Email Spam Classification Project Report

Table of Contents

Email Spam classification Project Report:

The project titled “Email Spam classification” . You will get to know Business understanding, Data Understanding (Data Description and Exploration), Data Preparation, Modelling, and Evaluation steps. Project is implemented using Python class object-based style. Email spam detection is done using machine learning algorithms Naive Bayes and SVM (Support vector machines). Further, it shows the complete program flow for Python-based email spam classifier implementation such as Data Retrieval Flow, Data Visualization Flow, Data Preparation Flow, Modelling, and Evaluation Flow. Also, including the section regarding Data ethics.

Business Understanding

Most of us consider spam emails as one which is annoying and repetitively used for purpose of advertisement and brand promotion. We keep on blocking such email-ids but it is of no use as spam emails are still prevalent. Some major categories of spam emails that are causing great risk to security, such as fraudulent e-mails, identify theft, hacking, viruses, and malware. In order to deal with spam emails, we need to build a robust real-time email spam classifier that can efficiently and correctly flag the incoming mail spam, if it is a spam message or looks like a spam message. The latter will further help to build an Anti-Spam Filter.

Google and other email services are providing utility for flagging email spam but are still in the infancy stage and need regular feedback from the end-

user. Also, popular email services such as Gmail, Yandex, yahoo mail, etc provide basic services as free to the end-user and that of course comes with EULA. There is a great scope in building email spam classifiers, as the private companies run their own email servers and want them to be more secure because of the confidential data, in such cases email spam classifier solutions can be provided to such companies.

Data understanding

The email spam classifier focuses on either header, subject, and content of the email. In this project, we are focusing mainly on the subject and content of the email.

To download the email spam classification dataset files and complete code and visit the link **email spam detection and classification project GitHub repository.**

Data Description

The dataset contains two columns. The total corpus of 5728 documents. The descriptive feature consists of text. The target feature consists of two classes ham and spam, the column name is spam. The classes are labelled for each document in the data set and represent our target feature with a binary string-type alphabet of {ham; spam}. Classes are further mapped to integer 0 (ham) and 1 (spam).

	target	text
0	0	Go until jurong point, crazy.. Available only ...
1	0	Ok lar... Joking wif u oni...
2	1	Free entry in 2 a wkly comp to win FA Cup fina...
3	0	U dun say so early hor... U c already then say...
4	0	Nah I don't think he goes to usf, he lives aro...

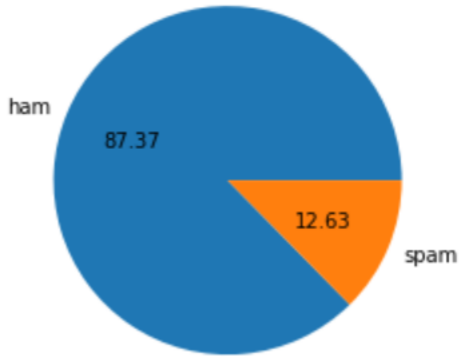
Data exploration

Spam email percentage in the dataset = 12.63 %

Ham email percentage in the dataset = 87.37 %

The bar graph given below depicts the percentage of Ham and Spam emails in the given dataset. The blue bar represents the count of ham emails and the red bar shows the count of spam emails in the dataset.

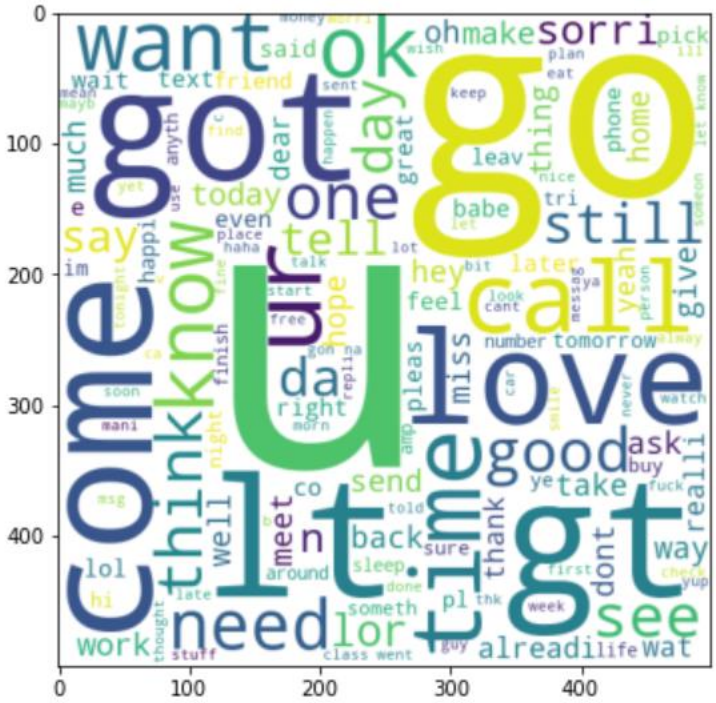
```
import matplotlib.pyplot as plt
plt.pie(df['target'].value_counts(), labels=['ham','spam'],autopct="%0.2f")
plt.show()
```



Word cloud for ham emails

```
plt.figure(figsize=(15,6))
plt.imshow(ham_wc)
```

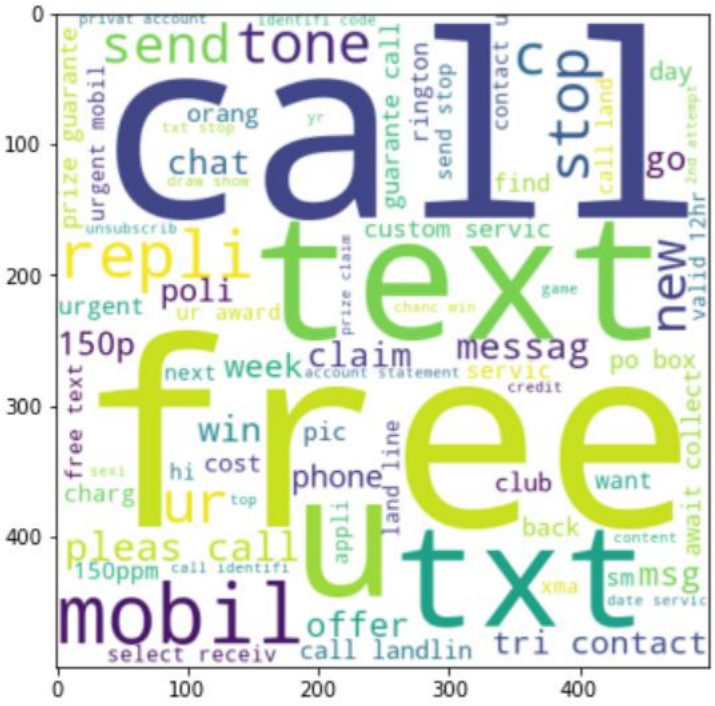
```
<matplotlib.image.AxesImage at 0x1c294e44130>
```



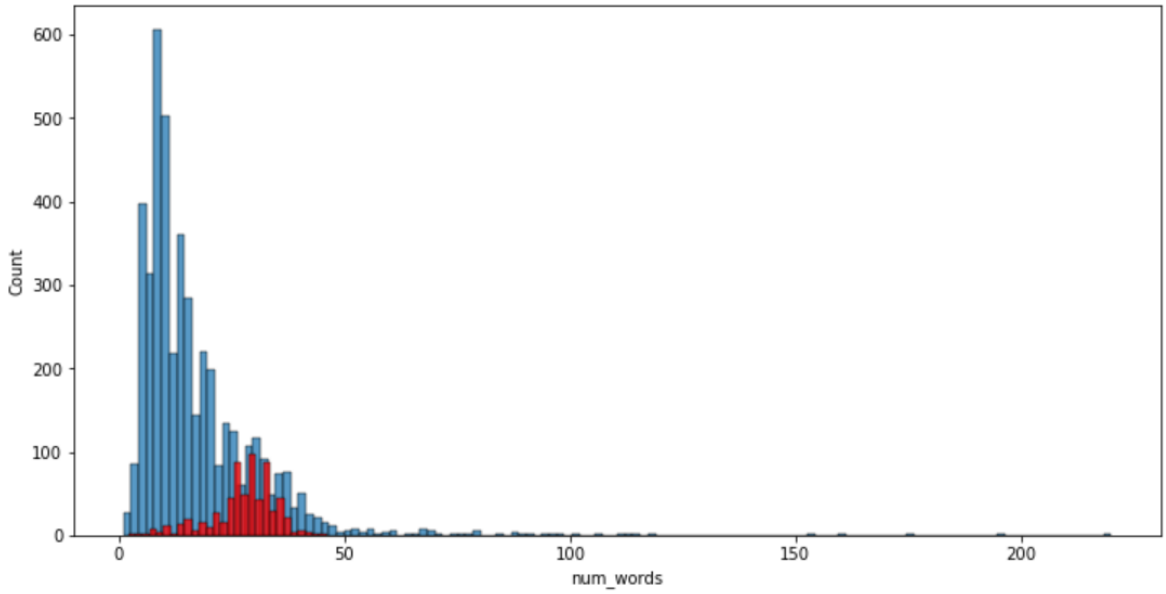
Word cloud for spam emails

```
plt.figure(figsize=(15,6))
plt.imshow(spam_wc)
```

```
<matplotlib.image.AxesImage at 0x1c2949531c0>
```



Distribution of the number of words: From the below figure we can see that, there is a spike in spam emails with a smaller number of words, even when our dataset includes 12.63 percent of spam emails out of total emails.



Data preparation

The following steps we used for data preparation.

- Identifying Missing values.
- Converting all text to lower case.
- Performing tokenization.
- Removing Stop words.
- Labelling classes: ham/spam: {0;1}
- Splitting Train and Test Data: 80% and 20%.

Modeling

Feature representation: Using word embedding technique CountVectorizer.

Models used: **Naive Bayes and SVM**. Email spam classification done using traditional machine learning techniques comprise Baive Bayes and SVM (support vector machines), due to not having sufficient hardware resources, takes less time to train. Also, not opting for neural algorithms due to less data and computing resources.

Reason for choosing SVM and Naïve Bayes: Both are good at handling large number of features; in the case of text classification each word is a feature and we have thousands of words based on the vocabulary of the corpus. SVM works best with high dimensional data, a vocabulary with 1000 words means each text in the corpus will be represented with a vector of 1000 dimension.

When we have a sufficient number of features, both SVM and Naïve Bayes can work with less data as well.

Naïve Bayes does not suffer from curse-of-dimensionality because it treats all features as independent of one another. Also, one of the benefits of features being independent is: For example, most spam emails contain words such as money and investment, etc, but it is not necessary that all the mails containing both words money and investment are considered to be spam.

Feature representation: Word embeddings can be broadly classified into two categories: Frequency and prediction-based. I have chosen a count vector that shows the count of occurrence of a feature in the given

document, thus it is a matrix of document vs vocabulary (containing all the features as a column).

In our case, the size of the Count vector matrix is 5728 x 20114, where 5728 represents the number of documents in the corpus and 20114 represents the number of features in the vocabulary.

Splitting Training and Testing Data: Splitting the data into training and test datasets, where training data contains 80 percent and test data contains 20 percent.

Applying model SVM and Naïve Bayes: I trained the model for both SVM and Naive without tuning hyperparameters as I got results with default parameter settings.

Evaluation

Naïve Bayes Result on Test dataset:

```
from sklearn.naive_bayes import GaussianNB,MultinomialNB,BernoulliNB
from sklearn.metrics import accuracy_score,confusion_matrix,precision_score
```

```
gnb = GaussianNB()
mnb = MultinomialNB()
bnb = BernoulliNB()
```

```
gnb.fit(X_train,y_train)
y_pred1 = gnb.predict(X_test)
print(accuracy_score(y_test,y_pred1))
print(confusion_matrix(y_test,y_pred1))
print(precision_score(y_test,y_pred1))
```

```
0.8704061895551257
[[788 108]
 [ 26 112]]
0.509090909090909
```

```
mnb.fit(X_train,y_train)
y_pred2 = mnb.predict(X_test)
print(accuracy_score(y_test,y_pred2))
print(confusion_matrix(y_test,y_pred2))
print(precision_score(y_test,y_pred2))
```

```
0.971953578336557
[[896   0]
 [ 29 109]]
1.0
```

```
bnb.fit(X_train,y_train)
y_pred3 = bnb.predict(X_test)
print(accuracy_score(y_test,y_pred3))
print(confusion_matrix(y_test,y_pred3))
print(precision_score(y_test,y_pred3))
```

```
0.9835589941972921
[[895   1]
 [ 16 122]]
0.991869918699187
```

```
svc = SVC(kernel='sigmoid', gamma=1.0)
knc = KNeighborsClassifier()
mnb = MultinomialNB()
dtc = DecisionTreeClassifier(max_depth=5)
lrc = LogisticRegression(solver='liblinear', penalty='l1')
rfc = RandomForestClassifier(n_estimators=50, random_state=2)
abc = AdaBoostClassifier(n_estimators=50, random_state=2)
bc = BaggingClassifier(n_estimators=50, random_state=2)
etc = ExtraTreesClassifier(n_estimators=50, random_state=2)
gbdt = GradientBoostingClassifier(n_estimators=50,random_state=2)
xgb = XGBClassifier(n_estimators=50,random_state=2)
```

```
For  SVC
Accuracy -  0.9748549323017408
Precision -  0.9745762711864406
For  KN
Accuracy -  0.9052224371373307
Precision -  1.0
For  NB
Accuracy -  0.971953578336557
Precision -  1.0
For  DT
Accuracy -  0.9332688588007737
Precision -  0.8415841584158416
For  LR
Accuracy -  0.9564796905222437
Precision -  0.9696969696969697
For  RF
Accuracy -  0.9758220502901354
Precision -  0.9829059829059829
For  AdaBoost
Accuracy -  0.9613152804642167
Precision -  0.9454545454545454
For  BgC
Accuracy -  0.9593810444874274
Precision -  0.8692307692307693
```

Deployment

A supportive tool using a browser plugin or API can be built for companies running their own email servers so that they can keep a check on emails and can identify and flag spam emails. Such a supportive tool can be used in conjunction with existing email service providers as well.

Data Ethics

There are many ethical and legal issues that can really take a toll on designing such models. Bank and Investment organizations that run their own email servers have confidential emails and identity information related to customers. Using such confidential information for the predictive analysis required to safeguard the client data with the care of a professional fiduciary. Need to protect the customer data from both intentional and inadvertent disclosure, also protecting it from misuse.

Also, while implementation it is possible to generate false positive, which means the emails which are not spam fall into the spam category. An important piece of information a company can miss if the user’s legit email is marked as spam. A client or user who is a loyal customer, his email can be marked spam, which is an ethical issue.

Program Flow

We have four classes in total. One is the parent and three child classes.

- Parent class: data_read_write
- Child class: generate_word_cloud
- Child class: data_cleaning
- Child class: apply_embedding_and_model

CONCLUSION

Given a set of words, we used feature selection to obtain words which allow us to distinguish between spam and ham emails. We also compared the accuracy of various classifiers in predicting the class attribute. We see that RF method gives the highest classification accuracy no matter how many attributes are used and which method is used. It is possible that the accuracy may increase more than 97.5 % if we further increase the number of attributes.

INTRODUCTION

In today's globalized world, email is a primary source of communication. This communication can vary from personal, business, corporate to government. With the rapid increase in email usage, there has also been increase in the SPAM emails. SPAM emails, also known as junk email involves nearly identical messages sent to numerous recipients by email. Apart from being annoying, spam emails can also pose a security threat to computer system. It is estimated that spam cost businesses on the order of $100 billion in 2007. In this project, we use text mining to perform automatic spam filtering to use emails effectively. We try to identify patterns using Data-mining classification algorithms to enable us classify the emails as HAM or SPAM.

LEARNING DATA The data used for this project was taken from the Spam Assassin public corpus website. It consists of two data sets: train and test. Each dataset contains a randomly selected collection of emails in plain text format, which have been labelled as HAM or SPAM. The training data is used to build a model for classifying emails into HAM and SPAM. The test data is used to check the accuracy of the model built with the training data. The training data set contains 400 emails with 283 ham and 117 spam emails. The test data contains 200 emails with 139 ham and 61 spam emails.

DATA PREPROCESSING

The emails in the learning data are in plain text format. We need to convert the plain text into features that can represent the emails. Using these features we can then use a learning algorithm on the emails. A number of pre-processing steps are first performed. We convert the plain text files to files with one word per line. In this project, we look at emails just as a collection of words. So, to make it easier we convert each file into a list of words using Bourne Shell Scripts (extractmultfiles.sh and extractwords.sh).The output files are named as 'filename.words.'

STOP WORDS

There are some English words which appear very frequently in all documents and so have no worth in representing the documents. These are called STOP WORDS and there is no harm in deleting them. Example: the, a, for etc. There are also some domain specific (in this case email) stop words such as mon, tue, email, sender, from etc. So, we delete these words from all the files using a Bourne Shell Script. These words are put in a file 'words.txt'. The shell script takes multiple files as an argument and then deletes all the stop words mentioned in the words.txt file.

STEMMING

The next step to be performed is stemming. Stemming is used to find a root of a word and thus replacing all words to their stem which reduces the number of words to be considered for representing a document. Example: sings, singing, sing have sing as their stem. In the project, we use JAVA implementation of Porter stemming algorithm which is slightly modified to meet our needs. The resultant files are named with an extension 'words_stemmed'.