comp723-text-mining

Group member name	student id
Yousef Aldawoud	18038023
Bilal Siddque	17956171

Abstract

This project is a small illustration of modern text mining and tools, and how it is use these days. The projects takes an email data set that has a large number of spam emails. Our objective is to build a text mining (machine learning) model to classify spam emails from others.

Introduction

As the lkdrjejtj increases for example dvgfgdfg. Which generates all types of data. In this study we will be focusing on text mining. Text mining is defined as "the process of analyzing natural language text to glean information & patterns that are useful". Emailing is a very common way of communication used by both organisations and ind ery unstructured, which makes text mining challenging.

In this study we will be training different types of models using different libraries in python particularly Natural Language Tool-Kit (NLTK) and Scikit-Learn. These models are going to be trained on how to correctly identify legitimate emails from spam emails. The models we used to train and test the data are: fdhgdkjfhg dkf iofhsdklh dsoifhjkld

Data description

The data provided is a set of E-mails that were enlisted to 2 groups Spam emails (Emails that were useless to the people who received it) and Ham E-mails that mattered for the users.

The data contains `33,008` records in total. 3 mails were defective (includes characters that can't be processed) The emails were split into 5 files in no particular order.

Data statistics of total data set

Email type	Number of records	Percentage over-all
Spam emails	16464	49.879%
Ham (legitimate) emails	16544	50.121%

Methods

Data collection:

The data was provided to us by Auckland University of technology. The data is already separated into ham or spam folder respectfully. There are 5975 number of total email. Number of ham emails are 3672 and the number of spam emails are 1500. the ratio of ham emails to spam emails is 1:3 (spam:ham). The first ham email dates back to 10-12-1999, last ham email dates back to 11-01-2002. The first spam email dates back to 18-12-2003, last spam email dates back to 06-09-2005.

Data preparation:

Below are the pre-processing step we took to get the data ready for our models to be trained on

1. Splitting data into training and testing
2. Adding labels to the emails respectfully (ham or spam)
3. Stemming the data
4. Removing stop words
5. Removing punctuations
6. Vectorization

Splitting data into training and testing

For spliting the data set we used 2 different type of data spliting methods. the 2 methods were: * Using a 30:70 testing:training set whilst maintaining the ham:spam ratio * Using enron1, enron3 & enron5 as training and enron2 & enron4 for testing

We did this to ensure we can minimise any loss in accuracy. which could be caused by spliting the data sets.

Method One

When preforming the 30:70 used 30 percent of the email as testing about `9,903` emails and 70% of the emails about `23,108` for training the each of the classifiers. almost a 50:50 ratio of ham:spam was maintained between the test and training set.

Training set

The training set was created by using 70% of the original data set. It had equal parts ham and spam emails, containing `11,554` of both ham and spam emails. Containing `23,108` emails in total.

Email type	Number of records	Percentage over-all
Spam emails	11,554	50%
Ham (legitimate) emails	11,554	50%

Testing set

The testing set was created by using 30% of the original data set. it had almost equal pasts ham and spam, containing `4,910` spam emails & `4,900` ham emails with a total emails of `9,810`.

Email type	Number of records	Percentage over-all
Spam emails	4,910	50.01%
Ham (legitimate) emails	4,900	49.99%

Method Two

Training set

The training data set contain 3 folders out of 5. The total number of records of the training set is `15,865` the folders used in the training set are enron1, enron3 and enron5

Email type	Number of records	Percentage over-all
Spam emails	15,410	42.10%
Ham (legitimate) emails	9,187	57.90%

Testing set

The testing data set contain 2 folders out of 5. The total number of records of the testing set is `17362` the folders used in the testing set are enron2 and, enron4

Email type	Number of records	Percentage over-all
Spam emails	10,499	60.47%
Ham (legitimate) emails	7,362	39.13%

Adding labels

To ensure that the dataset doesn't take a large space in the storage we converted the labels to binary `(0 for spam, 1 for ham)`

Stemming

Stemming is done in order to normalize textual data. It also helps in reducing the number of words in the corpus. Which in-turn helps machine learning algorithms to perform better.

Removing stop words & punctuations

Stop words are words in sentences which do not add any additional meaning to the sentence. So, therefore they can be removed from the corpus in our case. Furthermore, removing punctuation helps in tokenization of the corpus, which we will get into next. Removal of stop words and punctuations both results in decreasing the size of the corpus which in-turn yields greater performance in machine learning algorithms.

Vectorization

Since most machine learning models doesn't work with text directly we had to convert the data set to a vector to be able to process it. We used `TFIDF (Term Frequency Inverse Document Frequency)`

`TF = Number of time word appear in the document / Total number of word in document`

Machine learning process

Algorithms

We used 3 different types of algorithms, to level out the playing field in order to create the model which best classifies the emails with the highest accuracy, precision & recall

The three algorithms we used: - Naive Bayes - Decision Tree - Neural Network

All of the algorithms were used from `sklearn` library in order to create the models.

Naive Bayes

The naive Bayes algorithm is one of the most powerful and commonly used algorithm in machine learning. this algorithms uses supervised leaning and classifies using the Bayes theorem reff. It is particularly easy to build and, one of the advantages of using Naive Bayes algorithm in our case, is that it works great of data sets which are large reffffff.

Decision Tree

The decision Tree is another very commonly used algorithm used in machine learning. It uses a supervised approach which finds the best way to split the data set on different conditions.

Neural Networks

The specific type of neural network well be using is from the `sklearn` library call `MLP` which is short for Multi-layer Perceptron. Has a minimum of three layers, which uses a supervised approach to classify.

Feature selection

Feature selection is the process of choosing a subset of features from the original data set. This effects the machine learning process in multiple ways, in consequence it also helps increase the accuracy of the models. It helps the machine learning process by, reducing the overall corpus size, by decreasing the number of features the algorithm has to process. Making training and applying new algorithms easier. Another way feature selection positively affects the machine learning process is by getting rid of the feature which are noisy. Resulting in the algorithms preforming better.

Results

Since we had split the data 2 different ways. We obtained 2 different set of results for the algorithms we ran.

Splitting method One

As discussed earlier, method one of splitting. Splits the data set into 30% for testing and 70% for training.

Findings

When running the algorithms, we found that the decision tree algorithm out preformed both Naive Bayes and the MLP classifier. With an accuracy of 0.94, recall of 0.83 & precision of 0.94.

Summary

task one	Naive Bayes	Decision tree	NN
Accuracy	0.84375	0.9375	0.62444
recall	0.66666	0.83334	0.523174
Precision	0.89	0.94	0.611357

Splitting method Two

As discused earlier, method two of splitting. splits the enron folder for training and testing. Enron1, enron3 & enron5 for training and enron2 & enron4 for testing.

Findings

Similar finding were observed, when splitting the data set by method 2. The decision tree classifier had still the best performance. However a significant increase in the Naive Bayes classifier was observed. On the other Hand, MLP classifier performed even worse than the performance in method 1.

Summary

task two	Naive Bayes	Decision tree	NN
Accuracy	0.912489	0.92981	0.59484
recall	0.76248	0.85614	0.51156
Precision	0.90632	0.93845	0.57591

Discussion

Explanation of Results

In this study we mainly focused on building the best classifier to correctly classify emails as spam or ham (legitimate) emails. Looking at the problem, we wanted to create a classifier that will have a very less likelihood of misclassify ham (legitimate) emails as spam. In saying that, our main goal for finding the best classifier was having the best precision metric.

In our case precision of the classifier was more significant of a metric than accuracy and recall.

The best classifier we found was the decision tree classifier. Due to the fact, it scored the highest results in both different data splitting methods.

Computational power || Problems we Faced

One of the main challenges we faced is not having enough processing power to run the algorithms for vectorization. When we tried to run the vectorizing algorithms on our PCs we ended up getting `out of memory` error (An error that occurs when there's a lack of RAM memory for the running script)

Possible solutions

Adding more RAM (Random access memory)

As discussed above the main problem we faced is not having enough RAM memory to carry out the process of vectorizing the dataset. Adding more RAM was a possible solution however in our case this option wasn't feasible due to the uncertainty of the amount of the RAM we needed to run the algorithms efficiently.

Reducing corpus size

The reason that the algorithm raises a memory error is having a lot of documents to process. Hence reducing corpus size was a viable option to make the algorithms run our PCs. However, reducing the corpus size will have negative effects on the machine learning model accuracy.

Process outsourcing

This approach seems to be the most viable option in our case. We used Azure cloud services, in order to run our algorithms. Since we had experience in managing servers. We took this approach, as our solutions, to the main problem we came across with.

Conclusion

As we have gone through the whole process of a creating intelgent systems out of data, we have a clear understanding of how data mining and text processing works. When doing this assignment our main goal is to create a machine learning model that can classify emails to spam and legit emails.

We were able to train three different models (Naive bayes, Decision tree, Neural network) on the dataset provided using Microsoft Azure servers and we found that decision tree was the best classifier as a solution for this problem, due to it scoring the highest values in accuracy and precision.

Acknowledgements

References

- https://scikit-learn.org/
- https://www.datacamp.com/community/tutorials/feature-selection-python
-

Appendix

The code is seprated to different files and classes. Each class has a specific functionality.

The code is available one github

github link : https://github.com/bilsiq/comp723-text-mining/

EmailCollector.py

This class is responsible for reading all the emails and converting them into a dataframe that can be easily processed by any classifiers.

- importing the necessary libraries.

```
import sys
import os
from nltk import PorterStemmer
import pandas as pd
```

- Init method:

file_name defines the directory name such as `enron` and file_number defines the number of files there. `SPAM_CODE` is the code assigned to the spam emails and `HAM_CODE` is the number assigned to the legit emails

```
class EmailCollector:
    SPAM_CODE = 0
    HAM_CODE = 1
    START_POINT = 1
    NUMBER_OF_FOLDERS = 2

    def __init__(self, file_name, file_numbers):
        self.file_name = file_name
        self.training_set = []
        self.invalid_files_number = 0
        self.file_numbers = file_numbers
```

Example :

```
emailCollector = EmailCollector("enron",[1,2,5])
# it will read the files inside the folders enron1, enron2, and enron5
```

- get_file_contents method:

This method is used to get the file contents of any file. It is a static method . file name refers to the targeted file.

```
class EmailCollector:
...
    @staticmethod
    def get_file_contents(file_name):
        with open(file_name, "r") as f:
            return f.read()
```

Example:-

```
'''
file in directory
./main.py
./note.txt >> "Hello world"
'''
note = EmailCollector::get_file_contents("note.txt")
print(note)
# outputs note.txt content 'Hello world'
```

- get_files_in_path method: Gets every file in a specific directory.

```
class EmailCollector:
...
    @staticmethod
    def get_files_in_path(path):
        return [f for f in os.listdir(path) if os.path.isfile(os.path.join(path, f))]
```

Example :-

```
'''
file in directory
./main.py
./notes/.
./notes/note-1.txt
./notes/note-2.txt
'''
notes = EmailCollector::get_files_in_path("/notes")
print(notes)
# outputs ['note-1.txt', 'note-2.txt']
```

- get_email_data method:

Gets the data inside the files and converts it to a dataframe.

```
emailCollector = EmailCollector("enron",[1,2,5])
emails = emailCollector.get_email_data("data")
print(emails)
# outputs emails with label
```

TextFilter.py

A class that cleans the text of unnecessary data - importing necessary libraries

```
from nltk import word_tokenize
from nltk import PorterStemmer
from nltk.corpus import stopwords
from string import punctuation
from sklearn.feature_extraction.text import CountVectorizer
import pandas as pd
from EmailCollector import EmailCollector
```

- Init method

sets the text and the stemmer type for the email

```
class TextCleaner:
    def __init__(self,text,stemmer):
        self.text = text
        self.stemmer = stemmer
```

- stem method

Stems the text.

```
class TextFilter:
    ...
    def stem(self):
        text_array = [self.stemmer.stem(word) for word in self.tokenize()]
        return TextCleaner(" ".join(text_array), self.stemmer)
```

- remove_stop_words method:

removes stop words

```
class TextCleaner:
    ...
    def remove_stop_words(self):
        filtered_words =
            [word for word in self.tokenize() if word not in stopwords.words('english')]
        return TextCleaner(" ".join(filtered_words),self.stemmer)
```

- remove_punctuation :

removes all punctuation

```
class TextCleaner:
    ...
    def remove_punctuation(self):
        filtered_words = [word for word in self.tokenize() if word not in punctuation]
        return TextCleaner(" ".join(filtered_words),self.stemmer)
```

- __str__ :

returns string after being processed:

```
class TextCleaner:
    ...
    def __str__(self):
        return self.text
```

classfiers.py

- Importing necessary libraries:

```
import pandas as pd
from sklearn.tree import DecisionTreeClassifier # Import Decision Tree Classifier
from sklearn.model_selection import train_test_split # Import train_test_split function
from sklearn import metrics
from sklearn.feature_selection import SelectKBest, chi2
from sklearn.metrics import confusion_matrix
from sklearn.naive_bayes import GaussianNB
import numpy as np
from sklearn.feature_selection import RFE
from sklearn.linear_model import LogisticRegression
```

- feature_select

Selects a number of features

```
def feature_select(data):
    columns = data.columns.values
    X,Y = data[columns[0:len(columns)-1]], data["email_label"]
    model = LogisticRegression()
    rfe = RFE(model, 30)
    fit = rfe.fit(X, Y)
    cols = []
    x = 0
    for col in columns:
        try:
            if fit.support_[x]:
                cols.append(col)
            x+=1
        except:
            print(x)
    return cols
```

- get_accuracy method

Takes confusion matrix as a parameter to calculate the accuracy.

```
def get_accuracy(cm):
    true_positive, false_positive = cm[0]
    false_negative, true_negative = cm[1]
    x = (false_negative+false_positive+true_negative+true_positive)
    result = (true_negative+true_positive)/x
    return result
```

- get_precision method

Takes confusion matrix as a parameter to calculate the precision.

```
def get_precision(cm):
    true_positive, false_positive = cm[0]
    false_negative, true_negative = cm[1]
    try:
        result = true_positive / (true_positive + false_positive)
    except:
        result = 0.404
    return result
```

- get_recall method

Takes confusion matrix as a parameter to calculate the recall.

```
def get_recall(cm):
    true_positive, false_positive = cm[0]
    false_negative, true_negative = cm[1]
    try:
        result = true_positive / (true_positive + false_negative)
    except:
        result = 0.404
    return result
```

- Creating and training naive bayes model

```
model = GaussianNB()
data = pd.read_csv("training_file.csv")
columns = data.columns.values

X=data[columns[0:len(columns)-1]]
y=data['email_label']
model.fit(X,y)
```

- Using the model to predict the test dataset

Also creating a confusion matrix from the test data set and the model predicted values

```
y_pred=model.predict(test[columns[0:len(columns)-1]])
cm = confusion_matrix(test["email_label"], y_pred)
```

- printing out the accuracy ,recall and precision "'python print("Precision =",get_precision(cm)) print("recall =",get_recall(cm)) print("Accuracy =",get_accuracy(cm))

`- Creating and training decision tree model:`python

model = DecisionTreeClassifier() model.fit(X,y) print("Decision tree :-") y_pred=model.predict(test[new_features]) cm = confusion_matrix(data["email_label"], y_pred) print(cm)

print("Precision =",get_precision(cm)) print("recall =",get_recall(cm)) print("Accuracy =",get_accuracy(cm))

"'

NN.py

- Creating and training neural network model

```
model = MLPClassifier(hidden_layer_sizes=(13,13,13),max_iter=500)
model.fit(X_train,y_train)
```

- predicting the test dataset

```
predictions = model.predict(X_test)
```

- showing confusion matrix

```
print(confusion_matrix(y_test,predictions))
```

- showing recall, precision, and accuracy

```
print("Precision = ",get_precision(cm))
print("recall = ",get_recall(cm))
print("Accuracy = ",get_accuracy(cm))
```