Chapter 12

Text Analytics

**Abstract**

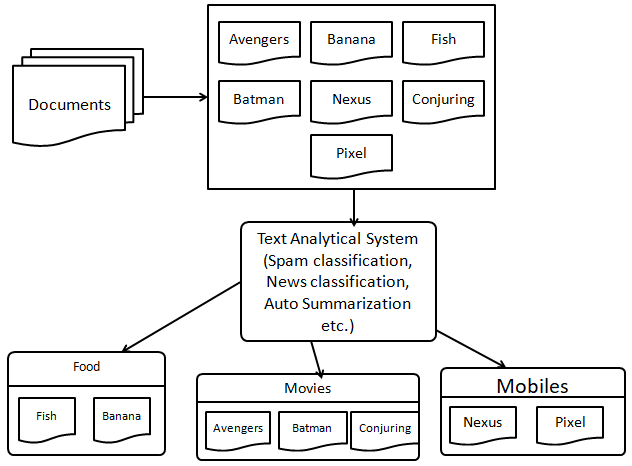
Most of the data generated appear in the form of text. Text analytics is the process of getting insights from the text by focusing on the small pieces of information in the text. The different applications of text analytics are sentimental analysis, spam classification and automatic summarization. Text analytics involes components such as pre-processing, generating bag of words, word cloud model and classification. During the pre-processing stage, the text can be divided into pieces of information with certain key phrases. Lemmatization and tokenization processing is used to get these pieces of text. Further, classification models like Naïve Bayes, decision tree and SVM can be used in text analytics. These models mainly help in the categorization of the text into different classes based on the initial pre-processing of the textual data. In the previous chapters, the classification models with basic examples were discussed. In this chapter, the different stages of text analytics is first discussed and later followed by the case studies. Thre case studies that are discussed as a part of this chapter are Automatic summarization, spam classification, question classification and sentimental analysis.

# Text Analytics

## *Introduction to Text Analytics*

Most of the information available on the web is in textual format or in a semi-structured way. Analytics on this textual information plays an important role in gaining insights. Text analytics is a process of representation and modeling of textual content to gain insights into it. It helps in uncovering the interesting patterns underlying the large text information. For example, text analytics on product reviews can be helpful in making decisions around the good products and bad products. One of the main intentions of text analytics is to categorize documents into various classes [1]. This type of classification helps in applications such as spam classification, news classification and story classification.

A conceptual view of text analytics is represented in the figure 12.1. A collection of documents or in a single document large amount of information is present. This information need to be segregated into separate classes so that it is easy to identify the different classes of information. For example, in the figure 12.1 a collection of documents such as Avengers, Conjuring, Batman and so on are present. This collection can be categorized into three main classes namely Food, Mobile Phones and Movies. In this way, each text classification technique at the end produces different classes of documents. In this chapter, the different concepts around the text analytics is discussed with examples.



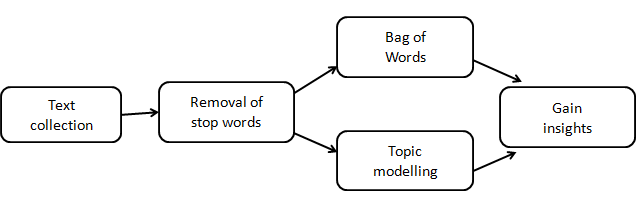
**Fig. 12.1.** **Conceptual view of Text Analytics**

## *Steps involved in Text Analytics*

Since, text analytics play a vital role in gaining insights about various information, necessary steps need to be performed for carrying out analytics [1] [2]. The various steps that are involved in Text Analytics are as shown in the figure 12.2. The process starts with collection of text, removal of stop words, generating bag of words, topic modelling and finally gain insights around it. The different types of information processed in each step are discussed as follows:

* **Text collection:** The very first step in text analytics is collection of text i.e. either in raw format, semi-structured format. The unstructured data such as tweets, RSS feed need to be pre-processed first and then analysis can be carried out. Some of the techniques for text collection are included as a part of web scraping.
* **Removal of stopwords:** Some of the phrases such as is,was,if,else,then,thus,so etc does not convey significant information about the text. Such phrases are called as stopwords. These stopwords are removed from the text initially for further processing.
* **Bag of words:** In the next phase of text analytics, a word cloud model and bag of words is built. A vector of words representing their frequencies in the text is depicted through the bag of words and word cloud model.
* **Topic modelling:** The pre-processed text in the earlier phases are used for categorization of documents based on the topic. For example, spam/non-spam, sports/politics/media etc.

For each of this phases of text analytics, a small example and its significance is presented in the upcoming sections of the chapter. The same steps are applied for two case studies discussed in this chapter.



**Fig. 12.2. Text Analytics steps**

## *Names, Numbers and Stopwords removal*

In the text classification process, the first step is to pre-process the information of the text so that the significant content of the information is available for text analytics. In this section, a small example is presented in python for removal of stopwords, names and numbers in the text that doesn’t cary necessary information [3]. The following code demonstrates the removal process of numbers and stopwords.

The main module required for text analytics and classification is nltk module. In this module *‘stopwords’* module is used for removal process. A sentence is initialized first that contains numbers and stopwords. Some of the examples of stopwords are if,is,was,else,in,not etc. The sentence is converted to lower case first using the *lower()* function and then the punctuations are removed using *punctuation()* method.

The stopwords are then removed from the sentence using the stopwords module by using the parameter *english* to it. The numbers in the sentence are removed using the regular expression *[0-9],* that identifies the numbers in the sentence. The original sentence and the pre-processed sentence are presented in the code.

import string

from nltk.corpus import stopwords

import re

sentence="PretzelBros, airbnb for people who like pretzels, raises $2 million"

sentence=sentence.lower()

sentence

**'pretzelbros, airbnb for people who like pretzels, raises $2 million'**

symbols=string.punctuation

sentence="".join([x for x in sentence if x not in symbols])

sentence

**'pretzelbros airbnb for people who like pretzels raises 2 million'**

sentence=" ".join([x for x in sentence.split() if x not in stopwords.words('english')])

sentence=re.sub('[0-9]',"",sentence)

print(sentence)

**pretzelbros airbnb people like pretzels raises million**

The last sentence in the code represents the sentence where the names, numbers and stopwords are removed. The next step for text classification/analytics is to calculate the number of times/frequency of words/terms in the text. In the next section, the process of carrying out word frequency analysis is presented.

## *Word frequency analysis*

Once the stopwords are removed from the text, the next step of text analytics is calculating the frequency of words in the text. The main aim of calculating the word frequency is to determine the sentences with highest frequency of the words. It is also referred to as Term frequency analysis. The number of occurrences of the word is calculated based on the corpus.

The following code demonstrates word frequency analysis in Python [2] [3]. A text is initialised with two sentences as a list in the *Text.* The sentence is pre-processed first by removing the stopwords using the same procedure as discussed in the previous section. *CountVectorizer* module is used for counting the occurrences of the word in the text. A small model is built using the *tf\_vectorizer()* function. The vocabulary of the model can be printed to see the occurrences of frequencies of each word in the document. For the same vocabulary, a graph can be plotted using the matplotlib module. The output of the plot is as shown in the figure 12.4.

from nltk.corpus import stopwords

import string

from sklearn.feature\_extraction.text import CountVectorizer

import  matplotlib.pyplot as plt

Text = ["Computer is a device that can be instructed to perform specified instructions." ,

       "Computer is used to automate manual labor through unmatching instructions execution."]

def preprocess(sentence):

    sentence=sentence.lower()

    sentence="".join([x for x in sentence if x not in string.punctuation])

    sentence=[x for x in sentence.split(" ") if x not in stopwords.words('english')]

    sentence=[x for x in sentence if x!='']

    return " ".join(sentence)

tf\_vectorizer = CountVectorizer(lowercase=True,preprocessor=preprocess)

model = tf\_vectorizer.fit(Text)

print(model.vocabulary\_)

x = [ i for i in range(len(model.vocabulary\_)) ]

y = []

x\_t = []

for item in model.vocabulary\_.keys():

    x\_t.append(item)

    y.append(model.vocabulary\_[item])

plt.figure(figsize=(30,30))

plt.bar(x,y)

plt.xticks(x,x\_t,rotation='vertical')

plt.show()

The above code generates the following output and plot:

{'computer': 1, 'device': 2, 'instructed': 4, 'perform': 8, 'specified': 9, 'instructions': 5, 'used': 11, 'automate': 0, 'manual': 7, 'labor': 6, 'unmatching': 10, 'execution': 3}

# https://lh4.googleusercontent.com/cOBWqPzjlEwpRQx25U7UAIuMmb-JPVEKbgs7rcl2lla-8Zk6toPZQ7-FbhDejlnNPyOm-hLu2oG7BFNvOAV55JjtjZKUK2Z6PWVgZvdK_3zwtrIGj3OfyOoFi0r36FbrAWKaFjaW

**Fig. 12.4. Word Frequency analysis model**

## *Generating Bag-of-Words*

In Text analytics, once the frequency of the words is calculated a bag of words need to be generated for further analysis. In this section, the generation of Bag-of-words is discussed with examples.

When it comes to text analysis, the biggest concern is to represent the textual data with numbers that make sense to the statistical models. One of the basic representations used is the “Bag of Words” model. The main aim of it is to put the corpus of words into a common bag and then count the frequency of each word from the bag. Due to this, it is called the Bag of words (BOW) model. One point that is to be noted in this model is that the context of the words in the corpus is completely ignored while adding them to the bag. It gives some context about the data being considered for analysis. While performing advanced tasks that involve the context of the words, the more powerful representations like word embeddings are used.

A bag of words model is built in two phases :

* First, a list of words is generated from the corpus.
* Then, for each word in the corpus, we generate the frequency of the occurrence of that word in the trained corpus.

The following corpus of information is based on Artificial Intelligence and is taken from Wikipedia

*…………………….An artificial brain (or artificial mind) is software and hardware with cognitive abilities similar to those of the animal or human brain. Research investigating "artificial brains" and brain emulation plays three important roles in science:An ongoing attempt by neuroscientists to understand how the human brain works, known as cognitive neuroscience. …………………………*

The following code demonstrates the use of Bag of words from scikit-learn package of python. Initially, the text data is loaded from using the file matter.txt from the wikipedia. The data is cleaned and then used to fit the estimator to the data. Finally, the transform method is used to get the required representation from the count matrix.

*#import the modules required.*

from nltk.corpus import stopwords

import string

from sklearn.feature\_extraction.text import CountVectorizer

from nltk.tokenize import sent\_tokenize,word\_tokenize

*#Load the corpus from a text file and tokenize it into sentences.*

with open('matter.txt','r') as f:

    data = f.read()

Text = sent\_tokenize(data)

*# Total number of sentences in the data. Prints 14 for this text.*

print(len(Text))

#*Define the preprocessor routine for the data.*

def preprocess(sentence):

    sentence=sentence.lower()

    sentence="".join([x for x in sentence if x not in string.punctuation])

    sentence=[x for x in sentence.split(" ")

if x not in stopwords.words('english')]

    sentence=[x for x in sentence if x!='']

    return " ".join(sentence)

*# Fit a bag of words estimator and transform the count matrix.*

bow\_vectorizer = CountVectorizer(lowercase=True,preprocessor=preprocess)

model = bow\_vectorizer.fit(Text)

bag\_of\_words=model.transform(Text)

*#Get the frequencies of the words.*

bow = bag\_of\_words.todense()

#Get the words in the corpus.

words = model.get\_feature\_names()

*#See the details of the estimator values.*

print(bow.shape)   *# prints (14, 159)*

print(len(words))  *# prints 159*

In the above code, a bag of words vector is generated for every sentence. In the last lines of the code, the shape of the BOW vector can be seen. It is a 2D matrix of shape 14 X 159.  Here, 14 indicates the sentences in the corpus. Whereas 159 indicates the total number of words in the corpus.

**Example on BOW**

A simpler example to understand how BOW works is considered in this section. Consider the following sentences about computers:

*Computer is a device that can be instructed to perform specified instructions.*

*Computer is used to automate manual labor through unmatching instructions execution.*

On applying the bag of words model, there are 12 words in the two sentence corpus described above. Once the frequency of each word in the corpus is computed, following BOW vectors obtained as shown in the table 12.5.

**Table. 12.5. BOW vector for computer example**

|  |  |  |
| --- | --- | --- |
| **Words** | **Ssentence 1** | **Sentence 2** |
| automate | 0 | 1 |
| computer | 1 | 1 |
| device | 1 | 0 |
| execution | 0 | 1 |
| instructed | 1 | 0 |
| instructions | 1 | 1 |
| labor | 0 | 1 |
| manual | 0 | 1 |
| perform | 1 | 0 |
| specified | 1 | 0 |
| unmatching | 0 | 1 |
| used | 0 | 1 |

The above table 12.5 describes the BOW vectors for the provided corpus. Each column of the vector represents the frequency of the word in the document or sentence provided. In the table 12.5 of the BOW vector, most of the entries will be zeroes since represent all the words in the corpus are represented in each vector. This representation becomes quite unrealistic if the number of words in the data becomes large. For example  if we take a corpus of 1000 sentences and 5000 words, we would get a vector of 1000 X 5000 entries which would need  2GB of memory which would be very hardware intensive task. In order to reduce the memory overhead, the sparse matrix representation is used. Sparse matrix representation is that we store the indices of the entries whose values are non-zero.

The following code represents the generation of BOW vector for another corpus of text. In this way, the BOW vector model can be built on the given corpus information of text.

**from** nltk.corpus **import** stopwords  
**import** string  
**from** sklearn.feature\_extraction.text **import** CountVectorizer

*#load dataset*Text = [  
 **"PretzelBros, airbnb for people who like pretzels, raises $2 million"**,  
 **"Top 10 reasons why Go is better than whatever language you use."**,  
 **"Why working at apple stole my soul (I still love it though)"**,  
 **"80 things I think you should do immediately if you use python."**,  
 **"Show HN: carjack.me -- Uber meets GTA"**]

**def** preprocess(sentence):  
 sentence=sentence.lower()  
 sentence=**""**.join([x **for** x **in** sentence **if** x **not in** string.punctuation])  
 sentence=[x **for** x **in** sentence.split(**" "**) **if** x **not in** stopwords.words(**'english'**)]  
 sentence=[x **for** x **in** sentence **if** x!=**''**]  
 **return " "**.join(sentence)

bog\_vectorizer = CountVectorizer(lowercase=**True**,preprocessor=preprocess)  
model = bog\_vectorizer.fit(Text)

bag\_of\_words=model.transform(Text)  
print(bag\_of\_words.todense())  
print(model.get\_feature\_names())

**Output:**

['10', '80', 'airbnb', 'apple', 'better', 'carjackme', 'go', 'gta', 'hn', 'immediately', 'language', 'like', 'love', 'meets', 'million', 'people', 'pretzelbros', 'pretzels', 'python', 'raises', 'reasons', 'show', 'soul', 'still', 'stole', 'things', 'think', 'though', 'top', 'uber', 'use', 'whatever', 'working']

## *Word to Vector model*

In the pre-processing of text analytics, once the bag of words and word cloud is created, it becomes easy for further processing. One of the alternative ways to create a word cloud model is using a word to vector model. In this section, word to vector model is presented with visualization.

The basic procedure of getting a vector model using the words in the text is to first preprocess the sentences using the nltk module in python. The following code demonstrates creating a word to vector model for text analytics. The corpus considered for the example is the inbuilt *abc* of the *sklearn* module. The corpus *abc* consists of random text where the sentences are extracted first and then tokenized into words.

Once the tokenized words are obtained, the stop words are removed from the list and a final word list is formed for further text analytics. The output of the tokenized words for the *abc* corpus is as shown below. It is a nested list where each list represents the words in the sentence. In this way, first the token of words are obtained for each sentence in the corpus.

**from** nltk.corpus **import** abc,stopwords  
**from** string **import** punctuation  
**from** gensim.models **import** Word2Vec  
**from** sklearn.manifold **import** TSNE  
**import** pandas **as** pd  
**import** matplotlib.pyplot **as** plt

sents = abc.sents()  
*#print(sents[:10])*puncs = list(punctuation)  
stop = set(stopwords.words(**'english'**) + puncs + [**"''"** , **"``"**])  
processed\_sents = []  
**for** sent **in** sents:  
 temp = []  
 **for** word **in** sent:  
 **if** word **not in** stop:  
 temp.append(word.lower())  
 processed\_sents.append(temp)  
print(processed\_sents[:10])

[['pm', 'denies', 'knowledge', 'awb', 'kickbacks', 'the', 'prime', 'minister', 'denied', 'knew', 'awb', 'paying', 'kickbacks', 'iraq', 'despite', 'writing', 'wheat', 'exporter', 'asking', 'kept', 'fully', 'informed', 'iraq', 'wheat', 'sales'], ['letters', 'john', 'howard', 'deputy', 'prime', 'minister', 'mark', 'vaile', 'awb', 'released', 'cole', 'inquiry', 'oil', 'food', 'program'], ['in', 'one', 'letters', 'mr', 'howard', 'asks', 'awb', 'managing', 'director', 'andrew', 'lindberg', 'remain', 'close', 'contact', 'government', 'iraq', 'wheat', 'sales'], ['the', 'opposition', 'gavan', 'o', 'connor', 'says', 'letter', 'sent', '2002', 'time', 'awb', 'paying', 'kickbacks', 'iraq', 'though', 'jordanian', 'trucking', 'company'], ['he', 'says', 'government', 'longer', 'wipe', 'hands', 'illicit', 'payments', 'totalled', '290', 'million'], ['the', 'responsibility', 'must', 'lay', 'may', 'squarely', 'feet', 'coalition', 'ministers', 'trade', 'agriculture', 'prime', 'minister', ',"', 'said'], ['but', 'prime', 'minister', 'says', 'letters', 'show', 'inquiring', 'future', 'wheat', 'sales', 'iraq', 'prove', 'government', 'knew', 'payments'], ['it', 'would', 'astonishing', '2002', 'prime', 'minister', 'i', 'done', 'anything', 'i', 'possibly', 'could', 'preserve', 'australia', 'valuable', 'wheat', 'market', ',"', 'said'], ['email', 'questions', 'today', 'inquiry', 'awb', 'trading', 'manager', 'peter', 'geary', 'questioned', 'email', 'received', 'may', '2000'], ['it', 'indicated', 'iraqi', 'grains', 'board', 'approached', 'awb', 'provide', 'sales', 'service', '".']]

For each of the processed sentences initially now the vector model is obtained using the function *Word2Vec*. The parameters passed to this function are *size, min\_count, workers and iter*. Here, the size is specified as 300 indicating the number of sentences in the corpus considered. The min\_count refers to the total number of words in the sentence. If the number of words in the sentence is 20, then the word to vector model is obtained.

Here, for interpreting the results, the word *government* is used. All the words in the corpus that are most similar to these words are added to the embeddings to the get the word vector model for it. It can be seen in the output that the words *court, governments, federal, industry, opposition* are taken into the word vector model with their occurrence of frequencies.

embeddings = Word2Vec(sentences=processed\_sents,size=300,min\_count=20,workers=4,sg=0,iter=5,hs=0)  
print(embeddings.wv.most\_similar(**'government'**))

[('court', 0.8545933365821838), ('governments', 0.7608055472373962), ('federal', 0.7528774738311768), ('industry', 0.7445326447486877), ('the', 0.7218483686447144), ('opposition', 0.7107964158058167), ('funding', 0.6966238021850586), ('inquiry', 0.6842386722564697), ('review', 0.6813172698020935), ('trade', 0.6796253323554993)]

Next, the words obtained in the embedding list are collected as one list called as *vocabulary.* The function *TSNE()* is used to obtain the word vector model for all the words in the corpus. It gives the distributed stochastic neighbor embedding model for the words in the corpus considered. The words that are similar are grouped into one and stored in the vector. The output of the TSNE is as shown below where the x and y values can be used to see position of the words in the corpus.

vocab = list(embeddings.wv.vocab)  
X = embeddings[vocab]  
tsne\_model = TSNE(n\_components=2)  
X\_tsne = tsne\_model.fit\_transform(X)

data = pd.DataFrame(X\_tsne, index=vocab, columns=[**'x'**, **'y'**])  
data = data[:100] *# use only first 100 words.*print(data)

x y

denies 41.000854 -38.010197

knowledge -1.009654 11.519361

awb 33.187195 41.710598

kickbacks 34.419754 -36.837307

the 35.065098 38.840485

prime 50.779877 30.259434

minister 52.725185 38.467823

knew -5.418348 -57.856190

paying 5.399355 -14.157794

iraq 18.697643 35.002403

despite 23.604540 37.188904

writing 32.363739 -21.261209

wheat 29.105671 42.581383

exporter 29.166904 39.915119

asking -33.615059 12.525192

kept -25.664696 -2.553294

fully -16.702623 -36.173401

sales 0.915510 42.332363

letters 41.420280 1.516451

john 52.070244 37.122097

howard 49.365101 28.190823

deputy 43.894024 25.362448

mark 49.109886 31.493637

vaile 49.242855 27.371328

released 31.614101 35.871021

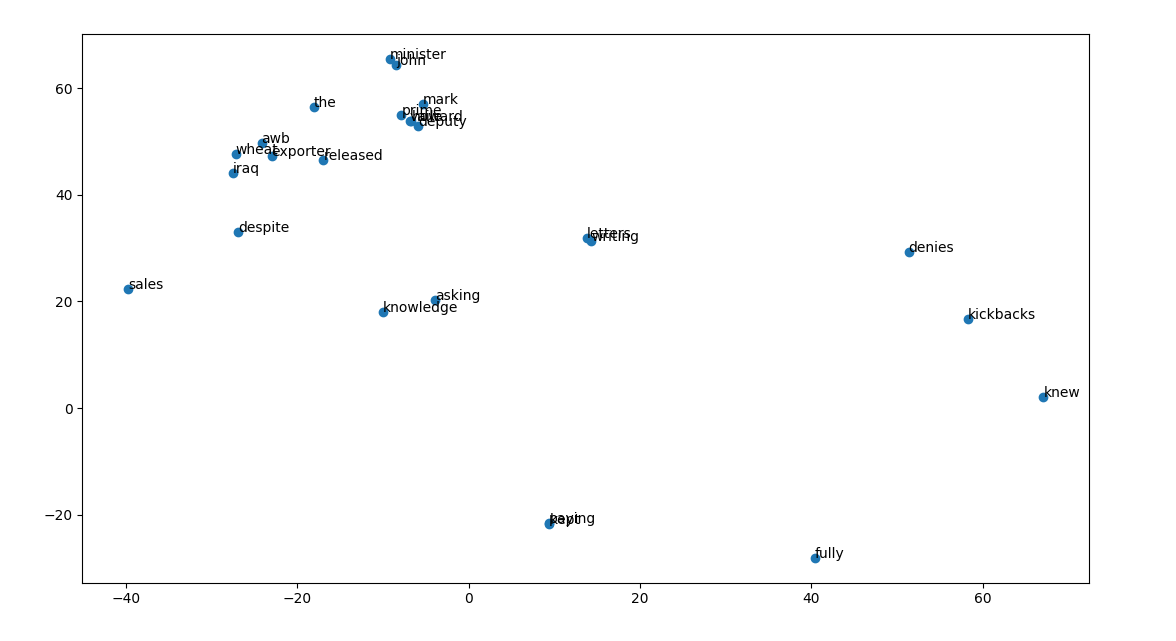
cole 25.016020 31.850077

inquiry 33.530052 41.403614

oil 22.474443 33.175205

food 22.632797 33.002705

program 32.533459 18.267725



**Fig. 12.6. Word to vector model**

The figure 12.6 represents the word to vector model for the code in python. It can be observed from the figure 12.6 that the words *released, exporter* are grouped closely than the other words like *denies, knew* in the word to vector model. The model shown in the figure 12.6 is for only 25 words in the corpus. For the entire corpus, the visualization needs to be divided into subsets.

In this section, the example discussed demonstrated the usage of word to vecto model. The word cloud model for applications like spam classification, sentimental analysis is discussed in the next section.

## *Word cloud model*

Word clouds commonly referred to as tag clouds represent graphically the word frequency appearing in a text file. It gives prominence to the words that appear most frequently in the text. Larger the font size in the word cloud, more frequently the word appears in the text. Visualization of content of a file using word cloud helps in identifying a set of most frequent words in major documents like interviews, formal reports etc. The main aim of word cloud is it can be used to communicate the thematic points of the text.

Various applications such as Spam classification, automatic summarization include word cloud as the beginning phase for exploration of the data. For example, in the case of spam classification building the word cloud model gives the word that occurs most frequently in the text that can be either spam/non-spam. In the same way, reviews of different products for recommendation systems can be used to build a word cloud model for identifying the keywords used in the recommendation. These applications are dealt separately in the upcoming chapters of this part.

### Word cloud model in Python

Word cloud models are very helpful in analyzing the various insights about a document. For example, in the case of a small documentary on a natural disaster like TSUNAMI’, a typical word cloud would be helpful in searching for the meanings of some important words related to Tsunami. Since, various applications can make use of word cloud for analysis in this section a small example on the word cloud model in python is discussed.

The following code demonstrates the word cloud model in python. Initially, the modules required for the word cloud model are imported. The corpus of words from the nltk module is imported for stopwords and tokenization of words. A file consisting of random text is considered here for building the word cloud model. All the punctuation marks are removed from the text in the file as the first step. Each line of the file is read into word cloud for generating the data. The data is then fed into the word cloud module of nltk. The matplotlib module is used to plot the word cloud model as shown in the figure 12.7.1.

In the figure 12.7.1, the words that appear more frequently appear in large font size than the other words in the text file. For example, the word brain and artificial appear large in size than the other words. Another analysis that can be found from the word cloud model is the content of the file is more related to AI, machine, cognitive, human and brain.

**from** wordcloud **import** WordCloud  
**import** matplotlib.pyplot **as** plt  
**from** nltk.tokenize **import** sent\_tokenize,word\_tokenize  
**from** nltk.corpus **import** stopwords  
**from** string **import** punctuation  
**from** nltk.probability **import** FreqDist  
puncs = (list(punctuation))  
puncs.extend([**"'s"**,**"''"**,**"``"**])

**with** open(**'files/matter.txt'**) **as** f:  
 data = f.read()  
wordcloud = WordCloud()  
wordcloud.generate(data)  
plt.imshow(wordcloud, interpolation=**'bilinear'**)  
plt.axis(**"off"**)  
plt.show()

****

**Fig. 12.7.1. Word cloud model**

In the previous code of building the word cloud model, the sentences read from the file were not tokenized into individual words. It will be interesting to see what happens when the sentences are broken down into words and then the word cloud model is built. The following code demonstrates building the word cloud model using tokenization approach.

Once a line is read from the file each word in the line is extracted into tokens using the code below. The sent\_tokenize() is the method used by converting the sentences into lower case. Each word extracted now might be the case of stopwords/punctuations. Hence, first the words are appended to a list and then the stopwords and words with punctuations are removed for generating the data for word cloud. For the final word list, the word cloud model is built using the Wordcloud() module.

**with** open(**'files/matter.txt'**) **as** f:  
 data = f.read()  
sentences = sent\_tokenize(text=data.lower().strip())  
words = []  
**for** i **in** sentences:  
 words.extend(word\_tokenize(i))  
stop = set(stopwords.words(**'english'**) + puncs )  
final\_words = []  
**for** word **in** words:  
 **if** word **not in** stop:  
 final\_words.append(word)  
table = FreqDist(final\_words)  
wordcloud = WordCloud()  
wordcloud.generate\_from\_frequencies(table)  
  
plt.imshow(wordcloud, interpolation=**'bilinear'**)  
plt.axis(**"off"**)  
plt.show()

****

**Fig. 12.7.2. Word cloud model with tokenization**

The output of the word cloud model is as shown in the figure 12.7.2. It can now be observed in the figure 12.7.2 that machine is not highlighted with larger font size as in the previous word cloud model. However, other words artificial, brain and human font sizes remain the same. In this way, the word cloud model can be built efficiently using the tokenization approach.

So far, in the previous sections the basic steps necessary for text analytics like word cloud model, word to vector model, bag of words were discussed with examples. These steps are pre-processing steps that is needed for every text analytics problem that need to be studied. Once, these steps are executed further machine learning techniques can be employed for advanced analytics.

In the upcoming sections, case studies on text analytics such as automatic summarization, spam classification and question classification are discussed with python examples. For these case studies, open datasets are used and hence can be used as examples for simple analysis. However, it can be further extended to larger problems for analysis.

## *Automatic summarization case study*

Natural language processing (NLP) is a domain area in computer science that is related to human and computer interaction. It is now tending towards Artificial Intelligence (AI) and has become the core integral part of AI and automation. The current world of AI in every aspect requires large volume of information and understanding. For example, Speech recognition needs lot of information about the taxonomy and semantics of processing. In this chapter, the focus is on summarization which is one of the key roles of NLP.

Summarization can be defined as the creation of small version of text from large version using automation. The increase in information around the internet has lead to the topic of summarization. When information is required and searched in the internet, voluminous information is presented instead of quick short summary of it [4]. For example, the key term ‘network data analytics’ refers to large amount of information and a search in the internet does not give a short summary that gives the key concepts on it.

A summary of text can be defined as “text that conveys the information about one or more documents and usually half the size of the original document”. The general examples of summarization are news headlines, movie reviews, meeting minutes etc. In the current era of internet world, summary of information plays a key role in defining the qualitative information. In this section, a case study on the summarization is discussed in Python. Before diving into the code, some of the key aspects of summarization are discussed as below.

### Supervised and Unsupervised summarization

In supervised method, the summarization of text is obtained from the training text data set. The corpus of the information includes information elements like dictionary and grammar text. The basic approach followed in the supervised approach is certain key phrases are characterized and ranked. Summary of information is obtained based on these key phrases in the testing document. The drawback of this approach is supervised methods are domain specific. For example, if the training set includes corpus of information on scientific discoveries summary will be concentrated on it only. When other text based on movies, news are given summary obtained may not be accurate.

In the case of unsupervised method, statistical techniques are used for obtaining the summary of the text. The content of the text is divided into small pieces of information where each chunk of the information is ranked according to a weight function matrix. Based on the ranking of the contents, the information of the summary is obtained. The drawback of the domain basis in supervised approach is eliminated here using ranking approach. Since, each time the corpus of information is built around the different information in the text it is domain independent.

### Abstract and Extractive summarization

The abstraction of information from various documents is a complex approach. For example, if we read an article in a newspaper and rewrite an abstract for the same, sometimes the context and the meaning might not appear right. In the same way when a machine learning model need to be used for summarization it involves a set of steps with complex process. The abstractive way of summarization searches for the general phrases in the text and a lexical model is built first for summarization. The summary of the text is obtained from the lexical chain of information of the text. For example, in a compiler lexical analysis is performed to identify the errors in a program and point to the line of error.

In extractive method of summarization, a subset of words, sentences and paragraphs are taken from the original text to form the summary of the text. Some of the typical phases of extractive summarization methods are tokenization, sentence extraction, ranking of sentences and then finally the summary. The sub-steps of the extractive summarization method are integrated one-by-one for forming a summary. For example, the words that are extracted in the phase of tokenization are used in sentence extraction for forming the summary.

### Sentence extraction

The different types of summary as discussed in the previous section mainly involve sentence extraction as the basic entity for summarization. In this step of summarization, the sentence extracted need to be scored and ranked for generating the summary. The key concepts that are involved in sentence extraction are as follows:

* **Keyword-occurrence:** Certain keywords in the sentences highlight the importance of the sentence and thus can be used for scoring and ranking. For example, words such as ‘because’, ’hence’, ’therefore’ are some of the keywords that mark the sentence as important and can be used for summarization.
* **Location-heuristic:** In certain domain-specific documents, the location of the documents represented by the headings can be used for finding useful information about the text. Generally, abstract and conclusion are the key locations where most of the information is present and the same can be used for summary.
* **Pronouns:** The sentences with more pronouns are not used as a part of the summary as generally it involves redundant information. For example, consider the text “Data Analytics is a process of analyzing data with machine learning methods. It involves various steps in arriving the final results”, where the second sentence with pronoun can be excluded in the finally summary.
* **Scoring:** The sentences with the key phrases are scored according to the corpus information available in the text. Once the summary is generated using the scoring approach, then if the summary is not satisfied rescoring need to be done once again for the summary.

With the prerequisites of different methods of summarization, auto summarization in python is presented in the next section.

### Auto Summarization in Python

The following code demonstrates auto summarization in Python using extractive technique of summarization and nltk module.

Firstly, the modules required for summarization are snt\_tokenize, stopwords, punctuation and FreqDist. These modules are imported first using the nltk library. In the text, punctuations need to be removed and thus a list of punctuations are imported from the inbuilt module of python and initialized into a variable puncs. The file with the text to be summarized is opened and the lines are read. Each line read from the file are converted to lower case and tokenized into words.

All the words extracted from the sentence are appended to a list of words. From this list of words, stopwords need to be removed. The stopwords are removed using the words method and the parameter passed to it is ‘english’. The final word list is prepared by comparing the ‘words’ list and ‘stop’ list. This list is used for generating the summary of a text.

In the next phase, a frequency distribution of the final words is prepared for ranking the sentences. A table of sorted words with their frequencies is prepared. Using this table, each sentence in the extracted text is compared. If the sentence contains the word, then rank is increased. In this way a dictionary ‘sent\_ranks’ is prepared that contains the rank for each sentence.

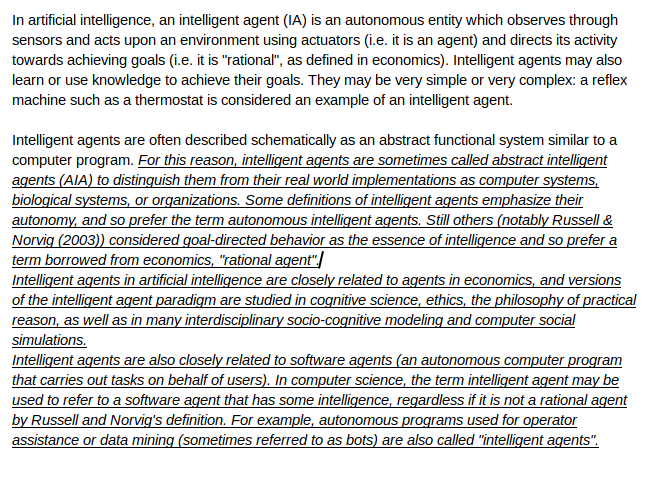
The last phase contains the code for generating summary. The dictionary ‘sent\_ranks’ is first sorted based on the items. The dictionary is then reversed to get the descending order of sentences i.e. highest ranked sentence is at the beginning of the dictionary. For each sentence in the dictionary that is ranked first, sentences are extracted one-by-one for generating the summary.

**from** nltk.tokenize **import** sent\_tokenize,word\_tokenize  
**from** nltk.corpus **import** stopwords  
**from** string **import** punctuation  
**from** nltk.probability **import** FreqDist  
puncs = (list(punctuation))**with** open(**'matter.txt'**,**'r'**) **as** f:  
 content = f.read()  
sentences = sent\_tokenize(text=content.lower())words = []  
**for** i **in** sentences:  
 words.extend(word\_tokenize(i))stop = set(stopwords.words(**'english'**) + puncs )  
final\_words = []  
**for** word **in** words:  
 **if** word **not in** stop:  
 final\_words.append(word)table = FreqDist(final\_words)  
ranked\_words = sorted(table,key=table.get)  
sent\_ranks = {}**for** sent **in** sentences:  
 w = word\_tokenize(sent)rank = 0  
 **for** word **in** w:  
 **if** word **in** ranked\_words:  
 rank = rank + ranked\_words.index(word)  
 sent\_ranks[rank] = sent   
final\_sents = sorted(sent\_ranks.items())  
final\_sents.reverse()  
final = []  
**for** item **in** final\_sents[0:10]:  
 final.append(sentences.index(item[1]))

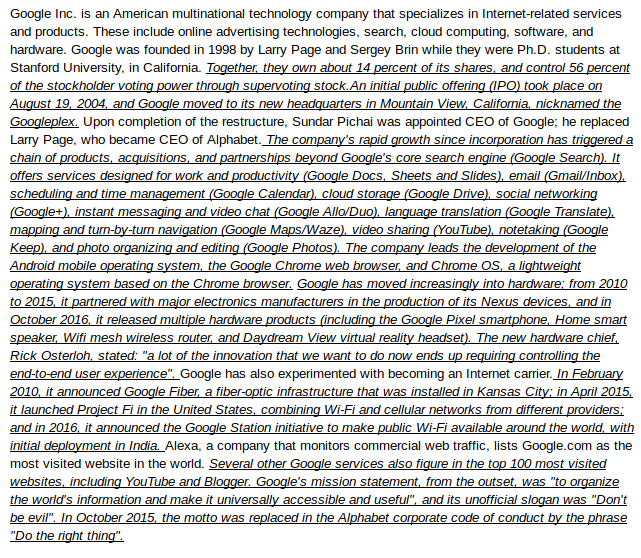
**for** index **in** sorted(final):  
 print(sentences[index] + **'\n'**)

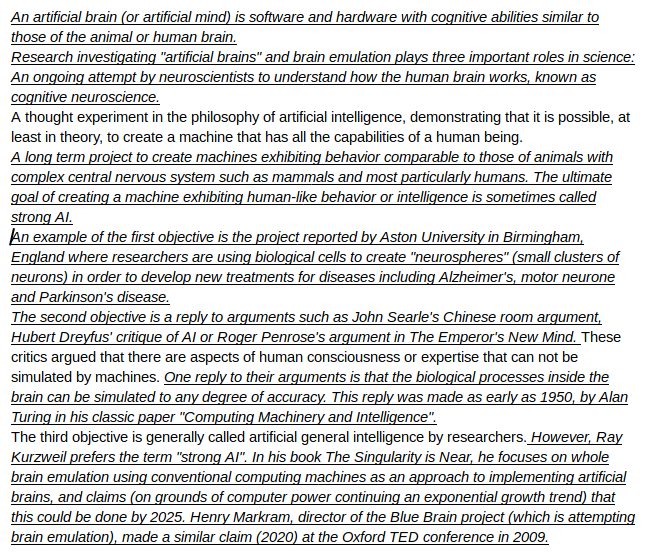
The outputs of the summarization are as shown in the following figures 12.8.4.1, 12.8.4.2, 12.8.4.3. In the plots, the underlined sentences are selected for the summarization. Figure 12.8.4.1 shows the summarization of AI corpus information. Figure 12.8.4.2 shows the summarization of google corpus information. Figure 12.8.4.3 shows the summarization of brain corpus information. It can be seen clearly from the plots that specific sentences are picked up for the summarization which are underlined. In this way, automatic summarizations can be obtained for other corpus information of the text using Python.

**Output of the summarization**



**Fig. 12.8.4.1. Automatic summarization for AI corpus**



**Fig. 12.8.4.2. Automatic summarization for google corpus** 

**Fig. 12.8.4.3. Automatic summarization for brain corpus**

## *Spam classification case study*

The data that revolves around the internet is generally unstructured. The sources of such data are images, email, audio and video. Algorithms that scrape and analyze on such data need to have more capabilities in understanding the domain. For example, consider a mailbox where there is a lot of mails and if one has to segregate manually into spam/non-spam. It becomes a tedious process for humans to carry out such process. Machine learning methods of classification can be employed here for analysis of spam/non-spam.

The basic methods of spam classification can be categorized into two approaches namely content based and non-content based. In the content based approaches, text classification methods such as clustering, SVM, logistic regression are employed. The contents are first classified as spam/non-spam first and then the actual spam classification is carried out. The basic examples of classification methods were discussed as a part of this book in Part 2. In the non-content based approaches, the contents are not available in hand for analysis. For example, in the case of social networks since the content is unstructured, the information cannot be categorized into spam/non-spam easily. It requires many steps for classification.

In this section, a spam filter is constructed for a dataset [7]. A spam filter is constructed with Naïve Bayes, SVM and neural network methods. Each of methods for classification is discussed with results. A comparison among all the three methods is finally made in the end where neural network outperform all the other methods. The main aim of this section is to get familiarize with the machine learning methods for spam classification.

### Spam classification with Naïve Bayes in Python

In this section, classification model for identifying spam/non-spam messages is discussed with an example in python [5] [6]. Initially, as explained in the earlier sections the text need to be preprocessed first by removing the stopwords and building the word cloud. Firstly, the word cloud in python is built and then the spam filter is modeled. The dataset considered for the spam classification from [7].

### Spam word cloud in Python

The word cloud for the dataset considered is implemented before pre-processing the text and after pre-processing. This gives a better clarity of pre-processing and word cloud. In the first section, the code for implementing word cloud before pre-processing is presented and then the word cloud is presented for the after pre-processing part.

### Word cloud before pre-processing

In the following code, the modules of nltk, WordCloud, TfidfVectorizer, Stemmer and GaussianNB are imported first. The dataset from the file *spam.csv* is loaded and the unwanted columns are removed. These columns are ignored for building the word cloud as they does not signify any important meaning.

The word cloud is built for the sentences that are identified as spam in the training dataset using the spam text. In the same way the word cloud is built for the sentences that are identified as ham in the training set. The word cloud model in the figure 12.9.2.1.1 is for spam data points and the word cloud model in the figure 12.9.2.1.2 is for ham data points.

In this section of the code, the text from the dataset was not preprocessed i.e. stopwords and punctuations were not removed. So, it can be seen from the word cloud model of spam data points in the figure 12.9.2.1.1that some of the stopwords such as will, please are present. These words does not have a significant meaning and thus preprocessing need to be done.

**from** nltk.corpus **import** stopwords  
**from** sklearn.feature\_extraction.text **import** TfidfVectorizer  
**import** string  
**import** pandas **as** pd  
**from** nltk.stem.porter **import** PorterStemmer  
**from** nltk.stem **import** SnowballStemmer  
**from** sklearn.model\_selection **import** train\_test\_split  
**from** sklearn.naive\_bayes **import** GaussianNB  
**from** sklearn.metrics **import** accuracy\_score,confusion\_matrix  
**import** matplotlib.pyplot **as** plt  
**from** wordcloud **import** WordCloud

dataset=pd.read\_csv(**"spam.csv"**,encoding=**'latin'**)  
dataset = dataset.drop([**"Unnamed: 2"**, **"Unnamed: 3"**, **"Unnamed: 4"**],axis=1)

x=dataset.copy()  
spam=x[x.v1==**"spam"**]  
spam=spam.v2  
spam\_text=**"."**.join(spam)  
wordcloud\_spam = WordCloud().generate(spam\_text)  
plt.imshow(wordcloud\_spam)  
plt.axis(**"off"**)  
print(**"The spam word cloud is:-"**)  
plt.show()

ham=x[x.v1==**"ham"**]  
ham=ham.v2  
ham\_text=**"."**.join(ham)  
wordcloud\_ham = WordCloud().generate(ham\_text)  
plt.imshow(wordcloud\_ham)  
plt.axis(**"off"**)  
print(**"The not spam word cloud is:-"**)  
plt.show()



**Fig. 12.9.2.1.1. Word cloud model before pre-processing for spam datapoints**



**Fig. 12.9.2.1.2. Word cloud model before pre-processing for ham datapoints**

### Word cloud after pre-processing

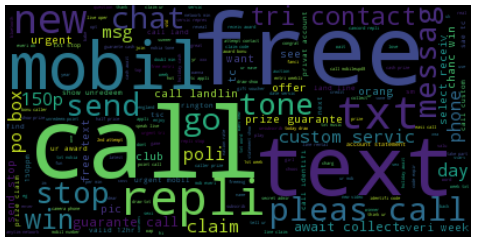
In this section, the code first pre processes the text considered in the previous section by removing the stopwords and punctuation. The word cloud models for spam and ham data points are modeled using the same approach as seen in the earlier section. The PortStemmer() method from the nltk module is used for stemming the words in the text. For example, the words fishing, fisher are reduced to the root word ‘fish’

The word cloud model after preprocessing with spam and ham data points are as shown in the figure 12.9.2.2.1 and figure 12.9.2.2.2. From the figure 12.9.2.2.1 it can be observed that the stopwords are not present in the word cloud model. Once the pre-processing is done, the word cloud appears more clear than the one without pre-processing. We can see from the figure 12.9.2.2.1 of spam data points where the words *‘call’* and *‘free’* are highlighted with greater font size. These are the generally used phrases that identify the message as spam. Hence, using the word cloud model these types of certain phrases identifying as spam can be seen.

**def** preprocess(sentence):  
 stemmer=PorterStemmer()  
 sentence=sentence.lower()  
 sentence=**""**.join([x **for** x **in** sentence **if** x **not in** string.punctuation])  
 sentence=[x **for** x **in** sentence.split(**" "**) **if** x **not in** stopwords.words(**'english'**)]  
 sentence=[x **for** x **in** sentence **if** x!=**''**]  
 sentence=[stemmer.stem(x) **for** x **in** sentence]  
 **return " "**.join(sentence)

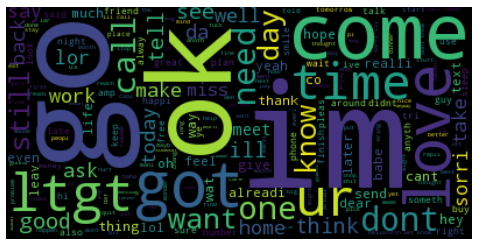
content=dataset[**'v2'**].copy()  
content=content.apply(preprocess)

x=dataset.copy()  
spam=x[x.v1==**"spam"**]  
spam=spam.v2  
spam=spam.apply(preprocess)  
spam\_text=**"."**.join(spam)  
wordcloud\_spam = WordCloud().generate(spam\_text)  
plt.imshow(wordcloud\_spam)  
plt.axis(**"off"**)  
print(**"The spam word cloud is:-"**)  
plt.show()



**Fig. 12.9.2.2.1. Word cloud model after pre-processing for spam datapoints**

ham=x[x.v1==**"ham"**]  
ham=ham.v2  
ham=ham.apply(preprocess)  
ham\_text=**"."**.join(ham)  
wordcloud\_ham = WordCloud().generate(ham\_text)  
plt.imshow(wordcloud\_ham)  
plt.axis(**"off"**)  
print(**"The not spam word cloud is:-"**)  
plt.show()



**Fig. 12.9.2.2.2. Word cloud model after pre-processing for ham datapoints**

### Spam filter in Python

In this section a spam filter is modeled for the dataset considered. The following code demonstrates the spam filter in Python using Naïve Bayes classification method. Initially, the modules in nltk, Portstemmer, Snowballstemmer and GaussianNB are loaded. The csv file *‘spam.csv’* is loaded as the main text file for classification. From the dataset the columns 2,3 and 4 are dropped as it is not needed for classification. The first ten rows of the data is as shown in the figure 12.9.3.1.

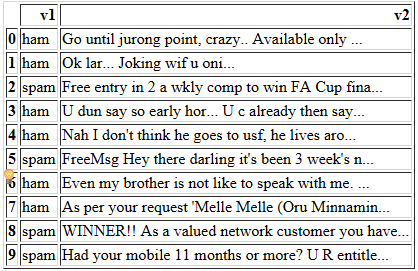
The sentences in the third column from the dataset are extracted for classification. Initially, the PortStemmer() module is used for stemming the sentences with whitespace and other characters. The sentence is then converted into lower case and then the punctuation and stop words are removed. The sentences are then rejoined back for training the dataset for classification.

**from** nltk.corpus **import** stopwords  
**from** sklearn.feature\_extraction.text **import** TfidfVectorizer  
**import** string  
**import** pandas **as** pd  
**from** nltk.stem.porter **import** PorterStemmer  
**from** nltk.stem **import** SnowballStemmer  
**from** sklearn.model\_selection **import** train\_test\_split  
**from** sklearn.naive\_bayes **import** GaussianNB  
**from** sklearn.metrics **import** accuracy\_score,confusion\_matrix

dataset=pd.read\_csv(**"spam.csv"**,encoding=**'latin'**)  
dataset = dataset.drop([**"Unnamed: 2"**, **"Unnamed: 3"**, **"Unnamed: 4"**],axis=1)  
dataset.head(10)

**def** preprocess(sentence):  
 stemmer=PorterStemmer()  
 sentence=sentence.lower()  
 sentence=**""**.join([x **for** x **in** sentence **if** x **not in** string.punctuation])  
 sentence=[x **for** x **in** sentence.split(**" "**) **if** x **not in** stopwords.words(**'english'**)]  
 sentence=[x **for** x **in** sentence **if** x!=**''**]  
 sentence=[stemmer.stem(x) **for** x **in** sentence]  
 **return " "**.join(sentence)

content=dataset[**'v2'**].copy()  
content=content.apply(preprocess)

****

**Fig. 12.9.3.1. Dataset for spam classification**

tfidf\_vectorizer = TfidfVectorizer(**"english"**)  
tfidf\_vectorizer.fit(content)  
features = tfidf\_vectorizer.transform(content)  
features = features.todense()  
features\_train, features\_test, labels\_train, labels\_test = train\_test\_split(features, dataset[**'v1'**], test\_size=0.3,shuffle=**True**)

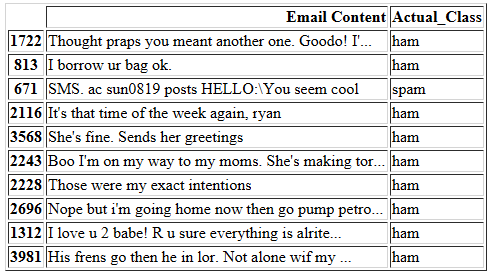
model=GaussianNB()  
model.fit(features\_train,labels\_train)

**GaussianNB(priors=None)**

test=dataset.sample(10).copy()  
test\_features=test[**'v2'**]  
test\_lables=test[**'v1'**]  
test.rename(columns={**'v1'**:**'Actual\_Class'**,**'v2'**:**'Email Content'**},inplace=**True**)  
test=test.reindex\_axis([**'Email Content'**,**'Actual\_Class'**],axis=1)  
test

The joined sentences are applied for preprocessing and the term frequency (TF) and document frequency (DF) is calculated. The nltk module of the vectorizer itself is used for this purpose and then features for spam classification are extracted. The pre-processed dataset with no stopwords and punctuations are split into training and testing for classification. The test data set size is set to 0.3 here.

A naïve bayes classification model is built using the GaussianNB() function. In order to see classification, a sample rows is selected from the dataset with actual class as seen in the figure 12.9.3.2. The preprocessed test features are applied to the model with TF and DF for classification. The predicted class of the dataset is as shown in the figure 12.9.3.3. In the figure 12.9.3.3 only sample rows are selected for viewing the output. The actual class and predicted class values are equal. But it does not mean the same for all the rows in the dataset. Hence, the accuracy score and confusion matrix is printed out. We can see the accuracy score is 88% and the confusion matrix represents that

****

**Fig. 12.9.3.2. Dataset of spam classification with actual class**

test\_features=test\_features.apply(preprocess)  
test\_features=tfidf\_vectorizer.transform(test\_features)  
test\_features=test\_features.todense()  
model.predict(test\_features)  
test[**'Predicted\_Class'**]=model.predict(test\_features)  
test

print(**"The confusion matrix:-\n"**,confusion\_matrix(labels\_test,model.predict(features\_test)))  
print(**"accuracy "**,accuracy\_score(labels\_test,model.predict(features\_test)))

The confusion matrix:-

[[1284 163]

[ 28 197]]

accuracy 0.885765550239

## 

**Fig. 12.9.3.3. Dataset of spam classification with predicted class**

In this section, spam classification was carried out using Naïve Bayes method and the accuracy was 89%. The confusion matrix shows that 1284 data points are correctly classified as ham whereas 163 are incorrectly classified as 163. In the next section, the same dataset is considered for spam classification using SVM method.

### Spam classification with SVM

SVM machine learning technique was discussed in the in Part 2 of this book. SVM technique is used for classification of datasets where multiple classes are involved. In this regard, spam classification is carried out using SVM method. Even tough Spam classification appears as a binary classification problem i.e. identifying whether it is spam/non-spam, SVM gives a better accuracy over the other machine learning techniques.

The same dataset considered in the earlier section is used. The procedure for classification remains the same except the model used. Initially, the pre-processing of the text is carried out using the stemmer, stopwords and tokenizer modules. A linear support vector classification is used as a classifier model for this task. In addition to its computational efficiency, another advantage of this approach is interpretability. Since each class is represented by one and one classifier only, it is possible to gain knowledge about the class by inspecting its corresponding classifier.

**import** nltk  
**from** nltk.tokenize **import** word\_tokenize  
**import** numpy **as** np  
**import** random  
**import** pandas **as** pd  
**from** nltk.corpus **import** stopwords  
**from** nltk.stem **import** WordNetLemmatizer   
**import** re  
**from** sklearn **import** svm  
**from** sklearn.model\_selection **import** train\_test\_split  
**from** sklearn.metrics **import** accuracy\_score  
**from** sklearn.feature\_extraction.text **import** CountVectorizer  
vectorizer=CountVectorizer(analyzer = **"word"**,tokenizer = **None**,preprocessor = **None**,stop\_words = **None**,max\_features = 5000)  
lemmatizer = WordNetLemmatizer()  
stop\_words = set(stopwords.words(**'english'**))  
  
df=pd.read\_csv(**"spam.csv"**,encoding=**'latin-1'**) *#Read in the data into a dataframe*df=df[[**'v1'**,**'v2'**]]  
df.columns=[**'label'**,**'sms'**]  
  
*#Function to split the text messages into into words***def** create\_lexicon(sent):  
 sent=re.sub(**"[^a-zA-Z]"**,**" "**,sent)  
 sent=sent.lower()  
 all\_words = word\_tokenize(sent)  
 lex=list(all\_words)  
 lex = [lemmatizer.lemmatize(i) **for** i **in** lex] *#Converting each word to its root word* lex = [w **for** w **in** lex **if not** w **in** stop\_words] *#Removing stop words* lex = [w **for** w **in** lex **if**(len(w)>1)] *#Removing single letter words* **return** (**" "**.join(lex))  
  
rows,col=df.shape  
  
split\_msg=[] *#Creating new column in the dataframe***for** i **in** range(0,rows): *#which contains the lemmatized words of* broken=create\_lexicon(df[**'sms'**][i]) *#each message with the stop words removed* split\_msg.append(broken)  
df[**'split\_msg'**]=split\_msg  
  
*#Splitting data into training and testing set, 80% training data and 20% testing data*X\_train,X\_test,y\_train,y\_test=train\_test\_split(df[**'split\_msg'**],df[**'label'**],test\_size=0.2)  
  
  
train\_data=vectorizer.fit\_transform(X\_train)  
train\_data=train\_data.toarray()  
  
test\_data=vectorizer.transform(X\_test)  
test\_data=test\_data.toarray()  
  
model=svm.SVC(kernel=**'linear'**)  
model.fit(train\_data,y\_train)  
  
*#Testing the model*predicted = model.predict(test\_data)  
print (**"Accuracy"**)  
print (accuracy\_score(y\_test, predicted))

**Table. 12.9.4. Accuracy scores of spam classification**

|  |  |
| --- | --- |
| **Method** | **Accuracy score** |
| Naïve Bayes | 88% |
| SVM | 95% |

It can be seen from the table 12.9.4 the accuracy scores of the each of the method for spam classification case study.The results show that the classification accuracy is more with SVM than the Naïve Bayes classification method. This is because Naive Bayes classification has many drawbacks.

The important drawbacks of Naïve Bayes classification method are listed as follows.

* Naïve Bayes is based on the assumption that data distribution is correct for any two independent features given in the data set.
* If there are no class labels for the given input dataset then the prediction is not accurate. For example, if there is a class as *‘Play=No’* and no instances are present in the training data then for another feature that uses *Play* prior probability, the conditional probability will be 0.
* Naïve Bayes classification will be difficult for continuous variables of the dataset. In that case, the values need to be converted into discrete set using the probability density functions.

The advantages of SVM over Naïve Bayes are listed as follows.

* SVM avoids overfitting the data and helps to generalize the model in a better way.
* It can be used for real-world problems such as text and image classification, hand-writing recognition, and bioinformatics and biosequence analysis.
* There is no upper limit on the number of attributes.

Since, SVM outperforms Naïve Bayes in the listed points above there is a better accuracy with SVMs in the spam classification case study. In this section of text analytics on the spam classification case study initially, the bag of words, word cloud model were created and then the classifiers was modeled for the problem. In this modeling classification, SVM outperformed Naïve Bayes classification method as it has advantages over Naïve Bayes.

## *Question classification case study*

In this section, a case study is discussed on the question classification. The dataset considered here consists of a list of question and their categories [8] [9]. It is as shown in the table 12.10.. It consists of only fee questions in the dataset. As seen in the table 12.10, for each row, a question id, question and category is present. The main aim of this case study is to classify the questions based on the category.

The steps followed for the classification of the case study are listed as follows.

* Preprocessing
* Creating features
* Classification using SVM

**Table. 12.10. Question classification dataset**

|  |  |
| --- | --- |
| **Question** | **Category** |
| Who did Arthur H. Bremer try to assassinate on May 15 1972 ? | 1 |
| Doesn't ROS stand for Return on Sales? | 0 |
| How do you buy stocks ? | 3 |
| Doesn't AAOS mean American Academy of Orthopaedic Surgeons? | 0 |
| What was the infamous feat of Germany 's U-2 submarine ? | 4 |
| How fast is a 45Mhz processor ? | 3 |
| Does Ahvaz lie in Iran? | 2 |
| What is the highest amount that can be got by using exactly 5 zeros? | 5 |
| What is the largest lake in North America ? | 4 |
| Is there not a drug test that can detect any drug that is in your system? | 4 |
| What was the Great Britain population from 1699-172 ? | 5 |
| What do I call the sons and daughters of my first cousins ? | 4 |
| Is that lady a scientist? | 1 |
| Is there not a way to prevent someone from seeing my answers on Quora? | 3 |
| Is there any way for a student to get an internship at Microsoft or at Facebook from high-school? | 3 |
| Does CASLPA mean Canadian Association of Speech-Language Pathologists and Audiologists? | 0 |
| Is that boy a celebrity? | 1 |
| What is the highest number that can be got by using exactly 5 ones? | 5 |
| Is there not a way to get Quora (or websites in general) to display with fonts of my own choosing in Chrome on Linux? | 3 |
| What does the acronym CPR mean ? | 3 |

### Pre-processing

In the pre-processing stage, the data is read from the csv file by removing the *id* and *label* column. These columns are not necessary for analysis as they cannot be used for the classification on a standalone basis. All the words are tokenized using the tokenize function of nltk module. If the words are digits and punctuations, they should be ignored. In this regard, the tokens are checked to see if there are digits and punctuations.

The text document of the questions was first split according to its constituent sentences. These sentences were further split into its constituent words and then the question number or any punctuation was also removed in order to standardize the questions. The numbers in the questions itself were ignored because some of them included years such as “1680” and these contributed classification of the sentences in most cases. After this the words were reconstituted into a sentence and placed into a list of strings.

**import** nltk  
**import** csv  
**import** pickle  
**from** nltk.classify.scikitlearn **import** SklearnClassifier  
**from** sklearn.linear\_model **import** LogisticRegression,SGDClassifier  
**from** sklearn.svm **import** SVC, LinearSVC  
f=open(**'train\_questions.txt'**,**'rU'**)

**with** open(**'train\_labels.csv'**,**'rb'**) **as** k:  
 reader=csv.reader(k)  
 train\_labels=list(reader) *#Reads in the training labels file*train\_labels.remove(train\_labels[0]) *#removes 'id' and 'label' from the label file*train\_data=f.read()  
train\_sent=train\_data.splitlines()   
train\_sent.remove(train\_sent[0]) *#split the training set into its corresponding   
#print len(train\_sent)*final\_set=[]  
all\_words1=[]  
token=nltk.RegexpTokenizer(**r'\w+'**) *#the word tokenizer that does not read in punctuation*all\_words=token.tokenize(train\_data) *#All words in the file are tokenized***for** j **in** all\_words:   
 **if** j.isdigit() **is False**: *#Read in only non numerical words present in the entire train set* all\_words1.append(j)  
e=0  
**for** i **in** train\_sent: *# Creates a list of list of lists with words of each question and the* words=[] *# corresponding label [0-6]* set1=[]  
 set2=[]  
 words=nltk.word\_tokenize(i)  
 set1.append(words[2:])   
 set1.append(train\_labels[e][1])  
 final\_set.append(set1)  
 e=e+1

all\_words2=nltk.FreqDist(all\_words1) *#The frequency distribution of all of the words present in the train file*word\_features=list(all\_words2.keys())  
*#print len(word\_features)*

### Creating Features

Since, the classifier is based on the bag of words model, this requires a numeric representation of the frequency of every word in all the questions. Bag of words is a rudimentary approach to text classification in this case question classification but because of its robust nature and its ability to work well smaller sized data made it the perfect choice for the model for this particular task. For this the questions were then passed to the CountVectorizer function which split them into its corresponding matrix of token counts. This gives a representation of the frequency of all the words that occur in all the questions and thus gives a representation of what kind of words would put a particular question under a particular category or label. None of the features were removed due to the fact that most words occurred in more than 90% of the documents and losing these features would drastically affect the output of the classifier. Considering also the small size of the training data any loss of features would affect the classifier performance. The training data was split into a 90:10 ratio for the actual training data and the testing data.

**def** find\_features(sent): *# Finding the features of each question and storing it as a dictionary* words2=set(sent)  
 features={}  
 **for** w **in** word\_features:  
 features[w]=(w **in** words2)  
 **return** features  
featuresets=[(find\_features(rev),category) **for** (rev, category) **in** final\_set]  
*# Finds all the features of all the questions present in the training set and puts it in the form of a list*

### SVM Classifier

A Linear Support Vector Classification was used as a classifier model for this task. The ability of LinearSVC to handle multi label data by using the one-vs-rest scheme makes a better choice for the case study considered. Also known as one-vs-all, this strategy consists of fitting one classifier per class or label. For each classifier, the class is fitted against all the other classes. In addition to its computational efficiency, another advantage of this approach is interpretability. Since each class is represented by one and one classifier only, it is possible to gain knowledge about the class by inspecting its corresponding classifier. This solution works particularly well in this case because of the multi label nature of the classification of questions.

training\_set=featuresets[:2900]  
testing\_set=featuresets[2900:]  
*#Split of 80:20 for training and testing set*print **"Training Classifier ......"**LinearSVC\_classifier = SklearnClassifier(LinearSVC())  
LinearSVC\_classifier.train(training\_set)  
print **"Accuracy"**print nltk.classify.accuracy(LinearSVC\_classifier, testing\_set)

**Ouput**

*Training Classifier ......*

*Accuracy*

*0.88*

**Table. 12.10.3. Predicted category of questions**

|  |  |
| --- | --- |
| **Question** | **Category** |
| Who did Arthur H. Bremer try to assassinate on May 15 1972 ? | 1 |
| Doesn't ROS stand for Return on Sales? | 0 |
| How do you buy stocks ? | 0 |
| Doesn't AAOS mean American Academy of Orthopaedic Surgeons? | 0 |
| What was the infamous feat of Germany 's U-2 submarine ? | 4 |
| How fast is a 45Mhz processor ? | 3 |
| Does Ahvaz lie in Iran? | 2 |
| What is the highest amount that can be got by using exactly 5 zeros? | 5 |
| What is the largest lake in North America ? | 4 |
| Is there not a drug test that can detect any drug that is in your system? | 4 |
| What was the Great Britain population from 1699-172 ? | 5 |
| What do I call the sons and daughters of my first cousins ? | 4 |
| Is that lady a scientist? | 1 |
| Is there not a way to prevent someone from seeing my answers on Quora? | 3 |
| Is there any way for a student to get an internship at Microsoft or at Facebook from high-school? | 3 |
| Does CASLPA mean Canadian Association of Speech-Language Pathologists and Audiologists? | 0 |
| Is that boy a celebrity? | 1 |
| What is the highest number that can be got by using exactly 5 ones? | 5 |
| Is there not a way to get Quora (or websites in general) to display with fonts of my own choosing in Chrome on Linux? | 3 |
| What does the acronym CPR mean ? | 3 |

## *Sentimental analysis*

### Introduction

Social computing involves constructing the models for different activities that take place in the social networks. Innovative and intellectual application development is the main focus of the social network analysis. People share their opinions and views on different types of products or issues through social networking sites like facebook, google, twiiter etc. Gaining insights into such comments on different products and views gives interesting conclusions in various domain areas like online- retail, marketing, scientific surveys, job-marketing, health care, customer marketing and other fields [11].

Opinions form the core part of human behaviours. Sometimes, when decision needs to be taken by the humans, they are dependent on the others opinions. For example, if a television need to be purchased an opinion is asked among our friends to see if the brand, configuration, warranty and others are reliable. Similar in enterprises, the customer opinions on products play a key role in long term assessment of the company.

Sentimental analysis is an ongoing field of research in computer science that analyzes the different types of content available in social networking sites. Information generated by the users in the form of tweets, status on social networking platforms like google, facebook are used to know the sentiments of the users on different issues of the world. Some of the companies use sentimental analysis for endorsing their brands for creating the awareness and reputation. Most of the data in the social networking sites is unstructured. It poses a great challenge in converting the unstructured data into a specific format for analysis.

Generally, when an opinion is needed for business application a survey is collected for a set of products and this survey is posted back o the company website for the consumer to know the insights. But due to the growth of social media, it is easy to approach to the consumers for getting the survey of products without the actual questionnaire. It is no longer needed to prepare a questionnaire, review it, select a set of users and ask them to provide the survey. However, the large amount of text in the social networking sites need to be collected first, then analyzed to see the positive and negative opinions. The average human reader cannot get a bigger insight into such information.

Due to proliferation of social media as discussed, automated sentiment analysis systems are needed. In this section, sentimental analysis is discussed with the examples in python. Before diving into the actual sentimental analysis using python, some of the concepts related to sentimental analysis are discussed in the upcoming sections.

### Different types of sentimental analysis

Opinions can be shared in the form of documents, status, short text, audio and video. Each format of the data shared has its own structure and need to be broken down into small tokens for opinion mining. The conventional approach of extracting the data and storing them in relational database systems will not be helpful. The schema of the database system might need to be changed every time. Thus, an alternative platforms and technologies are needed. However, the basic categories of sentimental analysis remains the same irrespective of the platform/technology. In this regard, the different levels of analysis that can be carried out are summarized as follows [11]:

* **Document level:** In this level of sentimental analysis, the main aim is to categorize opinion of the document whether it expresses positive or negative. For example, if there are a bunch of documents about the reviews of different products can the system be able to categorize it into positive and negative opinions. The drawback of this analysis is it can be carried on a single entity but not with multiple entities. The review has to be for a single rather and not for multiple products.
* **Sentence level:** In this type of analysis, each sentence is extracted from the document and classified as positive, negative or neutral. Neutral opinion means there is no opinion given. It is based on the subjectivity of the information present in the sentence. Sometimes, there can be difficulty in understanding the objectivity and subjectivity of the sentence. For example if the sentence is *‘The car was bought last week but the wiper had fallen off’*. Here even though the subject is car, it doesn’t imply the review for the car but only the objective of buying is present. In python *nltk* module has inbuilt methods for extracting the sentences and corpus information is also available for building the word model for analysis.
* **Entity and aspect level:** In this level of analysis, particular entity in the sentence is analyzed. For example, consider a sentence *‘the phone quality is good but not battery life’*. Here, the entity targets are phone quality and battery life. It indicates a positive opinion on the phone quality and negative opinion on the batter-life of it. In this way, for each sentence the entity targets need to be found first and then the opinions can be formed.

Other than these type of different sentimental analysis *corpus based analysis* can be carried out. Here, the sentimental analysis is carried out only on the basis of certain corpus information and specific to it. So, if any information outside the corpus is present in the product reviews, they can be ignored for analysis. In this way, different sentimental analysis can be carried out. In the next section, sentimental analysis is discussed with a sample dataset. The method used is sentence level analysis. Although, it is sentence level analysis, initially the documents are collected as the sources for analysis.

### Sentimental analysis in Python

In this section, a case study on the sentimental analysis is discussed for a dataset obtained from [12]. The dataset contains sentences from three different websites imdb, amazon, and yelp. For each website, there exist 500 positive and 500 negative sentences. Those were selected randomly for larger datasets of reviews. The attributes are text sentences, extracted from reviews of products, movies, and restaurants. The main aim of the case study is to build a model on these set of sentences and carry out sentimental analysis for other datasets.

The following code demonstrates the sentimental analysis in python. Initally, the modules such as *nltlk, word\_tokenize,random, wordnetlemmatizer, wodcloud, sopwords* are imported. The main module used is *nltk* because sentimental analysis involves natural language processing. Initially, the reviews collected by each site yelp, amazon and imdb are extracted into a dataframe using *df()* function. The data frames are concatenated to obtain the training set for sentimental analysis.

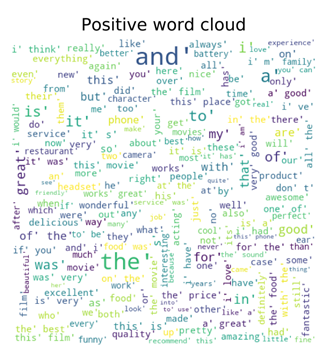
**import** nltk  
**from** nltk.tokenize **import** word\_tokenize  
**import** numpy **as** np  
**import** random  
**import** pickle  
**from** collections **import** Counter  
**from** nltk.stem **import** WordNetLemmatizer  
lemmatizer = WordNetLemmatizer()  
**import** tensorflow **as** tf  
**import** pandas **as** pd  
**import** re  
**import** csv  
**from** wordcloud **import** WordCloud,STOPWORDS  
**import** matplotlib **as** mpl  
**import** matplotlib.pyplot **as** plt  
  
  
df=pd.read\_table(**'yelp\_labelled.txt'**,names=(**'review'**,**'sentiment'**))  
df2=pd.read\_table(**'imdb\_labelled.txt'**,names=(**'review'**,**'sentiment'**))  
df3=pd.read\_table(**'amazon\_cells\_labelled.txt'**,names=(**'review'**,**'sentiment'**))  
df=pd.concat([df,df2])  
df=pd.concat([df,df3])  
  
Once, the data frames are concatenated, lexical analysis is performed on each sentence in the concatenated data set. To accomplish this, *word\_tokenize()* is used to get each words in each sentence. The set of words are included in a list *lex*. A list of positive and negative words is prepared initially that identifies the sentiment of the sentence. In the data set, each row is marked with 0 or 1, where 1 indicates a positive sentiment and 0 indicates a negative sentiment.

The sentences are iterated one by one to gather whether the sentence is tagged with positive or negative sentiment. Lemmatization is performed in the next step. The main aim of lemmatization is to map the sentences morphologically. For example, if the sentence is *the boy’s car are different colors* then in lemmatization, it is mapped as *the boy car be different colors.* In this way lemmatization is carried out for each sentence once the tokenized words are obtained.  
  
**def** create\_lexicon(sent,lex):  
 sent=re.sub(**"[^a-zA-Z]"**,**" "**,sent)  
 sent=sent.lower()  
 all\_words = word\_tokenize(sent)  
 lex+= list(all\_words)  
 **return** list(all\_words)  
  
  
lexicon = []  
pos\_words=[]  
neg\_words=[]  
**for** index, row **in** df.iterrows():  
 **if**(row[**'sentiment'**]==1):  
 pos\_words+=create\_lexicon(row[**'review'**],lexicon)  
 **else**:  
 neg\_words+=create\_lexicon(row[**'review'**],lexicon)  
  
  
lexicon = [lemmatizer.lemmatize(i) **for** i **in** lexicon]  
w\_counts = Counter(lexicon)  
l2 = []  
**for** w **in** w\_counts:  
 **if** 2000 > w\_counts[w] > 50:  
 l2.append(w)  
l3=[]  
**for** i **in** l2:  
 **if**(len(i)>1):  
 l3.insert(0,i)  
  
In this module, the lexicons are prepared for the data in data frame 2 i.e. for amazon reviews. Initially, the reviews are obtained from the data frame and tokenized into words. These words are then first converted into lowercase and appended to the features of the words to be compared for sentimental analysis.

**def** create\_feature(df2,lexicon):  
 featureset = []  
 **for** l **in** df2[**'review'**]:  
 current\_words = word\_tokenize(l.lower().decode(**'utf-8'**))  
 current\_words = [lemmatizer.lemmatize(i) **for** i **in** current\_words]  
 features = np.zeros(len(lexicon))  
 **for** word **in** current\_words:  
 **if** word.lower() **in** lexicon:  
 index\_value =lexicon.index(word.lower())  
 features[index\_value] += 1  
  
 features = list(features)  
 featureset.append(features)  
  
 **return** featureset

The codes in the previous sections deal with the pre-processing of the text. Once the pre-processing is carried out, the data is split into training set and testing set where first 2500 sentences are used for training and another 2500 sentences are used for testing. The data set is then iterated over the sentences to get the sentiment associated with it. From this iteration, a word cloud model of positive words and negative words are obtained in the next phase.

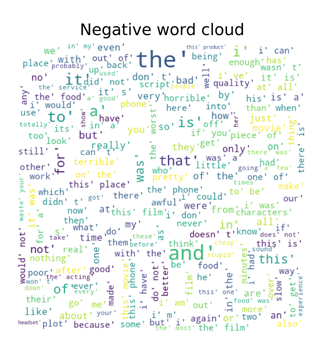
X\_train=create\_feature(df[:2500],l3)  
X\_test=create\_feature(df[2500:],l3)  
  
  
y\_train=list(df[**'sentiment'**][:2500])  
**for** i **in** range(len(y\_train)):  
 l=[0]\*2  
 l[int(y\_train[i])]=1  
 y\_train[i]=l  
y\_test=list(df[**'sentiment'**][2500:])  
**for** i **in** range(len(y\_test)):  
 l=[0]\*2  
 l[int(y\_test[i])]=1  
 y\_test[i]=l  
  
  
wordcloud = WordCloud(background\_color=**'white'**,  
 stopwords=STOPWORDS,  
 max\_words=200,  
 max\_font\_size=40,   
 random\_state=42  
 ).generate(str(pos\_words))  
print wordcloud  
plt.imshow(wordcloud)  
plt.axis(**'off'**)  
plt.title(**"Positive word cloud"**)  
plt.show()



**Fig. 12.11.3.1. Positive word cloud for sentimental analysis**

wordcloud = WordCloud(background\_color=**'white'**,  
 stopwords=STOPWORDS,  
 max\_words=200,  
 max\_font\_size=40,   
 random\_state=42  
 ).generate(str(neg\_words))  
print wordcloud  
plt.imshow(wordcloud)  
plt.axis(**'off'**)  
plt.title(**"Negative word cloud"**)  
plt.show()

The output of the plots for positive word cloud model and negative word cloud model is as shown in the figure 12.11.3.1 and figure 12.11.3.2 respectively. It can be observed from the figure 12.11.3.1 of positive word cloud model where the words such as *good, wonderful* indicate positive sentiments. Similarly, in the figure 12.11.3.2 of the negative word cloud model, the words like *terrible* and *bad* indicate the negative sentiments of the of the sentences in the data set.



**Fig. 12.11.3.2. Negative word cloud for sentimental analysis**

The actual training of the model is done using the following code. A tensorflow model is used for training and modelling of sentimental analysis where the three hidden layers are used. For each of the hidden layer, 2000 nodes are used. The *epochs* used is 500 which tunes the network to the best possible extent. All the hidden layers are encoded with the required training weights.

hiddden\_layer\_1 =2500  
hidden\_layer\_2 = 2500  
hidden\_layer\_3 = 2500  
  
n\_classes = 2  
batch\_size = 100  
epochs = 500  
  
x = tf.placeholder(**'float'**)  
y = tf.placeholder(**'float'**)  
  
hidden\_1\_layer = {**'f'**:hiddden\_layer\_1,  
 **'weight'**:tf.Variable(tf.random\_normal([len(X\_train[0]), hiddden\_layer\_1])),  
 **'bias'**:tf.Variable(tf.random\_normal([hiddden\_layer\_1]))}  
  
hidden\_2\_layer = {**'f'**:hidden\_layer\_2,  
 **'weight'**:tf.Variable(tf.random\_normal([hiddden\_layer\_1, hidden\_layer\_2])),  
 **'bias'**:tf.Variable(tf.random\_normal([hidden\_layer\_2]))}  
  
hidden\_3\_layer = {**'f'**:hidden\_layer\_3,  
 **'weight'**:tf.Variable(tf.random\_normal([hidden\_layer\_2, hidden\_layer\_3])),  
 **'bias'**:tf.Variable(tf.random\_normal([hidden\_layer\_3]))}  
  
output\_layer = {**'f'**:**None**,  
 **'weight'**:tf.Variable(tf.random\_normal([hidden\_layer\_3, n\_classes])),  
 **'bias'**:tf.Variable(tf.random\_normal([n\_classes])),}  
  
  
**def** layers(data):  
  
 layer\_1 = tf.add(tf.matmul(data,hidden\_1\_layer[**'weight'**]), hidden\_1\_layer[**'bias'**])  
 layer\_1 = tf.nn.relu(layer\_1)  
  
 layer\_2 = tf.add(tf.matmul(layer\_1,hidden\_2\_layer[**'weight'**]), hidden\_2\_layer[**'bias'**])  
 layer\_2 = tf.nn.relu(layer\_2)  
  
 layer\_3 = tf.add(tf.matmul(layer\_2,hidden\_3\_layer[**'weight'**]), hidden\_3\_layer[**'bias'**])  
 layer\_3 = tf.nn.relu(layer\_3)  
  
 output = tf.matmul(layer\_3,output\_layer[**'weight'**]) + output\_layer[**'bias'**]  
  
 **return** output  
  
**def** train\_model(x):  
 pred = layers(x)  
 cost = tf.reduce\_mean( tf.nn.softmax\_cross\_entropy\_with\_logits(logits=pred,labels=y) )  
 optimizer = tf.train.AdamOptimizer(learning\_rate=0.001).minimize(cost)  
  
 **with** tf.Session() **as** sess:  
 sess.run(tf.global\_variables\_initializer())  
 **for** epoch **in** range(epochs):  
 epoch\_loss = 0  
 i=0  
 **while** i < len(X\_train):  
 start = i  
 end = i+batch\_size  
 batch\_x = np.array(X\_train[start:end])  
 batch\_y = np.array(y\_train[start:end])  
  
 \_, k = sess.run([optimizer, cost], feed\_dict={x: batch\_x,y: batch\_y})  
 epoch\_loss += k  
 i+=batch\_size  
   
 print(**'Epoch'**, epoch+1, **'completed out of'**,epochs,**'loss:'**,epoch\_loss)  
 correct = tf.equal(tf.argmax(pred, 1), tf.argmax(y, 1))  
 accuracy = tf.reduce\_mean(tf.cast(correct, **'float'**))  
 print(**'Accuracy:'**,accuracy.eval({x:X\_test, y:y\_test}))  
 **return** pred  
  
  
*# In[86]:*p=train\_model(x)

In this way, the sentimental analysis can be carried out based on the reviews. The model considered here for the sentimental analysis has an accuracy score of 93%. In this section, a case study presented on the sentimental analysis showed the different steps in involved in the analysis with an example.

# Exercises

1. Why is removing stop words an important step in the bag of words model ?
2. Why do we removing very high frequency and very low frequency words when creating the feature set for question classification? What would happen if we did not remove them?
3. How does the preprocessing step in the spam classification using SVM help reduce the number of duplicate features?
4. How does SVM avoid overfitting of data, and thus provides better and more reliable results as compared to other traditional machine learning models?
5. For the question classification case study, how would the accuracy be affected if there is a change to the type of penalty used from L2 to L1 in LinearSVC classifier that was used?
6. How does the “one vs rest” classification tactic, give higher accuracy when it comes to multi label classification? What is the result “one vs one” Classifier is used?
7. How does Lemmatization of words affect the feature set? What would happen if words were not lemmatized?
8. For the sentimental analysis case study, try with the dataset provided by the link <http://ai.stanford.edu/~amaas/data/sentiment/>
9. Create the Bag of words model for the dataset provided in <https://archive.ics.uci.edu/ml/datasets/bag+of+words>
10. Create the set of positive and negative word cloud models for the dataset provided in the question 8.

# References

1. Aggarwal, C. C., & Zhai, C. (Eds.). (2012). *Mining text data*. Springer Science & Business Media.
2. Bird, S., & Loper, E. (2004, July). NLTK: the natural language toolkit. In *Proceedings of the ACL 2004 on Interactive poster and demonstration sessions* (p. 31). Association for Computational Linguistics.
3. Bird, S., Klein, E., & Loper, E. (2005). NLTK Tutorial: Introduction to Natural Language Processing. *Creative Commons Attribution*.
4. Neto, J. L., Freitas, A. A., & Kaestner, C. A. (2002, November). Automatic text summarization using a machine learning approach. In *Brazilian Symposium on Artificial Intelligence* (pp. 205-215). Springer, Berlin, Heidelberg.
5. Kim, S. B., Han, K. S., Rim, H. C., & Myaeng, S. H. (2006). Some effective techniques for naive bayes text classification. *IEEE transactions on knowledge and data engineering*, *18*(11), 1457-1466.
6. Youn, S., & McLeod, D. (2007). A comparative study for email classification. In *Advances and innovations in systems, computing sciences and software engineering* (pp. 387-391). Springer, Dordrecht.
7. Spam classification dataset: http://www.dt.fee.unicamp.br/~tiago/smsspamcollection/
8. Metzler, D., & Croft, W. B. (2005). Analysis of statistical question classification for fact-based questions. *Information Retrieval*, *8*(3), 481-504.
9. Zhang, D., & Lee, W. S. (2003, July). Question classification using support vector machines. In *Proceedings of the 26th annual international ACM SIGIR conference on Research and development in informaion retrieval* (pp. 26-32). ACM.
10. Yu, Y., Duan, W., & Cao, Q. (2013). The impact of social and conventional media on firm equity value: A sentiment analysis approach. *Decision Support Systems*, *55*(4), 919-926.
11. Fang, X., & Zhan, J. (2015). Sentiment analysis using product review data. *Journal of Big Data*, *2*(1), 5.
12. https://archive.ics.uci.edu/ml/datasets/Sentiment+Labelled+Sentences#