# NLP Applications

This chapter discusses NLP applications. Here, we will put all the learning from the previous chapters into action and will see what kind of application can be developed using the concepts we have learned. This will be a complete hands-on chapter. In the last few chapters we have learned most of the preprocessing steps that are required for any NLP application. We know how to use tokenizer, POS tag, and NER and how to perform parsing. This chapter will give you an idea how we can developed some of the complex NLP application using the concepts we have learned.

There are so many applications of NLP in the real world. Some of the most exciting and common examples you can observe are Google Search, Siri, machine translation, Google News, Jeopardy, and spell check. Some of these took many years for researchers to reach this level and bring these applications to their current state. NLP is complicated too; we have seen in the previous chapters that most of the processing steps, such as POS and NER, are still research problems. But with the use of NLTK, we have solved many of these problems with reasonable accuracy. We will not cover the more sophisticated applications such as machine translation or speech recognition in this book. But at this point in time, you should have enough background knowledge to understand some of the basic blocks of these applications. As a NLP enthusiast we should have a basic understanding of these NLP applications. I urge you to try and look for some of these NLP applications on the web and try to understand them.

By the end of this chapter :

* We will introduce reader to few common NLP applications.
* We will develop a NLP application (News summarizer) using what we have learnt so far.
* The importance of different NLP applications and essential details about each of them.

## Building your first NLP application

Let's start with one of the very complex NLP applications, which is **summarization**. The concept of summarization is quite simple. We are given an article/passage/ story and you will have to generate a summary of the content automatically. Summarization actually requires deep knowledge of NLP because we need to understand not just the structure of the sentence but also the structure of the entire text. We also need to know about genre of the text and the theme of the content.

Since it all looks very complex to us, let's try a very intuitive approach. We will assume that summarization is nothing but ranking of the sentences based on their importance and significance to you. We will create a few rules based on the understanding and the preprocessing tools we have learned so far and will try to come up with an acceptable summary of the news article.

I have scraped an article from the *New York Times* in a text file nyt.txt, in the following example. The idea here is to summarize this news article for us. Let's build a version of Google News for our personal use.

To start off, we need to keep in mind that, typically, a sentence that has more entities and nouns has greater importance than other sentences. We will try to normalize the same logic while calculating an **importance score**, using the following code. To get the top-n sentence, we can choose a threshold for the importance score.

Let's read the content of the news article. You can choose any news article with only contents of the news dumped into a text file. The content will look like this:

**>>>import sys**

**>>>f=open('nyt.txt','r')**

**>>>news\_content=f.read()**

**""" President Obama on Monday will ban the federal provision of some types of military-style equipment to local police departments and sharply restrict the availability of others, administration officials said.**

**The ban is part of Mr. Obama's push to ease tensions between law enforcement and minority communities in reaction to the crises in Baltimore; Ferguson, Mo.; and other cities. - - blic." It contains dozens of recommendations for agencies throughout the**

**country."""**

Once we parse the contents of the news we will need to split the entire news article into a list of sentences. We will go back to our old sentence tokenizer to break the entire news snippet into sentences. Let's also provide some form of sentence number so that we can identify and rank a sentence. Once we have the sentence, we will pass it through a word tokenizer and eventually through the NER tagger and POS tagger.

**>>>import nltk**

**>>>results=[] >>>for sent\_no,sentence in enumerate(nltk.sent\_tokenize(news\_content)):**

**>>> no\_of\_tokens=len(nltk.word\_tokenize(sentence))**

**>>> #print no\_of\_toekns**

**>>> # Let's do POS tagging**

**>>> tagged=nltk.pos\_tag(nltk.word\_tokenize(sentence))**

**>>> # Count the no of Nouns in the sentence**

**>>> no\_of\_nouns=len([word for word,pos in tagged if pos in**

**["NN","NNP"] ]) >>> #Use NER to tag the named entities.**

**>>> ners=nltk.ne\_chunk(nltk.pos\_tag(nltk.word\_tokenize(sentence)), binary=False)**

**>>> no\_of\_ners= len([chunk for chunk in ners if hasattr(chunk,**

**'node')])**

**>>> score=(no\_of\_ners+no\_of\_nouns)/float(no\_of\_toekns)**

**>>> >>> results.append((sent\_no,no\_of\_tokens,no\_of\_ners,\ no\_of\_nouns,score,sentence))**

In the preceding code, we are iterating over a list of sentences calculating a score based on a formula that is nothing but the fraction of tokens being entities as compared to a normal token. We are creating a tuple of all these as the results.

Now, the result is a tuple with all the scores, such as the number of nouns, entities, and so on. We can sort it based on the score in descending order, as shown in the following example:

**>>>for sent in sorted(results,key=lambda x: x[4],reverse=True):**

**>>> print sent[5]**

Now, the result of this will be sorted by the rank of the sentence. You will be amazed by the kind of results we get for the news article.

Once we have a list of no\_of\_nouns and no\_of\_ners scores, we can actually create some more complex rules around this. For example, a typical news article will start with very important details about the topic, and the last sentence will be a conclusion to the story.

Can we modify the same snippet to incorporate this logic?

The other theory of this kind of summarization is that the important sentences generally contain important words and that most of the the discriminatory words across the corpus will be important. The sentences that has very discriminatory words are important. A very simple measure of that is to calculate the **TF-IDF** (**term frequency–inverse document frequency**) score of each and every word and then look for an average score normalized by the words that are important; this can then be used as the criteria to choose sentences for our summary.

For explaining the concepts instead of the entire article, just take the first three sentences of the article. Let's see how you can implement something this complex using very few lines of code:

This code require installing scikit. If you have installed anaconda or canopy you should be fine otherwise install scikit using this link.

[IUUQTDJLJUMFBSOPSHTUBCMFJOTUBMMIUNM](http://scikit-learn.org/stable/install.html)

**>>>import nltk**

**>>>from sklearn.feature\_extraction.text import TfidfVectorizer**

**>>>results=[]**

**>>>news\_content="Mr. Obama planned to promote the effort on Monday during a visit to Camden, N.J. The ban is part of Mr. Obama's push to ease tensions between law enforcement and minority \communities in reaction to the crises in Baltimore; Ferguson, Mo. We are, without a doubt, sitting at a defining moment in American policing, Ronald L. Davis, the director of the Office of Community Oriented Policing Services at the Department of Justice, told reporters in a conference call organized by the White House"**

**>>>sentences=nltk.sent\_tokenize(news\_content)**

**>>>vectorizer = TfidfVectorizer(norm='l2',min\_df=0, use\_idf=True, smooth\_ idf=False, sublinear\_tf=True)**

**>>>sklearn\_binary=vectorizer.fit\_transform(sentences)**

**>>>print countvectorizer.get\_feature\_names()**

**>>>print sklearn\_binary.toarray() >>>for i in sklearn\_binary.toarray():**

**>>> results.append(i.sum()/float(len(i.nonzero()[0]))**

In the preceding code, I am using some unknown methods, such as

TfidfVectorizer, which is a scoring method that will calculate a vector of TF-IDF scores for each sentence in a given list of sentences. Don't worry, we will talk about this in more detail. For this chapter, consider it as a black-box function that, for a given list of sentences/documents, will give you the score corresponding to each sentence and will also provide the ability to build a term-doc matrix that will look just like our output.

We got a dictionary of all the words present across all the sentences and then we have a list of lists where each element assigns each word its individual TF-IDF score. If you got that right, then you can see some of the stop words will get a near-zero score while some discriminatory words like ban and Obama will get a very high score. Now once we have this in the code, I will look for the average TF-IDF score by using only non-zero TF-IDF words. This will give us a similar kind of score as we got in our first approach.

You will be amazed by the kind of results a simple algorithm can give. I think now you are all set to write your own news summarizer that summarizes any given news article with the two preceding algorithms and the summary will look quite decent. While this kind of approach will give you a decent summarization, it's actually very poor when you compare it with the current state of summarization research. I would recommend looking for some literature relating to summarization. I would also like you to try and combine both the approaches for summarization.

## Other NLP applications

Some of the other NLP applications are text classification, machine translation, speech recognition, information retrieval, information extraction, topic segmentation, and discourse analysis. Some of these problems are actually very difficult NLP tasks and a lot of research is still going on in these areas. We will discuss some of these in depth in the next chapter, but as NLP students, we should have a basic understanding of these applications.

### Machine translation

The easiest way to understand machine translation is to know how we translate from one language to other. Our mind parses the sentence structure and tries to understand the sentence. Once we understand the sentence, we will try to substitute the words from the original language with those from the target language. While substituting, we use the grammar rules of the target sentence and finally achieve the correct translation.

Interlingua

Semantic

Decomposition

Semantic

Composition

Semantic

Structure

Semantic

Structure

Semantic

Generation

Syntactic

Structure

Syntactic

Generation

Morphological

Generation

T

arget T

ext

Direct

Syntactic

T

ransfer

Semantic

T

ransfer

Syntactic

Structure

Semantic

Analysis

Syntactic

Analysis

W

ord

Structure

Source T

ext

Morphological

Analysis

W

ord

Structure

Loosely, the process can be translated to something like the pyramid in the preceding figure. If we start from the source language text, we have to tokenize the sentences that we will parse the tree (for syntactic structure in easy words) to make sure the sentences are correctly formulated. Semantic structure holds the meaning of the sentences, and at the next level, we reach the state of Interlingua, which is an abstract state that is independent from any language. There are multiple ways in which people have developed methods of translation. The more you go on towards the root of the pyramid, the more intense is the NLP processing required. So, based on these levels of transfer, there are a variety of methods that are available. I have listed two of them here:

* **Direct translation**: This will be more of a dictionary-based machine translation while you have huge corpora of source and target language words. This kind of transfer is possible for applications where we have a large corpus of languages available. It's popular because of its simplicity.
* **Syntactic transfer**: Here you will try to build a parser of the source language. There are varieties of ways in which people have approached the problem of parsing. There are deep parsers that actually take care of some parts of semantics too. Once you have a parser, target word substitution happens and the target parser can generate the final sentence in the target language.

### Statistical machine translation

**Statistical machine translation** (**SMT**) is one of the latest approach of machine translation, where people have come up with a variety of ways to apply statistical methods to almost all the aspects of machine translation. The idea behind this kind of algorithm is that we have a huge volume of corpora, parallel text, and language models that can help us predict the language translation in the target language. Google Translate is a great example of SMT, where it learns from the corpora of different language pairs and builds an SMT around it.

### Information retrieval

**Information retrieval** (**IR**) is also one of the most popular and widely used applications. The best exmple of IR is Google Search, where—given an input query from the user—the information retrieval algorithm will try to retrieve the information which is relevant to the user's query.

In simple words, IR is the process of obtaining the most relevant information that is needed by the user. There are a variety of ways in which the information needs can be addressed to the system, but the system eventually retrieves the most relevant infromation.

The way a typical IR system works is that it generates an indexing mechanism, also known as **inverted index**. This is very similar to the indexing schemes used in books, where you will have an index of the words present throughout the book on the last pages of the book. Similarly, an IR system will create an inverted index poslist. A typical posting list will look like this:

**< Term , DocFreq, [DocId1,DocId2] >**

**{"the",2 --->[1,2] }**

**{"US",1 --->[2] }**

**{"president",2 --->[1,2] }**

So if any word occurs in both document 1 and document 2, the posting list will be a list of documents pointing to terms. Once you have this kind of data structure, there are different retrieval models that can been introduced. There are different retrieval models that work on different types of data. A few are listed in the following sections.

#### Boolean retrieval

In the Boolean model, we just need to run a Boolean operation on the poslist. For example, if we are looking for a search query like "US president", the system should look for an intersection of the postlist of "US" and "president".

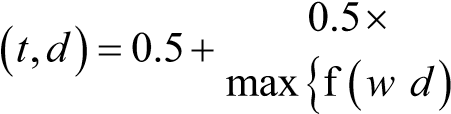
**{US}{president}=> [2]**

Here, the second document turns out to be the relevant document.

#### Vector space model

The concept of **vector space model** (**VSM**) derives from geometry. The way to visualize the documents in the high dimension space of vocabulary is to represent it as a vector. So each and every document is represented as a vector in that space. We can represent the vector in various ways, but one of the most useful and efficient ways is using TF-IDF.

Given a term and a corpus, we can calculate the **term frequency** (**TF**) and **inverse document frequency** (**IDF**) using the following formula:

f (*t d*, )

tf

, : *w*∈*d*}

The TF is nothing but the frequency in the document. While the IDF is the inverse of document frequency, which is the count of documents in the corpus where the term occurs:

##### idf (*t D*, )= log {*d* ∈ ∈*DN*:*t d*}

There are various normalization variants of these, but we can incorporate both of these to create a more robust scoring mechanism to get the scoring of each term in the document. To get to a TF-IDF score, we need to multiply these two scores as follows: tfidf (*t d*, ,*D*)= tf (*t*,*d*)×idf (*t*,*D*)

In TF-IDF, we are scoring a term for how much it is present in the current document and how much it is spread across the corpus. This gives us an idea of the terms that are not common across corpora and where ever they are present have a high frequency. It becomes discriminatory to retrieve these documents. We have also used TF-IDF in the previous section, where we describe our summarizer.The same scoring can be used to represent the document as a vector. Once we have all the documents represented in a vectorized form, the vector space model can be formulated.

In VSM, the search query of the user is also considered as a document and represented as a vector. Intuitively, a dot product between these two vectors can be used to get the cosine similarity between the document and the user query.

D1

President

D2

US

The

Querv=Obama

In the preceding diagram, we see that these same documents can be represented using each term as an axis and the query Obama will have as much relevance to *D1* as compared to *D2*. The scoring of the query for relevant documents can be formulated as follows:

( ) d j ⋅q = ∑*iN*=1 *w wi j*, , *i q*, sim *d qj* , = d j q ∑ ∑*N w*2 *N wi q*2,

*i*=1 *i j*, *i*=1

#### The probabilistic model

The probabilistic model tries to estimate the probability of the user's need for the document. This model assumes that the probability of the relevance depends on the user query and document representation.The main idea is that a document that is in the relevant set will not be present in the non-relevant set. We denote dj as the document and q as user query; R represents the relevant set of documents, while P represents the non-relevant set. The scoring can be done like this:



sim(*d qj* , )= ( *j* )) *P R*( | *d*

*P R* | *d j*

For more topics on IR, I would recommend that you read from the following link:

http://nlp.stanford.edu/IR-book/html/htmledition/

irbook.html

### Speech recognition

Speech recognition is a very old NLP problem. People have been trying to address this since the era of World War I, and it still is one of the hottest topics in the area of computing. The idea here is really intuitive. Given the speech uttered by a human can we convert it to text? The problem with speech is that we produce a sequence of sounds, called **phonemes**, that are hard to process, so speech segmentation itself is a big problem. Once the speech is processable, the next step is to go through some of the constraints (models) that are built using training data available. This involves heavy machine learning. If you see the figure representing the modeling as one box of applying constraints, it's actually one of the most complex components of the entire system. While acoustic modeling involves building modes based on phonemes, lexical models will try to address the modeling on smaller segments of sentences, associating a meaning to each segment. Separately language models are built on unigrams and bigrams of words.

Once we build these models, an utterence of the sentences is passed through the process. Once processed for initial preprocessing, the sentence is passed through these acoustic, lexical, and language models for generating the token as output.

Applying

Constraints

Acoustic

Models

Lexical

Models

Language

Models

Representation

Search

Recognized

W

ords

Speech

Signal

T

raining Data

### Text classification

Text classification is a very interesting and common application of NLP. In your daily work, you interact with many text classifiers. We use a spam filter, a priority inbox, news aggregators, and so on. All of these are in fact applications built using text classification.

Text classification is a well-defined and somewhat solved problem, and it has been applied across many domains. Typically, any text classification is the process of classifying text documents using words and the combination of words. While it's a typical machine learning problem, many of the preprocessing steps used in text classification are from NLP.

An abstract diagram of text classification is shown here:

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| |  |  |  |  |  |  |  | | --- | --- | --- | --- | --- | --- | --- | | Tokenization |  | stop-word |  | stemming |  | pos\_tag |   preprocessing | ! #  # *D*1  # *D*  # 2 # :  # :  ## *D*  % *n* | *T T*  1 2 *w w*   1. 21 *w w* 2. 22   : :  : : *w w*  1*n* 2*n* |        | |  |  |  | | --- | --- | --- | | |  | | --- | | Text classifier | |  |   *Tt* "  $ *w*  *t*1 $ *w* $  *t*2 $  : $ : $  $$  *wtn* & | pos/ neg  class |

Here we have a bunch of documents for a set of classes. For simplicity, we will use just binary 1/0 as the class. Now let's assume it's a spam detection problem where 1 represents spam and 0 represents normal text which is not to be considered as spam.

The process involves some of the preprocessing steps we learned in previous chapters. While some of these are essential, it depends on the kind of text classification problem we are trying to solve. So in few cases, it's more a case of feature engineering while we drop some of the preprocessing steps. The final goal of feature engineering is to generate a **Term doc matrix** (**TDM**), which holds the vocabulary of the entire corpus: columns and rows are the documents, while the matrix represents a scoring mechanism to show the **Bag of word** (**BOW**) representation. The weighting scheme can be varied to TF, TF-IDF, Bernoulli, and other variations of term frequency.

There are also ways to induce features such as the POS of a given feature, contextual POS, and others, to make our feature space more NLP intense. Once the

TDM is generated, the text classification problem becomes a typical supervised/ unsupervised classification problem, where given a set of samples, we need to predict what sample belongs to what class. The next chapter is dedicated entirely to this topic. This is definitely a splendid application of NLP/ML and is used quite often for commercial purposes.

Some of the most common use cases in day-to-day scenarios are sentiment analysis, spam classification, e-mail categorization, news categorization, patent classification, and so on. We will talk about text classification in more detail in the next chapter.

### Information extraction

**Information extraction** (**IE**) is a process of extracting meaningful information from unstructured text. IE is yet another widely popular and highly important application. In general, an information extraction engine harnesses huge numbers of unstructured documents and generates some sort of structured/semi-structured **knowledge base** (**KB**) that can be deployed to build an application around it. A simple example is that of generating a very good ontology using a huge set of unstructured text documents. A similar project in this line is DBpedia, where all the Wikipedia articles are used to generate the ontology of artifacts that are interrelated or have some other relationship.

There are mainly two ways of extracting information:

* **Rule-based extraction**: This method is where one uses a template filling mechanism. The idea is to look for some kind predefined use cases for expected outcomes and try to mine the unstructured text for that specific template. For example, building a knowledge base of football will involve getting information on all the players and their profiles, the statistics, some personal information, and so on. All that can be well defined and extracted using either pattern-based rules or POS tags, NERs and relation extraction.
* **Machine learning based**: The other approach involves deeper NLP-based methods such as building a parser specific to the need of our knowledge base. Some of the KBs will require mining the entities that can't be extracted using a pre-trained NER, so we have to build a custom NER. We might want to develop a relation extraction algorithm specific to the KB we are trying to build. This is a more NLP-intensive approach, where we are developing a NLP-based parser or tagger to use for heavy machine learning.

### Question answering systems

**Question answering** (**QA**) systems are intelligent systems that can address any question based on their knowledge base. One of the major examples of this is IBM Watson, which took part in the TV show *Jeopardy* and won over human opponents. A QA system can be broken down to building components from speech recognition for querying the knowledge base while the knowledge base is generated using information retrieval and extraction.

Once you have a question for the system, one big problem is to classify/categorize the question in different ways. The other aspect is to search the knowledge base effectively and retrieve the most precise document. Even after that, we have to generate the answer in a natural way using some of the other applications, such as summarization and parsing.

### Dialog systems

Dialog systems are considered the dream application, where given a speech in source language, the system will perform speech recognition and transcribe it to text. This text will then go to a machine translation system that can translate the speech into the target language and then a text-to-speech system will convert it into speech in the target language. This is one of the most desirable applications of NLP, where we can talk to a computer in any language and the computer will reply in the same language. This kind of application can actually destroy the language barrier that exists in the world.

Apple Siri and Google Voice are examples of some of the commercial applications in the line of dialog systems intelligent enough to understand our information needs, try to address them in a set of actions or information, and respond in a human-like manner.

### Word sense disambiguation

**Word sense disambiguation** (**WSD**) is also one of the difficult challenges not solved even after years of research and one of the major causes of application problems, such as question answering, summarization, search, and so on. A simple way to understand the concept is that many words have different meanings when used in different contexts. For example, "cold" in the following example: • The ice-cream is really cold

• That was cold blooded!

Here the word "cold "has two different senses, and it's really hard for computers to understand this concept. Some of the other NLP processing options, such as POS tagging and NER, are used to resolve some of these problems.

### Topic modeling

Topic modeling, in the context of a large amount of unstructured text content, is really an amazing application, where the primary task is to identify the emerging topics in the corpus and then categorize the documents in the corpus as per these topics. We will discuss this briefly in the next chapter.

Topic modeling uses the same NLP preprocessing, for example, sentence split, tokenization, stemming, and so on. The beauty of the algorithms is that we have an unsupervised way of categorizing the document; also, topics are generated without explicitly mentioning anything prior to the process. I encourage you to look at topic modeling in more detail. Try reading about **latent dirichlet allocation** (**LDA**) and **latent semantics indexing** (**LSI**) for more detail.

### Language detection

Given a snippet of text, the detection of language is also a problem. The application of language detection is very important for some of the other NLP applications, such as search, machine translation, speech, and so on. The main concept is learning from the text as features what the language is. A variety of machine learning and NLP techniques are used for feature engineering in the process.

### Optical character recognition

**Optical character recognition** (**OCR**) is an application of NLP and computer vision, where given a handwritten document/ non-digital document, the system can recognize the text and extract it into digital format. This has also been widely researched in the area of machine learning for many years. Some of the big OCR projects are Google Books, where they use OCR to convert non-digital books into a centralized library.

## Summary

In conclusion, there are many NLP applications around us that we interact with in our day-to-day routines. NLP is difficult and complex, and some of these problems are still unsolved or do not yet have perfect solutions. So anybody who is looking for problems in NLP, try exploring the literature around that. It's a great time to be an NLP researcher. In the era of Big Data, NLP applications are very popular. Many research labs and organizations are currently working on NLP applications such as speech recognition, search, and text classification.

I believe we have learned a lot up until this chapter. For the next couple of chapters, we will delve deeply into some of the applications described here. We have reached a point where we know enough NLP related preprocessing tools and also have a basic understanding about some of the most popular NLP applications. I hope you leverage some of this learning to build a version of an NLP application.

In the next chapter, we will start with some of the important NLP applications, such as text classification, text clustering, and topic modeling. We will move slightly away from the pure NLTK applications on to how NLTK can be used in conjunction with other libraries.