Building a Spam Detector - I

You've learnt all the basic preprocessing steps required for most text analytics applications. In this section, you will learn how to apply these steps to build a spam detector.

Until now, you had learnt how to use the scikit-learn library to train machine learning algorithms. Here, Krishna will demonstrate how to build a spam detector using NLTK library which is, as you might have already realised, is your go-to tool when you're working with text.

Now, it is not necessary for you to learn how to use NLTK's machine learning functions. But it's always nice to have knowledge of more than one tool. More importantly, he'll demonstrate how to extract features from the raw text without using the scikit-learn package. So take this demonstration as a bonus as you'll learn how to preprocess text and build a classifier using NLTK. Before getting started, download the Jupyter notebook provided below to follow along:

SPAM Ham Detection

In [1]:

```
import random
import nltk
import pandas as pd
from nltk.tokenize import word_tokenize
from nltk.corpus import stopwords
from nltk.stem.porter import PorterStemmer
from nltk.stem import WordNetLemmatizer
```

In [2]:

```
## Reading the given dataset
spam = pd.read_csv("SMSSpamCollection.txt", sep = "\t", names=["label", "message"])
```

In [3]:

```
print(spam.head())
```

```
  label                                            message
0   ham  Go until jurong point, crazy.. Available only ...
1   ham                      Ok lar... Joking wif u oni...
2  spam  Free entry in 2 a wkly comp to win FA Cup fina...
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 aro...
```

In [4]:

```
## Converting the read dataset in to a list of tuples, each tuple(row) contianing the messa
data_set = []
for index,row in spam.iterrows():
    data_set.append((row['message'], row['label']))
```

In [5]:

```
print(data_set[:5])
```

```
[('Go until jurong point, crazy.. Available only in bugis n great world la e
buffet... Cine there got amore wat...', 'ham'), ('Ok lar... Joking wif u on
i...', 'ham'), ("Free entry in 2 a wkly comp to win FA Cup final tkts 21st M
ay 2005. Text FA to 87121 to receive entry question(std txt rate)T&C's apply
08452810075over18's", 'spam'), ('U dun say so early hor... U c already then
say...', 'ham'), ("Nah I don't think he goes to usf, he lives around here th
ough", 'ham')]
```

In [6]:

```
print(len(data_set))
```

```
5572
```

Preprocessing

In [7]:

```
## initialise the inbuilt Stemmer and the Lemmatizer
stemmer = PorterStemmer()
wordnet_lemmatizer = WordNetLemmatizer()
```

In [8]:

```
def preprocess(document, stem=True):
    'changes document to lower case, removes stopwords and lemmatizes/stems the remainder o

    # change sentence to lower case
    document = document.lower()

    # tokenize into words
    words = word_tokenize(document)

    # remove stop words
    words = [word for word in words if word not in stopwords.words("english")]

    if stem:
        words = [stemmer.stem(word) for word in words]
    else:
        words = [wordnet_lemmatizer.lemmatize(word, pos='v') for word in words]

    # join words to make sentence
    document = " ".join(words)

    return document
```

In [9]:

```
## - Performing the preprocessing steps on all messages
messages_set = []
for (message, label) in data_set:
    words_filtered = [e.lower() for e in preprocess(message, stem=False).split() if len(e)
    messages_set.append((words_filtered, label))
```

In [11]:

```
print(messages_set[0])
```

```
(['jurong', 'point', 'crazy..', 'available', 'bugis', 'great', 'world', 'buf
fet', '...', 'cine', 'get', 'amore', 'wat', '...'], 'ham')
```

In [11]:

```
print(messages_set[:5])
```

```
[(['jurong', 'point', 'crazy..', 'available', 'bugis', 'great', 'world', 'bu
ffet', '...', 'cine', 'get', 'amore', 'wat', '...'], 'ham'), (['lar', '...',
'joke', 'wif', 'oni', '...'], 'ham'), (['free', 'entry', 'wkly', 'comp', 'wi
n', 'cup', 'final', 'tkts', '21st', 'may', '2005.', 'text', '87121', 'receiv
e', 'entry', 'question', 'std', 'txt', 'rate', 'apply', '08452810075over1
8'], 'spam'), (['dun', 'say', 'early', 'hor', '...', 'already', 'say',
'...'], 'ham'), (['nah', "n't", 'think', 'usf', 'live', 'around', 'though'],
'ham')]
```

Words less than a certain threshold are removed to eliminate special characters such as double exclamation marks, or double dots (the period character). And you won't lose any information by doing this because there are no words less than two characters other than some stopwords (such as 'am', 'is', etc.).

You've already learnt how to create a bag-of-words model by using the NLTK's CountVectorizer function. However, Krishna will demonstrate how to build a bag-of-words model without using the NLTK function, that is, building the model manually. The first step towards achieving that goal is to create a vocabulary from the text corpus that you have. In the following video, you're going to learn how to create vocabulary from the dataset.

Preparing to create features

In [12]:

```
## - creating a single list of all words in the entire dataset for feature list creation

def get_words_in_messages(messages):
    all_words = []
    for (message, label) in messages:
      all_words.extend(message)
    return all_words
```

In [15]:

```
## - creating a final feature list using an intuitive FreqDist, to eliminate all the duplic
## Note : we can use the Frequency Distribution of the entire dataset to calculate Tf-Idf s

def get_word_features(wordlist):
    #print(wordlist[:10])
    wordlist = nltk.FreqDist(wordlist)
    word_features = wordlist.keys()
    return word_features
```

In [22]:

```
## - creating the word features for the entire dataset
word_features = get_word_features(get_words_in_messages(messages_set))
print(len(word_features))
```

```
8395
```

In [32]:

```
print(type(word_features))
```

```
<class 'dict_keys'>
```

In [ ]:

```
### Preparing to create a train and test set
```

In [34]:

```
## - creating slicing index at 80% threshold
sliceIndex = int((len(messages_set)*.8))
print(sliceIndex)
```

```
4457
```

In [35]:

```
## - shuffle the pack to create a random and unbiased split of the dataset
random.shuffle(messages_set)
```

In [36]:

```
train_messages, test_messages = messages_set[:sliceIndex], messages_set[sliceIndex:]
```

In [38]:

```
print(len(train_messages))
print(len(test_messages))
```

```
4457
1115
```

You learnt how to create vocabulary manually using all the words in the text corpus. In the next section, you'll look at how to create a bag-of-words model.

Why do you think that Naive Bayes is a good choice when it comes to text classification problems such as spam detection

Naive Bayes assumes independence between features. Now, in text classification such as spam detection, only the presence of certain words matter. It doesn't matter if they occur before or after certain words. Hence, Naive Bayes often performs very well on these problems.

Preparing to create feature maps for train and test data

In [39]:

```
## creating a LazyMap of feature presence for each of the 8K+ features with respect to each
def extract_features(document):
    document_words = set(document)
    features = {}
    for word in word_features:
        features['contains(%s)' % word] = (word in document_words)
    return features
```

In [41]:

```
## - creating the feature map of train and test data

training_set = nltk.classify.apply_features(extract_features, train_messages)
testing_set = nltk.classify.apply_features(extract_features, test_messages)
```

In [42]:

```
print(training_set[:5])
```

```
False, 'contains(good)': False, 'contains(girls)': False, 'contains(situa
tion)': False, 'contains(seekers)': False, 'contains(part)': False, 'cont
ains(check)': False, 'contains(roommates)': False, 'contains(forever)': F
alse, 'contains(come)': False, 'contains(double)': False, 'contains(hai
r)': False, 'contains(dresser)': False, 'contains(wun)': False, 'contains
(cut)': False, 'contains(short)': False, 'contains(nice)': False, 'contai
ns(advise)': False, 'contains(follow)': False, 'contains(recent)': False,
'contains(review)': False, 'contains(mob)': False, 'contains(award)': Fal
se, 'contains(£1500)': False, 'contains(bonus)': False, 'contains(0906636
4589)': False, 'contains(song)': False, 'contains(dedicate)': False, 'con
tains(day..)': False, 'contains(valuable)': False, 'contains(frnds)': Fal
se, 'contains(rply)': False, 'contains(complimentary)': False, 'contains
(trip)': False, 'contains(eurodisinc)': False, 'contains(trav)': False,
'contains(aco)': False, 'contains(entry41)': False, 'contains(£1000)': Fa
lse, 'contains(dis)': False, 'contains(18+6*£1.50)': False, 'contains(mor
efrmmob)': False, 'contains(shracomorsglsuplt)': False, 'contains(ls1)':
False, 'contains(3aj)': False, 'contains(hear)': False, 'contains(divorc
e)': False, 'contains(barbie)': False, 'contains(ken)': False, 'contains
(plane)': False, 'contains(month)': False, 'contains(wah)': False, 'conta
ins(lucky)': False, 'contains(save)': False, 'contains(money)': False, 'c
```

In [43]:

```
print('Training set size : ', len(training_set))
print('Test set size : ', len(testing_set))
```

```
Training set size :  4457
Test set size :  1115
```

Training

In [44]:

```
## Training the classifier with NaiveBayes algorithm
spamClassifier = nltk.NaiveBayesClassifier.train(training_set)
```

Evaluation

In [45]:

```
## - Analyzing the accuracy of the test set
print(nltk.classify.accuracy(spamClassifier, training_set))
```

```
0.9923715503702042
```

In [46]:

```
## Analyzing the accuracy of the test set
print(nltk.classify.accuracy(spamClassifier, testing_set))
```

```
0.97847533632287
```

In [47]:

```
## Testing a example message with our newly trained classifier
m = 'CONGRATULATIONS!! As a valued account holder you have been selected to receive a £900
print('Classification result : ', spamClassifier.classify(extract_features(m.split())))
```

```
Classification result :  spam
```

In [55]:

```
## Testing a example message with our newly trained classifier
m = 'CONGRATULATIONS...!I charge only 1000 dollars but I am free free free free for you dar
print('Classification result : ', spamClassifier.classify(extract_features(m.split())))
```

```
Classification result :  ham
```

In [27]:

```
## Priting the most informative features in the classifier
print(spamClassifier.show_most_informative_features(50))
```

```
Most Informative Features
         contains(award) = True              spam : ham    =    192.5 : 1.0
         contains(nokia) = True              spam : ham    =    102.8 : 1.0
        contains(urgent) = True              spam : ham    =     95.2 : 1.0
        contains(camera) = True              spam : ham    =     86.7 : 1.0
       contains(attempt) = True              spam : ham    =     86.7 : 1.0
         contains(await) = True              spam : ham    =     82.5 : 1.0
           contains(txt) = True              spam : ham    =     79.8 : 1.0
        contains(latest) = True              spam : ham    =     74.9 : 1.0
          contains(land) = True              spam : ham    =     74.0 : 1.0
       contains(private) = True              spam : ham    =     74.0 : 1.0
       contains(service) = True              spam : ham    =     71.0 : 1.0
         contains(final) = True              spam : ham    =     69.8 : 1.0
          contains(rate) = True              spam : ham    =     69.8 : 1.0
     contains(statement) = True              spam : ham    =     65.6 : 1.0
      contains(landline) = True              spam : ham    =     64.7 : 1.0
         contains(video) = True              spam : ham    =     62.2 : 1.0
           contains(100) = True              spam : ham    =     57.1 : 1.0
          contains(club) = True              spam : ham    =     57.1 : 1.0
          contains(draw) = True              spam : ham    =     48.7 : 1.0
contains(congratulations) = True              spam : ham    =     48.7 : 1.0
          contains(opt) = True              spam : ham    =     48.7 : 1.0
        contains(caller) = True              spam : ham    =     48.7 : 1.0
        contains(mobile) = True              spam : ham    =     45.4 : 1.0
      contains(motorola) = True              spam : ham    =     44.4 : 1.0
          contains(quiz) = True              spam : ham    =     44.4 : 1.0
        contains(player) = True              spam : ham    =     44.4 : 1.0
         contains(apply) = True              spam : ham    =     44.4 : 1.0
         contains(music) = True              spam : ham    =     39.3 : 1.0
        contains(orange) = True              spam : ham    =     37.2 : 1.0
           contains(box) = True              spam : ham    =     36.0 : 1.0
        contains(txting) = True              spam : ham    =     36.0 : 1.0
        contains(direct) = True              spam : ham    =     36.0 : 1.0
      contains(discount) = True              spam : ham    =     36.0 : 1.0
          contains(info) = True              spam : ham    =     34.3 : 1.0
         contains(pound) = True              spam : ham    =     33.5 : 1.0
       contains(network) = True              spam : ham    =     33.1 : 1.0
          contains(user) = True              spam : ham    =     31.7 : 1.0
       contains(auction) = True              spam : ham    =     31.7 : 1.0
      contains(customer) = True              spam : ham    =     31.0 : 1.0
           contains(mat) = True              spam : ham    =     29.2 : 1.0
       contains(receive) = True              spam : ham    =     27.8 : 1.0
       contains(contact) = True              spam : ham    =     27.7 : 1.0
      contains(sunshine) = True              spam : ham    =     27.5 : 1.0
          contains(sony) = True              spam : ham    =     27.5 : 1.0
          contains(goto) = True              spam : ham    =     27.5 : 1.0
        contains(reveal) = True              spam : ham    =     27.5 : 1.0
           contains(del) = True              spam : ham    =     27.5 : 1.0
          contains(cash) = True              spam : ham    =     27.5 : 1.0
         contains(offer) = True              spam : ham    =     25.0 : 1.0
        contains(select) = True              spam : ham    =     24.8 : 1.0
None
```

We've got an excellent accuracy of 98% on the test set. Although this is an excellent accuracy, you could further improve it by trying other models.

Note that, Krishna has created a bag-of-words representation that's created from scratch without using the CountVectorizer() function. He has used a binary representation instead of using the number of features to represent each word. In this bag-of-words table, '1' means the word is present whereas '0' means the absence of that word in that document. You can do this by setting the 'binary' parameter to 'True' in the CountVectorizer() function.

You also saw that Krishna used the pickle library to save the model. After creating models, they are saved using the pickle library on the disk. This way, you can even send the models to be used on a different computer or platform.

The steps that you just saw should convince you that to get excellent results, you need to take extra care of the nuances of the dataset you're working on. You need to understand the data inside-out to take these steps because these can't be generalised to every text classifier or even other spam datasets.

In [28]:

```
## storing the classifier on disk for later usage
import pickle
f = open('nb_spam_classifier.pickle', 'wb')
pickle.dump(spamClassifier,f)
print('Classifier stored at ', f.name)
f.close()
```

```
Classifier stored at  nb_spam_classifier.pickle
```