Chapter 6

Applications

6.1 Spam Filtering

6.1.1 Algorithm description

Text mining or deriving information from text, is a wide field which has gained popularity with the huge text data being generated. Automation of a number of applications like sentiment analysis, document classification, topic classification, text summarization, machine translation has been done using machine learning models. Spam filtering is a beginner's example of document classification task which involves classifying an sms or email as spam or non-spam (a.k.a. ham). Spam box in your Gmail account is the best example of this. So lets get started in building a spam filter on a publicly available sms corpus. The extracted subset on which we will be working can be downloaded from here.

We will walk through the following steps to build this application :

1. Preparing the text data.
2. Creating word dictionary.
3. Feature extraction process
4. Training the classifier

Further, we will check the results on test set of the subset created.

Preparing the text data

The data-set used here, has to be split into a training set and a test set. You must divide it equally between spam and ham sms. You will easily recognize spam and ham sms. In any text mining problem, text cleaning is the first step where we remove those words from the document which may not contribute to the information we want to extract. Sms may contain a lot of undesirable characters like punctuation marks, stop words, digits... which may not be helpful in detecting the spam sms. We'll use the bag-of-words approach, where each unique word in a text will be represented by one number. As a first step, we have to split a message into its individual words. You have to process in the sms in the following ways:

a) Remove of stop words – Stop words like "and", "the", "of", etc are very common in all English sentences and are not very meaningful in deciding spam or legitimate status, so these words have to be removed from the sms. b) Lemmatization – It is the process of grouping together the different inflected forms of a word so they can be analysed as a single item. For

example, “include”, “includes,” and “included” would all be represented as “include”. The context of the sentence is also preserved in lemmatization as opposed to stemming (another buzz word in text mining which does not consider meaning of the sentence). We still need to remove the non-words like punctuation marks or special characters from the sms documents. There are several ways to do it. Here, we have to remove such words after creating a dictionary, which is a very convenient method to do so since when you have a dictionary, you need to remove every such word only once.

Creating word dictionary

A sample sms in the data-set looks like this:

1. ham Go until jurong point, crazy.. Available only in bugis n great world la e buffet... Cine there got amore wat...
2. ham Ok lar... Joking wif u oni...
3. ham U dun say so early hor... U c already then say...
4. ham Nah I don’t think he goes to usf, he lives around here though
5. spam FreeMsg Hey there darling it’s been 3 week’s now and no word back! I’d like some fun you up for it still? Tb ok! XxX std chgs to send, £1.50 to rcv

This corpus will be our labeled training set. Using these ham/spam examples, we’ll train a machine learning model to learn to discriminate between ham/spam automatically. Then, with a trained model, we’ll be able to classify arbitrary unlabeled messages as ham or spam. It can be seen that the first word in each line is ham or spam and you will found the content of the sms after in the line. You will only perform text analytics on the content to detect the spam sms.

As a first step, we need to read the data from the sms.txt file and create the target and data list.

```
from sklearn import datasets
from sklearn.datasets import fetch_mldata
from sklearn.neural_network import MLPClassifier
import matplotlib.pyplot as plt
from collections import Counter

#open the file in the directory
file = open("C:/OLIVIER/PEDAGO/2018/APS/data/sms.txt", "r")
line = file.readline()

target=[]
data=[]

while 1:
    # split each line in a list, the delimiters are removed
    line = file.readline().split()

    # stop reading when the len of the line is equal to zero
    if len(line) ==0:
        break

    target.append(line[0])
```

```
            del(line[0])
            data.append(line)

        file.close()
```

As a second step, we need to create a dictionary of words and their frequency. For this task, training set of half sms is utilized. This python function creates the dictionary for you.

```
def make_Dictionary(train_dir):
    all_words = []
    # create a list of all words
    # to do
    %for i in range(len(data)):
    %    for j in range(len(data[i])):
    %        all_words += data[j]
    dictionary = Counter(all_words)
    # Paste code for non-word removal here
    return dictionary
```

Once the dictionary is created we can add just a few lines of code written below to the above function to remove non-words about which we talked in the first step. I have also removed absurd single characters in the dictionary which are irrelevant here. Do not forget to insert the below code in the previous function.

```
list_to_remove = dictionary.keys()
for item in list_to_remove:
    if item.isalpha() == False:
        del dictionary[item]
    elif len(item) == 1:
        del dictionary[item]
dictionary = dictionary.most_common(3000)
```

Dictionary can be seen by the command print dictionary. You may find some absurd word counts to be high but don't worry, it's just a dictionary and you always have the scope of improving it later. Make sure your dictionary has some of the entries given below as most frequent words. Here I have chosen 3000 most frequently used words in the dictionary. [('order', 1414), ('address', 1293), ('report', 1216), ('mail', 1127), ('send', 1079), ('language', 1072), ('email', 1051), ('program', 1001), ('our', 987), ('list', 935), ('one', 917), ('name', 878), ('receive', 826), ('money', 788), ('free', 762)

Feature extraction process

Once the dictionary is ready, we can extract word count vector (our feature here) of 3000 dimensions for each sms of training set. Each word count vector contains the frequency of 3000 words in the training file. Of course you might have guessed by now that most of them will be zero. Let us take an example. Suppose we have 500 words in our dictionary. Each word count vector contains the frequency of 500 dictionary words in the training file. Suppose text in training file was "Get the work done, work done" then it will be encoded as [0,0,0,0,0,.....0,0,2,0,0,0,.....,0,0,1,0,0,...0,0,1,0,0,.....2,0,0,0,0,0]. Here, all the word counts are placed at 296th, 359th, 415th, 495th index of 500 length word count vector and the rest are zero.

Once the classifiers are trained, we can check the performance of the models on test-set. You extract word count vector for each sms in test-set and predict its class (ham or spam) with the trained classifier.

6.1.2 Implementation

Exercise 14 (Spam filtering implementation)**.**

1. *Implement the spam filtering algorithm on the SMS dataset by using the labels (ham or spam). You have to split the dataset into a training dataset and testing dataset.*
2. *Test is there any difference in message length between spam and ham?*
3. *Check the performance of the method*
4. *Use on the same dataset the SVM method for the classification*

6.2 Word2Vec

6.2.1 Algorithm description

We're going to train a neural network to do the following. Given a specific word in the middle of a sentence (the input word), look at the words nearby and pick one at random. The network is going to tell us the probability for every word in our vocabulary of being the "nearby word" that we chose. When we say "nearby", there is actually a "window size" parameter to the algorithm. A typical window size might be 5, meaning 5 words behind and 5 words ahead (10 in total). The output probabilities are going to relate to how likely it is find each vocabulary word nearby our input word. For example, if you gave the trained network the input word "Soviet", the output probabilities are going to be much higher for words like "Union" and "Russia" than for unrelated words like "watermelon" and "kangaroo". We'll train the neural network to do this by feeding it word pairs found in the training documents "text8.zip".

The below example shows some of the training samples (word pairs) we would take from the sentence "The quick brown fox jumps over the lazy dog." A small window size of 2 is used in this example. The word highlighted in blue is the input word.