SPAM DETECTION

WITH THE USE OF MACHINE LEARNING ALGORITHMS

Jakub Sykulski, s1071754
Bezayite Amenu, s1076971

Abstract

In the era of the internet, we are able to easily search for information, communicate with one another or even shop in any place on the earth with just a few strokes on the keyboard of our computer. Unfortunately, the same thing that makes our life easier also introduces new ways of information gathering on people. The most common way to do that these days is by using 'spam' messages, email, social media accounts, etc. When 'spam' is shown to us, sometimes it looks really convincing as if it was a real product, message or email. Therefore we click on it to find out more without realizing how much we reveal about ourselves.

'Spam' is a huge problem in the current days of the internet. That's why we decided to try to create a program that would be able to scan through many messages, emails, etc. and detect which ones most likely fit the 'spam' profile. We make use of existing machine learning algorithms which include MLP, Support Vector Classifier, and Decision Trees which are applied on the chosen data sets. We found that MLP performs best compared to the other algorithms with a score of 92.16%.

Introduction

Over the years, the use of the internet has grown significantly for different purposes including communication through emails, sms messages, social media and so on. This increased usage of the internet for communication poses a serious issue of increased Spam messages or emails. These spam messages vary from just random pestering messages with the intention of spreading harmful/false information, they can also be in the form of emails with the aim of performing a phishing attack where the recipient is tricked into clicking on a malicious link. These spam emails are usually sent in bulk to a number of recipients who may not be aware of their intent. Spam emails may also cause security issues to the computer system. Several methods have been introduced over the years to try and tackle this issue. However, spammers have also gotten smarter and found ways to bypass these methods. In this project, we attempt to recognize the patterns with the use of different Machine Learning algorithms to aid in the classifications of the emails/messages as SPAM or HAM where ‘SPAM’ is used to classify the messages that are considered spam, and ‘HAM’ is used to classify messages that are not considered spam. We explore algorithms such as the Multilayer Perceptron, Support Vector Machine, and Decision Trees. We make use of samples of pre-classified training datasets and each of the algorithms will be used in order to classify datasets used in the future into ‘spam’ or ‘ham’ categories.

Existing Systems

- Harisinghaney et al. (2014), use the K-Nearest-Neighbor Algorithm, Naive Bayes and Reverse DBSCAN algorithm for spam detection in emails using the image and textual datasets.

- S.Ananthi, uses the K-Nearest-Neighbor Algorithm as the foundation of personalized spam filters, to produce her research titled, “Spam Filtering using K -NN”.

- Saumya Goyal et al. propose a classification model using the K-Nearest-Neighbor Algorithm and Decision Tree algorithm to classify spam messages in Social Networks

Data Set

The dataset that was used in our project was obtained from “https://www.kaggle.com/”. The “.csv” files that we are using are composed of spam data for emails and sms. All the data sets used in our project have the same structure, meaning two columns. Usually, the first column is a label in the format of “spam” or “ham”, where the word “spam” indicates that the given message is classified as a spam message in the data set. The second column is the message itself.
For our project we used the following kaggle.com repositories:

- “Spam Email | Kaggle”
- “https://www.kaggle.com/datasets/adepvenugopal/sms-spam”
- “SMS Spam Collection (Text Classification) | Kaggle”

As we can here we used both types of data sets for ‘sms’ and ‘email’, that is because after examining those datasets we discovered that around 90% of the keywords are the same. In order to find that result, we created a simple python program that iterated through those data sets and, as mentioned above, discovered that around 90% of all keywords indicating a spam message or email are the same.

Data Preprocessing

At the beginning of the project, we load all of our data sets into arrays. They have the following format ***[[string1, string2, …], [string1, string2, …], …].***
In order to train our classifiers we had to process our spam data in some way. After examining a couple of possible routes we decided to do as follows: One data set, here a “spam.csv” file, is used to create a dictionary of ‘spam’ words.

```
import numpy as np
from toolbox.load_assets import load_spam
from toolbox.prepare_data import prepare
from toolbox.conversion import conversion
from toolbox.reshape import reshape
#---------------------------------------------------- file loading
file = "./data/spam.csv" # 'spam' used to create a dictionary
initial_train_data = "./data/train.csv" # 'spam' used for training and testing the initial model
#----------------------------------------------------::::

#----------------------------------------------------creation of 'spam' words look up dictionary
spam_dic = load_spam(file)
#----------------------------------------------------::::

#----------------------------------------------------loading and extracting data for the initial training of the model
data = prepare(initial_train_data,'spam_text','label')
text =  data[0]
labels = data[1]
#----------------------------------------------------::::

#----------------------------------------------------conversion of data arrays to np.array
labels_np = np.asarray(labels)
#----------------------------------------------------::::

#----------------------------------------------------creation of the training data, format [[0,1,0,1...],....]
train_data_main = conversion(spam_dic, text)
reshape(train_data_main)
train_data_main_np = np.asarray(train_data_main)
#----------------------------------------------------::::

#----------------------------------------------------the number of features used to train the initial model
NR_FEATURES = len(train_data_main[0])
```

```
jakeISAC
def load_spam(file_name):
    spam_file = pd.read_csv(file_name, encoding='latin-1', engine='python')
    temp = []
    spam_array = []


    for index, row in spam_file.iterrows():
        if row['v1'] == 'spam':
            temp.append(row['v2'])

    for i in range(len(temp)):
        spam_array.append(temp[i].split(" "))


    return spam_array
```

Later, this array is used as a look-up table for another data set which is used for the creation of the array used in the models as training data. This is done with the help of the function inside the “conversion.py” file.

```
jakelSAC
def conversion(spam_array, data_array):

    final_array = []

    for i in range(len(data_array)):
        temp = []
        for j in range(len(data_array[i])):
            if any(data_array[i][j] in sublist for sublist in spam_array):
                temp.append(1)

        final_array.append(temp)

    return final_array

#%%
```

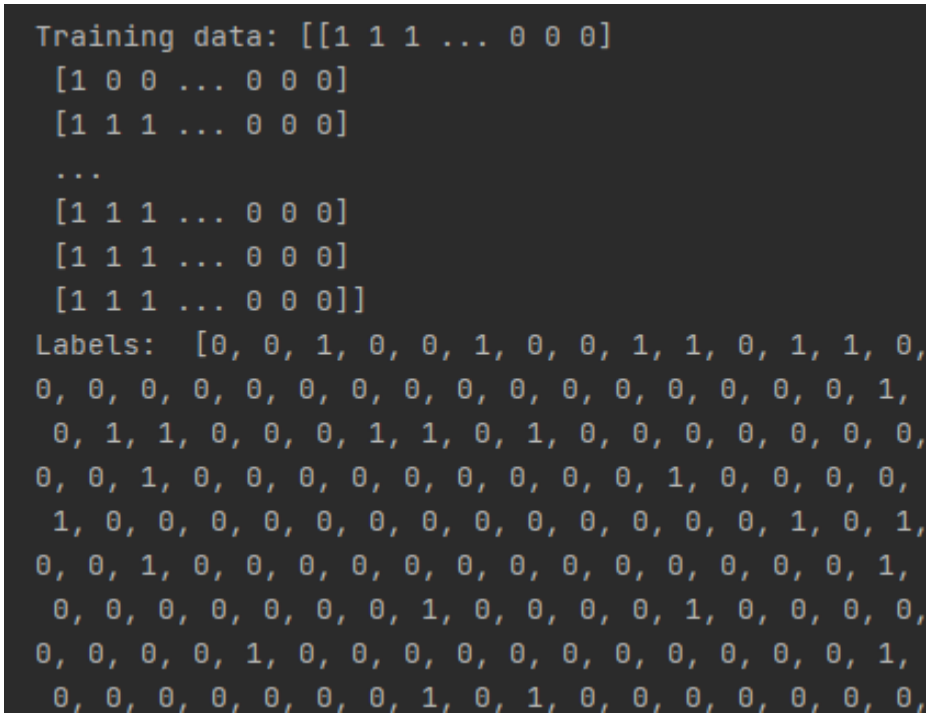
The function takes two parameters, a “spam_array” and a “data_array”. Then the function iterates through the “data_array” and checks if the word at a current position exists anywhere inside the “spam_array”; if it does, the function puts “1” in the “temp” array. In the end, the function returns the “final array” that will be in the format of:
[[1,1,1,...],[1,1],....]
But in order to make the data valid for our models they have to be reshaped so the dimensions are correct. This is done with the help of the “reshape” function inside the “reshape.py” file.

```
jakelSAC
def reshape(a):
    temp = []
    for i in a:
        temp.append(len(i))
    curr_max = max(temp)

    for r in a:
        for m in range(abs(len(r) - curr_max)):
            r.append(0)
```

The function works in the following way. It takes an array and iterates through it and puts all the lengths of the arrays inside into the temp array, then the max length is found. Then with the second for-loop, the array is modified such that all internal arrays have the same length, this is done by appending zeros to those arrays.

After all of those operations, the final result has the following format:



Machine Learning Models Used:

1. Multilayer Perceptron
   Multilayer Perceptron is a feed-forward facing artificial neural network that maps input data sets to a set of appropriate outputs. MLP is made up of several layers and each layer is connected to the other. Usually, they consist of at least one input layer, which will then send weighted inputs to a series of hidden layers, and an output layer at the end.

   In this project, we made use of K-fold cross-validation on the MLP. It partitions the dataset into equal K-folds where each fold is further split into train and test data. This ensures higher accuracy of the model because it will be tested on smaller sections of the equally distributed data. We chose to partition the dataset into 10 splits.

   In order to determine the optimal number of hidden layers, we first iterate through the 10 splits that were created and for every split, we check 25 different values for the ‘hidden layer sizes’

2. Support Vector Machine
   A support Vector Machine is an algorithm that performs supervised learning for the classification and regression of data. The foundation of SVMs is based on finding a hyperplane that divides any dataset into two classes where each class is found on one side. In spam detection, the idea is to distinguish the messages as spam or ham with the use of an optimal hyperplane

   In this project, we tested several kernel parameters that select the type of hyperplane to separate the data, by calculating the accuracy measure for each. The kernel parameters that were considered are, ‘linear’, ‘poly’, ‘rbf’, and ‘sigmoid’.

3. Decision Tree
   A Decision Tree is also another algorithm that performs supervised learning for the classification of data. It is a tree-like structure where the internal nodes represent the features of a dataset, each of the leaf nodes represents the outcome(class labels) and the branches represent the decision rules. The main objective of using a decision tree is to create a model that, based on various input variables, predicts the value of a target variable. The decision tree can be used to categorize the messages as spam or not based on the data that has been trained

Metrics Used for Evaluation

- Accuracy: is an evaluation metric which measures the total number of correct predictions that are made by a model in regards to the total number of predictions made.

- Confusion Matrix: a table used to describe the performance of a classification model on test data for which the true values are known. It consists of four different combinations of the predicted and actual values.

Results

After evaluating the accuracy of each of the models after data training and testing. The datasets were collected from the "https://www.kaggle.com/" resource. 5574 messages were used to train the different Machine learning models and 5559 messages were used for testing for the purpose of computing the accuracy score.

Our findings suggest that using the MLP model performs best in detecting spam messages with a high accuracy of 92.16%.

Possible improvements

In our current approach, we transform the messages into data_arrays by looking up words and checking if that specific word has been used in the known 'spam' messages.

The main improvement that can be done to the model is the modification of the "conversion" modulo, the way that we decided if the word is a known, often-used, spam keyword.

We have two ideas that could possibly improve our project:

1. In order to more accurately decide if a word inside a message should be marked as spam or ham, along with our 'spam_dic' we could also calculate how often this word appears throughout our data along with how often it is marked as spam keyword.
2. Some spam messages might be purposely written to resemble ham, in order to avoid spam detectors, and only include a direct link/URL or an embedded link

leading to the website set up by the person responsible for the spam message. To make our models more accurate, if such a link is detected it should be checked against a database of known malicious links and if the lookup turns out to be true, then the message should be immediately marked as spam.

GitHub + project cloud download

Link to the project GitHub repository:

- https://github.com/jacob-beza-radboud/spamDetectionProject.git

Link to download the project from the cloud:

- https://drive.proton.me/urls/RNQFQCCS94#8SLS9T1OMUU5