL: <https://archive.ics.uci.edu/ml/datasets/sms+spam+collection>. First, we will use this dataset to build a prediction model that will accurately classify which texts are spam. Naive Bayes classifiers are a popular statistical technique of e-mail filtering. They typically use bag of words features to identify spam e-mail. Therefore, We'll build a simple message classifier using Naive Bayes theorem. After training the model, it is desirable to have a way to persist the model for future use without having to retrain. we will develop a web application that consists of a simple web page with a form field that lets us enter a message. After submitting the message to the web application, it will render it on a new page which gives us a result of spam or not spam. First, we create a folder for this project called **SPAM-TEXT-MESSAGE-CLASSIFICATION**. this is the directory tree inside the folder.



The sub-directory `templates` is the directory in which Flask will look for static HTML files for rendering in the web browser, in our case, we have `index.html` .

App.py

The `App.py` file contains the main code that will be executed by the Python interpreter to run the Flask web application, it included the ML code for classifying SMS messages:

- We ran our application as a single module; thus we initialized a new Flask instance with the argument `__name__` to let Flask know that it can find the HTML template folder (`templates`) in the same directory where it is located.

- Next, we used the route decorator (`@app.route('/')`) to specify the URL that should trigger the execution of the `index` function.
- Our `home` function simply rendered the `index.html` HTML file, which is located in the `templates` folder.
- Inside the `predict` function, we access the spam data set, pre-process the text, and make predictions, then store the model. We access the new message entered by the user and use our model to make a prediction for its label.
- we used the `POST` method to transport the form data to the server in the message body. Finally, by setting the `debug=True` argument inside the `App.run` method, we further activated Flask's debugger.
- Lastly, we used the `run` function to only run the application on the server when this script is directly executed by the Python interpreter, which we ensured using the `if` statement with `__name__ == '__main__'`.

index.html

`index.html` file that will render a text form where a user can enter a message:

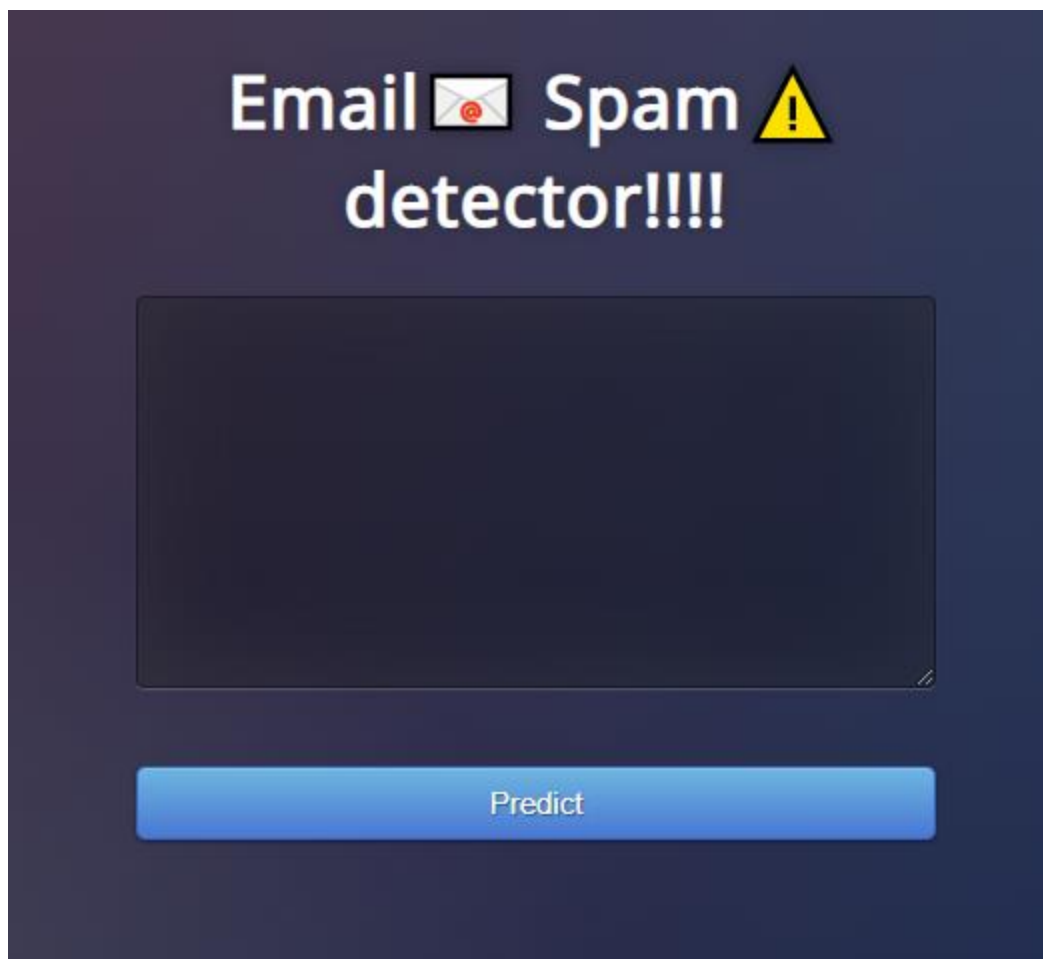
style.css

In the header section of `index.html`, we loaded `styles.css` file. CSS is to determine how the look and feel of HTML documents. `styles.css` has to be saved in a sub-directory called `static`, which is the default directory where Flask looks for static files such as CSS.

Now we can start running the API by executing the command from the Terminal: `python App.py`

```
C:\Users\nayak\Desktop\SPAM-TEXT-MESSAGE-CLASSIFICATION>python App.py
* Serving Flask app "App" (lazy loading)
* Environment: production
  WARNING: This is a development server. Do not use it in a production deployment.
  Use a production WSGI server instead.
* Debug mode: off
* Running on http://127.0.0.1:5000/ (Press CTRL+C to quit)
```

Now open a web browser and navigate <http://127.0.0.1:5000/>, we see a simple website with the content like so:



4. IMPLEMENTATION DETAILS OR PROGRESS

Table I: Implementation Environment, an example

PROGRAMMING LANGUAGE(S)	PYTHON
OPERATING SYSTEM	WINDOWS, MAC, LINUX
LIBRARY PACKAGES OR APIS USED (IF ANY)	PANDAS, NUMPY, SKLEARN, NLTK, MATPLOTLIB
INTERFACE DESIGN (GUI / WEB / OTHER)	WEB
SERVERS USED	HTTP://127.0.0.1:5000/

To ensure the necessary libraries are installed correctly and up-to-date, printing the version numbers for each library. This will also improve the reproducibility of project.

```
Python: 3.8.3 (default, Jul 2 2020, 17:30:36) [MSC v.1916 64 bit (AMD64)]
NLTK: 3.5
Scikit-learn: 0.23.2
pandas: 1.0.5
numpy: 1.18.5
```

class distribution

```
ham      4825
spam     747
Name: 0, dtype: int64
```

use regular expressions to replace email addresses, URLs, phone numbers, other numbers

- Replace email addresses with 'email'
processed = text_messages.str.replace(r'^.+@[^\.\.*\.[a-z]{2,}\$', 'emailaddress')
- Replace URLs with 'webaddress'
processed = processed.str.replace(r'^http://[a-zA-Z0-9\-\.\.]+\.[a-zA-Z]{2,3}(\S*)?\$', 'webaddress')
- Replace money symbols with 'moneysymb' (£ can be typed with ALT key + 156)
processed = processed.str.replace(r'£\\\$', 'moneysymb')
- Replace 10 digit phone numbers (formats include paranthesis, spaces, no spaces, dashes) with 'phonenumber'
processed = processed.str.replace(r'^\\((?\\d){3}\\)?\\s-?\\d{3}\\s-?\\d{4}\$', 'phonenumber')

➤ Replace numbers with 'numbr'
`processed = processed.str.replace(r"d+(\.\d+)?", 'numbr')`

➤ Remove punctuation
`processed = processed.str.replace(r'[^\w\d\s]', '')`

➤ Replace whitespace between terms with a single space
`processed = processed.str.replace(r's+', '')`

➤ Remove leading and trailing whitespace
`processed = processed.str.replace(r'^\s+|\s+?$', '')`

we can start running the API by executing the command from the Terminal: `python App.py`

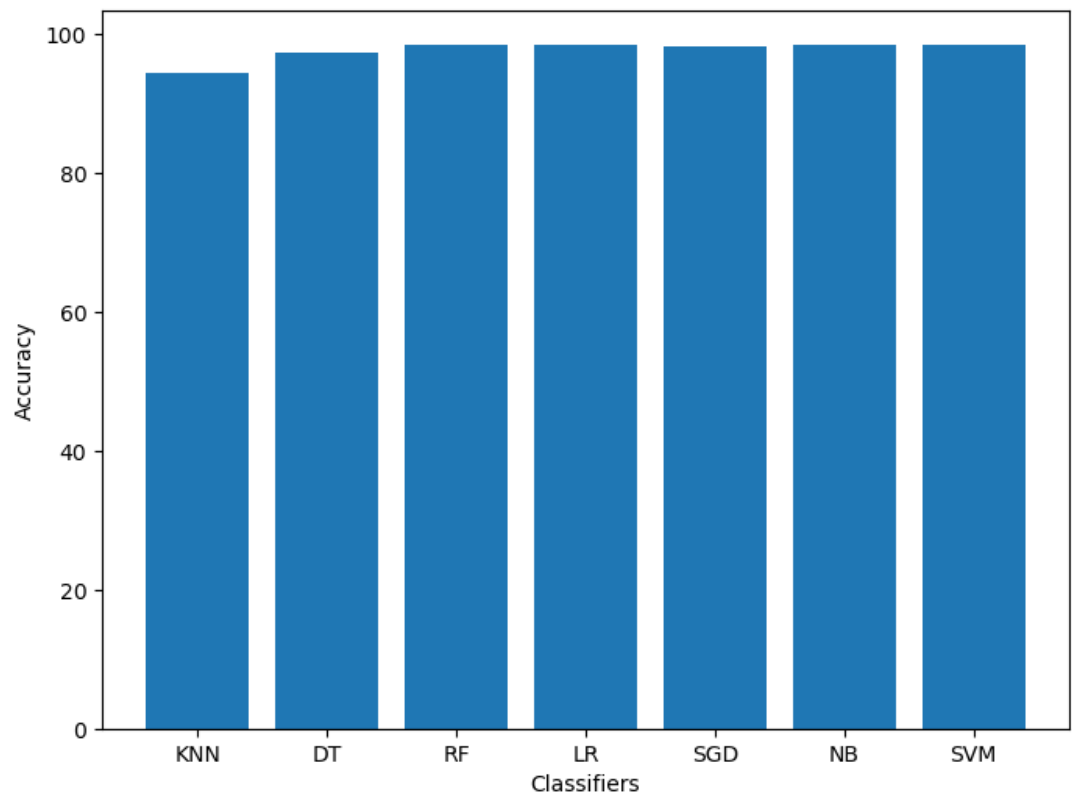
```
C:\Users\nayak\Desktop\SPAM-TEXT-MESSAGE-CLASSIFICATION>python App.py
* Serving Flask app "App" (lazy loading)
* Environment: production
  WARNING: This is a development server. Do not use it in a production deployment.
  Use a production WSGI server instead.
* Debug mode: off
* Running on http://127.0.0.1:5000/ (Press CTRL+C to quit)
```

5. RESULTS & OBSERVATIONS

The predictions(accuracy scores) using various ML algorithms are as follows:

K Nearest neighbors: Accuracy: 94.40057430007178					
Decision Tree: Accuracy: 97.34386216798278					
Random Forest: Accuracy: 98.56424982053123					
Logistic Regression: Accuracy: 98.56424982053123					
SGD classifier: Accuracy: 98.27709978463747					
Naive Bayes: Accuracy: 98.49246231155779					
SVM Linear: Accuracy: 98.49246231155779					
Ensemble Method Accuracy: 98.56424982053123					
	precision	recall	f1-score	support	
0	0.99	1.00	0.99	1207	
1	0.98	0.91	0.94	186	
accuracy			0.99	1393	
macro avg	0.98	0.95	0.97	1393	
weighted avg	0.99	0.99	0.99	1393	

Accuracy of Models

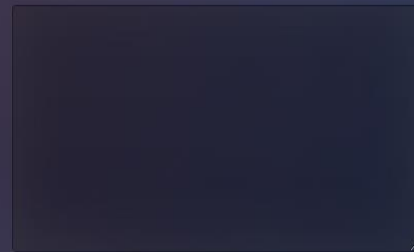


Email 📧 Spam ⚠️ detector!!!!

Free entry in 2 a wkly
comp to win FA Cup final
tkts 21st May 2005. Text
FA to 87121 to receive
entry question(std txt
rate)T&C's apply
00450010075 101

Predict

Email 📧 Spam ⚠️ detector!!!!



Predict

Looking Spam ⚠️ , Be safe

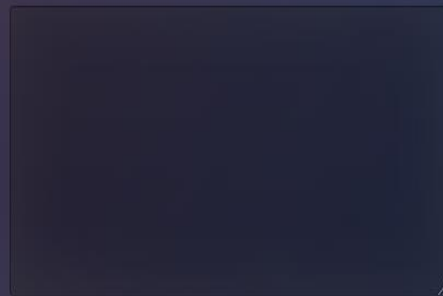
Email 📧 Spam ⚠️ detector!!!!

Even my brother is not like
to speak with me. They
treat me like aids
patent.,,,|



Predict

Email 📧 Spam ⚠️ detector!!!!



Predict

Not a Spam ❤️

6. CONCLUSION

Regular expressions are used for searching through (usually textual) data. They allow to search for pieces of text that match a certain form, instead of searching for a piece of text identical to the one we supply. We have building a machine learning model for spam SMS message classification, then create an API for the model, using Flask, the Python micro framework for building web applications. This API allows us to utilize the predictive capabilities through HTTP requests. We have identify whether the message is a spam messages or not in an email using Regular Expressions (regex). We have clean up the data by removing numbers, phone numbers, converting word to lowercase, removing punctuation, removing stopwords. As the dataset is a randomly sorted, we can directly divide the data into the training and test dataset. The algorithm uses the presence or absence of words to find the probability that a given SMS message is spam. From the results its clear that all models are showing the best results on the given data. With the help of Regular Expression we are able to build our model and get the best results. We get best results on the input which is not in the dataset.

Appendix A: Programming Code

spam.py

```
import sys

import nltk

import sklearn

import pandas

import numpy

print('Python: {}'.format(sys.version))

print('NLTK: {}'.format(nltk.__version__))

print('Scikit-learn: {}'.format(sklearn.__version__))

print('pandas: {}'.format(pandas.__version__))

print('numpy: {}'.format(numpy.__version__))

import pandas as pd

import numpy as np

import matplotlib.pyplot as plt

df=pd.read_table('SMSSpamCollection', header = None, encoding='utf-8')

print(df.info())

print(df.head())

classes = df[0]

print(classes.value_counts())

from sklearn.preprocessing import LabelEncoder

encoder = LabelEncoder()

Y = encoder.fit_transform(classes)

print(classes[:10])

print(Y[:10])

text_messages = df[1]
```

```

print(text_messages[:10])

processed = text_messages.str.replace(r'^.+@[^\.\.]*\.[a-z]{2,}$', 'emailaddr')

processed = processed.str.replace(r'http:\/\/[a-zA-Z0-9\-\.\.]+\.[a-zA-Z]{2,3}(\S*)7$', 'webaddress')

processed = processed.str.replace(r'£|$', 'moneysymb')

processed = processed.str.replace(r'^\([?\d]{3}\)?[s-]?[d]{3}[s-]?[d]{4}$', 'phonenumbr')

processed = processed.str.replace(r'd+(\.\d+)?', 'numbr')

processed = processed.str.replace(r'^[w\d\s]', ' ')

processed = processed.str.replace(r'\s+', ' ')

processed = processed.str.replace(r'^\s+|\s+?$', "")

processed = processed.str.lower()

print(processed)

from nltk.corpus import stopwords

stop_words = set(stopwords.words('english'))

processed = processed.apply(lambda x: ''.join(term for term in x.split() if term not in stop_words))

ps = nltk.PorterStemmer()

processed = processed.apply(lambda x: ''.join(ps.stem(term) for term in x.split()))

print(processed)

from nltk.tokenize import word_tokenize

all_words = []

for message in processed:

    words = word_tokenize(message)

    for w in words:

        all_words.append(w)

all_words = nltk.FreqDist(all_words)

print('number of words: {}'.format(len(all_words)))

print('Most common words: {}'.format(all_words.most_common(15)))

word_features = list(all_words.keys())[:1500]

def find_features(message):

```

```

        words = word_tokenize(message)

        features = { }

        for word in word_features:

            features[word] = (word in words)

    return features

features = find_features(processed[0])

for key, value in features.items():

    if value == True:

        print(key)

messages = list(zip(processed, Y))

seed = 1

np.random.seed = seed

np.random.shuffle(messages)

featuresets = [(find_features(text), label) for (text, label) in messages]

from sklearn import model_selection

training, testing = model_selection.train_test_split(featuresets, test_size = 0.25, random_state = seed)

print('training: {}'.format(len(training)))

print('testing: {}'.format(len(testing)))

from sklearn.neighbors import KNeighborsClassifier

from sklearn.tree import DecisionTreeClassifier

from sklearn.ensemble import RandomForestClassifier

from sklearn.linear_model import LogisticRegression, SGDClassifier

from sklearn.naive_bayes import MultinomialNB

from sklearn.svm import SVC

from sklearn.metrics import classification_report, accuracy_score, confusion_matrix

```



```
names = ['K Nearest neighbors', 'Decision Tree', 'Random Forest', 'Logistic Regression', 'SGD classifier', 'Naive Bayes', 'SVM Linear']
```

```
classifiers = [  
    KNeighborsClassifier(),  
    DecisionTreeClassifier(),  
    RandomForestClassifier(),  
    LogisticRegression(),  
    SGDClassifier(max_iter = 100),  
    MultinomialNB(),  
    SVC(kernel = 'linear')  
]
```

```
models = list(zip(names, classifiers))
```

```
from nltk.classify.scikitlearn import SklearnClassifier
```

```
for name, model in models:
```

```
    nltk_model = SklearnClassifier(model)  
    nltk_model.train(training)  
    accuracy = nltk.classify.accuracy(nltk_model, testing) * 100  
    print('{ }: Accuracy: { }'.format(name, accuracy))
```

```
from sklearn.ensemble import VotingClassifier
```

```
names = ['K Nearest neighbors', 'Decision Tree', 'Random Forest', 'Logistic Regression', 'SGD classifier', 'Naive Bayes', 'SVM Linear']
```

```
classifiers = [  
    KNeighborsClassifier(),  
    DecisionTreeClassifier(),  
    RandomForestClassifier(),  
    LogisticRegression(),  
    SGDClassifier(max_iter = 100),  
]
```

```

MultinomialNB(),

SVC(kernel = 'linear')

]

models = list(zip(names, classifiers))

nltk_ensemble = SklearnClassifier(VotingClassifier(estimators = models, voting = 'hard', n_jobs = -1))

nltk_ensemble.train(training)

accuracy = nltk.classify.accuracy(nltk_ensemble, testing) * 100

print('Ensemble Method Accuracy: {}'.format(accuracy))

txt_features, labels = zip(*testing)

prediction = nltk_ensemble.classify_many(txt_features)

print(classification_report(labels, prediction))

pd.DataFrame(

    confusion_matrix(labels, prediction),

    index = [['prediction', 'predicted'], ['ham', 'spam']],

    columns = [['predicted', 'prediction'], ['ham', 'spam']])


names = ['KNN', 'DT', 'RF', 'LR', 'SGD', 'NB', 'SVM']

acc =
[94.40057430007178, 97.34386216798278, 98.56424982053123, 98.56424982053123, 98.27709978463747, 98.49246
231155779, 98.49246231155779]

plt.figure(figsize=(8,6))

plt.subplot()

plt.bar(names, acc, width=0.8)

plt.xlabel('Classifiers')

plt.ylabel('Accuracy')

plt.suptitle('Accuracy of Models')

plt.show()

```

App.py

```
from flask import Flask, render_template, request

import pandas as pd

from sklearn.feature_extraction.text import CountVectorizer

from sklearn.naive_bayes import MultinomialNB

from sklearn.model_selection import train_test_split


app = Flask(__name__)


@app.route('/')

def home():

    return render_template('index.html')


@app.route('/predict', methods=['POST'])

def predict():

    df = pd.read_csv("spam.csv", encoding="latin-1")

    df.drop(['Unnamed: 2', 'Unnamed: 3', 'Unnamed: 4'], axis=1, inplace=True)

    # Features and Labels

    df['label'] = df['class'].map({'ham': 0, 'spam': 1})

    X = df['message']

    y = df['label']

    # Extract Feature With CountVectorizer

    cv = CountVectorizer()

    X = cv.fit_transform(X) # Fit the Data

    X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.33, random_state=42)

    # Naive Bayes Classifier

    clf = MultinomialNB()
```

```
clf.fit(X_train, y_train)
```

```
clf.score(X_test, y_test)
```

```
if request.method == 'POST':
```

```
    message = request.form['message']
```

```
    data = [message]
```

```
    vect = cv.transform(data).toarray()
```

```
    my_prediction = clf.predict(vect)
```

```
return render_template('index.html', prediction=my_prediction)
```

```
if __name__ == '__main__':
```

```
    app.run()
```

index.html

```
<!DOCTYPE html>

<html >

<head>

  <meta charset="UTF-8">

  <title>Spam Detection System</title>

  <link href='https://fonts.googleapis.com/css?family=Pacifico' rel='stylesheet' type='text/css'>

  <link href='https://fonts.googleapis.com/css?family=Arimo' rel='stylesheet' type='text/css'>

  <link href='https://fonts.googleapis.com/css?family=Hind:300' rel='stylesheet' type='text/css'>

  <link href='https://fonts.googleapis.com/css?family=Open+Sans+Condensed:300' rel='stylesheet' type='text/css'>

  <link rel="stylesheet" href="{{ url_for('static', filename='style.css') }}">

</head>

<body>

<div class="login">

  <h1> Email 📧 Spam ⚠️ detector!!!!</h1>

  <form action="{{ url_for('predict') }}" method="POST">

    <textarea name="message" rows="6" cols="50" required="required"></textarea>

    <br> </br>


    <button type="submit" class="btn btn-primary btn-block btn-large">Predict</button>

  <div class="results">

    {% if prediction == 1% }

    <h2 style="color:red;">Looking Spam ⚠️, Be safe</h2>
```

```
{% elif prediction == 0% }
```

```
<h2 style="color:green;"><b>Not a Spam 
```

```
{% endif % }
```

```
</div>
```

```
</form>
```

```
</div>
```

style.css

```
@import url(https://fonts.googleapis.com/css?family=Open+Sans);
```

```
.btn { display: inline-block; *display: inline; *zoom: 1; padding: 4px 10px 4px; margin-bottom: 0; font-size: 13px; line-height: 18px; color: #333333; text-align: center; text-shadow: 0 1px 1px rgba(255, 255, 255, 0.75); vertical-align: middle; background-color: #f5f5f5; background-image: -moz-linear-gradient(top, #ffffff, #e6e6e6); background-image: -ms-linear-gradient(top, #ffffff, #e6e6e6); background-image: -webkit-gradient(linear, 0 0, 0 100%, from(#ffffff), to(#e6e6e6)); background-image: -webkit-linear-gradient(top, #ffffff, #e6e6e6); background-image: -o-linear-gradient(top, #ffffff, #e6e6e6); background-image: linear-gradient(top, #ffffff, #e6e6e6); background-repeat: repeat-x; filter: progid:dximagetransform.microsoft.gradient(startColorstr=#ffffff, endColorstr=#e6e6e6, GradientType=0); border-color: #e6e6e6 #e6e6e6 #e6e6e6; border-color: rgba(0, 0, 0, 0.1) rgba(0, 0, 0, 0.1) rgba(0, 0, 0, 0.25); border: 1px solid #e6e6e6; -webkit-border-radius: 4px; -moz-border-radius: 4px; border-radius: 4px; -webkit-box-shadow: inset 0 1px 0 rgba(255, 255, 255, 0.2), 0 1px 2px rgba(0, 0, 0, 0.05); -moz-box-shadow: inset 0 1px 0 rgba(255, 255, 255, 0.2), 0 1px 2px rgba(0, 0, 0, 0.05); box-shadow: inset 0 1px 0 rgba(255, 255, 255, 0.2), 0 1px 2px rgba(0, 0, 0, 0.05); cursor: pointer; *margin-left: .3em; }
```

```
.btn:hover, .btn:active, .btn.active, .btn.disabled, .btn[disabled] { background-color: #e6e6e6; }
```

```
.btn-large { padding: 9px 14px; font-size: 15px; line-height: normal; -webkit-border-radius: 5px; -moz-border-radius: 5px; border-radius: 5px; }
```

```
.btn:hover { color: #333333; text-decoration: none; background-color: #e6e6e6; background-position: 0 -15px; -webkit-transition: background-position 0.1s linear; -moz-transition: background-position 0.1s linear; -ms-transition: background-position 0.1s linear; -o-transition: background-position 0.1s linear; transition: background-position 0.1s linear; }
```

```
.btn-primary, .btn-primary:hover { text-shadow: 0 -1px 0 rgba(0, 0, 0, 0.25); color: #ffffff; }
```

```
.btn-primary.active { color: rgba(255, 255, 255, 0.75); }
```

```
.btn-primary { background-color: #4a77d4; background-image: -moz-linear-gradient(top, #6eb6de, #4a77d4); background-image: -ms-linear-gradient(top, #6eb6de, #4a77d4); background-image: -webkit-gradient(linear, 0 0, 0 100%, from(#6eb6de), to(#4a77d4)); background-image: -webkit-linear-gradient(top, #6eb6de, #4a77d4); background-image: -o-linear-gradient(top, #6eb6de, #4a77d4); background-image: linear-gradient(top, #6eb6de, #4a77d4); background-repeat: repeat-x; filter: progid:dximagetransform.microsoft.gradient(startColorstr=#6eb6de, endColorstr=#4a77d4, GradientType=0); border: 1px solid #3762bc; text-shadow: 1px 1px 1px rgba(0, 0, 0, 0.4); box-shadow: inset 0 1px 0 rgba(255, 255, 255, 0.2), 0 1px 2px rgba(0, 0, 0, 0.5); }
```

```
.btn-primary:hover, .btn-primary:active, .btn-primary.active, .btn-primary.disabled, .btn-primary[disabled] { filter: none; background-color: #4a77d4; }
```

```
.btn-block { width: 100%; display: block; }
```

```
* { -webkit-box-sizing: border-box; -moz-box-sizing: border-box; -ms-box-sizing: border-box; -o-box-sizing: border-box; box-sizing: border-box; }
```

```
html { width: 100%; height:100%; overflow:hidden; }
```

```
body {
```

```
    width: 100%;
```

```
    height:100%;
```

```
    font-family: 'Open Sans', sans-serif;
```

```
    background: #092756;
```

```
    color: #fff;
```

```
    font-size: 18px;
```

```
    text-align:center;
```

```
    letter-spacing:1.2px;
```

```
    background: -moz-radial-gradient(0% 100%, ellipse cover, rgba(104,128,138,.4) 10%,rgba(138,114,76,0) 40%),-moz-linear-gradient(top,  rgba(57,173,219,.25) 0%, rgba(42,60,87,.4) 100%), -moz-linear-gradient(-45deg, #670d10 0%, #092756 100%);
```

```
    background: -webkit-radial-gradient(0% 100%, ellipse cover, rgba(104,128,138,.4) 10%,rgba(138,114,76,0) 40%), -webkit-linear-gradient(top,  rgba(57,173,219,.25) 0%,rgba(42,60,87,.4) 100%), -webkit-linear-gradient(-45deg, #670d10 0%,#092756 100%);
```

```
    background: -o-radial-gradient(0% 100%, ellipse cover, rgba(104,128,138,.4) 10%,rgba(138,114,76,0) 40%), -o-linear-gradient(top,  rgba(57,173,219,.25) 0%,rgba(42,60,87,.4) 100%), -o-linear-gradient(-45deg, #670d10 0%,#092756 100%);
```

```
    background: -ms-radial-gradient(0% 100%, ellipse cover, rgba(104,128,138,.4) 10%,rgba(138,114,76,0) 40%), -ms-linear-gradient(top,  rgba(57,173,219,.25) 0%,rgba(42,60,87,.4) 100%), -ms-linear-gradient(-45deg, #670d10 0%,#092756 100%);
```

```
    background: -webkit-radial-gradient(0% 100%, ellipse cover, rgba(104,128,138,.4) 10%,rgba(138,114,76,0) 40%), linear-gradient(to bottom,  rgba(57,173,219,.25) 0%,rgba(42,60,87,.4) 100%), linear-gradient(135deg, #670d10 0%,#092756 100%);
```

```
    filter: progid:DXImageTransform.Microsoft.gradient( startColorstr='#3E1D6D', endColorstr='#092756',GradientType=1 );
```

```
}
```

```
.login {
```

```
    position: absolute;
```

```
    top: 40%;
```



```
    left: 50%;  
  
    margin: -150px 0 0 -150px;  
  
    width: 400px;  
  
    height: 400px;  
  
}
```

```
.login h1 { color: #fff; text-shadow: 0 0 10px rgba(0,0,0,0.3); letter-spacing: 1px; text-align: center; }
```

```
textarea {  
  
    width: 100%;  
  
    margin-bottom: 10px;  
  
    background: rgba(0,0,0,0.3);  
  
    border: none;  
  
    outline: none;  
  
    padding: 10px;  
  
    font-size: 25px;  
  
    color: #fff;  
  
    text-shadow: 1px 1px 1px rgba(0,0,0,0.3);  
  
    border: 1px solid rgba(0,0,0,0.3);  
  
    border-radius: 4px;  
  
    box-shadow: inset 0 -5px 45px rgba(100,100,100,0.2), 0 1px 1px rgba(255,255,255,0.2);  
  
    -webkit-transition: box-shadow .5s ease;  
  
    -moz-transition: box-shadow .5s ease;  
  
    -o-transition: box-shadow .5s ease;  
  
    -ms-transition: box-shadow .5s ease;  
  
    transition: box-shadow .5s ease;  
  
}  
  
input:focus { box-shadow: inset 0 -5px 45px rgba(100,100,100,0.4), 0 1px 1px rgba(255,255,255,0.2); }
```