Email-Spam Detection Model

Ameet Deshpande

May 2017

1 Abstract

This is a spam-classifier built using the python programming language on the ENRON-SPAM dataset. The processed version of ENRON-SPAM dataset is being used for training and testing. The scikit-learn library of python is used for training and testing. The features being used are words, both unigram and bigram, after removing the stopwords. The classifier being used is the Naive-Bayes Classifier.

2 Code

```
import os
import numpy
from pandas import DataFrame
from sklearn.feature_extraction.text import
    ↪ CountVectorizer
from nltk.corpus import stopwords
from sklearn.naive_bayes import MultinomialNB
import sys
sys.path.insert(0,Training Directory)
import set_test_data

# Process all the files and store the text in a data
    ↪ frame


# Rows is a list of the text and the class(spam or ham)
    ↪ that it belongs to
rows = []
index = []

for (root, dirs, files) in os.walk('.', topdown=True):
    for name in dirs:
        # For all the sub-directories in the root folder,
            ↪ if the directory contains emails
        if str(name) == 'spam' or str(name) == 'ham':
            current_path = str(os.path.join(root, name))
            files1 = os.listdir(current_path)
            for file_name in files1:  # For all the files
                ↪ in the root directory
                f1 = open(current_path + str('/') + str(
                    ↪ file_name), 'r')
                content = f1.readlines()
                content = '␣'.join(content)  # Joins the
                    ↪ contents of the list into one
                    ↪ single string separated by a space
                if str(name) == 'spam':  # Append the
                    ↪ text after assigning the correct
                    ↪ class
                    rows.append({'text': content, 'class'
                        ↪ : 1})
                    index.append(str(file_name))
                else:
                    rows.append({'text': content, 'class'
                        ↪ : 0})
                    index.append(str(file_name))

                f1.close()

# Create a data_frame and randomly permute it

data_frame = DataFrame(rows, index=index)

# Add this if you want to split training and test data
    ↪ from single Data Frame
"""data_frame = \
    data_frame.reindex(numpy.random.permutation(
        ↪ data_frame.index))"""

# Extracting the features, here, the counts of the words

# Adding the word subject to stopwords, as it tends to
    ↪ appear in every email
stop = set(stopwords.words('english'))
stop = list(stop)
stop.append(unicode("subject"))

count_vectorizer = CountVectorizer(stop_words=stop,
    ↪ decode_error='ignore', ngram_range=(1,2))       #
    ↪ We used unigram and bigram counts
counts = count_vectorizer.fit_transform(data_frame['text'
    ↪ ].values)    # .values gives the underlying numpy
    ↪ array

# Training the data using Naive Bayes Classifier
classifier = MultinomialNB()
targets = data_frame['class'].values
classifier.fit(counts, targets)

test_data_frame = set_test_data.build_test_data()    #
    ↪ Test data is obtained as a data frame
test_counts = count_vectorizer.transform(test_data_frame[
    ↪ "text"])
predictions = classifier.predict(test_counts)
correct_answers = test_data_frame["class"]

till = len(predictions)
confusion_matrix = [[0,0],[0,0]]     # To store the
    ↪ precision and recall

for i in range(till):
    confusion_matrix[correct_answers[i]][predictions[i]]
        ↪ += 1

print "␣␣␣␣␣␣","Actual␣Ham","Actual␣Spam"
print "Predicted␣Ham␣␣", confusion_matrix[0]
print "Predicted␣Spam␣␣", confusion_matrix[1]
```

3 Analysis and Accuracy

The model gives an accuracy of 96.334 percent when enron1,2,3,4 are used as the training data and enron5,6 are used as the test data