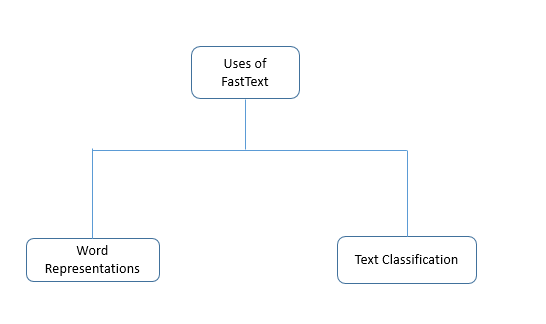
**Text Classification & Word Representations using fastText (An NLP library by Facebook)**

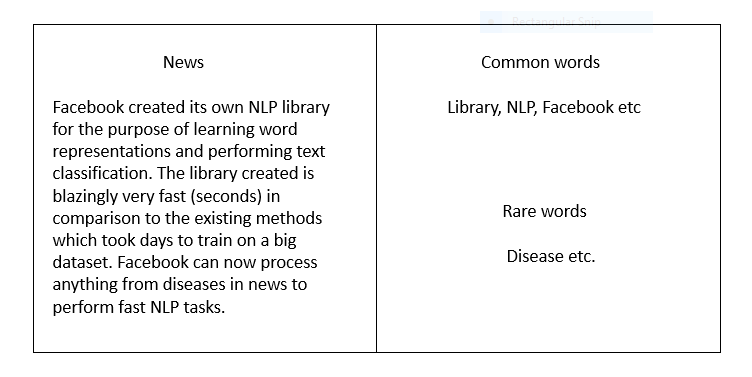
1. Introduction

fastText is a library created by the Facebook Research Team for **efficient learning of word representations** and **sentence classification**.



**Figure 1: Major Uses of FastText Library**

This library has gained a lot of traction in the NLP community and is a possible substitution to the gensim package which provides the functionality of Word Vectors etc. fastText differs in the sense that word vectors i.e. word2vec treats every single word as the smallest unit whose vector representation is to be found but fastText assumes a word to be formed by a n-grams of character, for example, sunny is composed of [sun, sunn, sunny], [sunny, unny, nny] etc, where n could range from 1 to the length of the word. This new representation of word by fastText provides the following benefits over word2vec or glove. It is helpful to find the vector representation for rare words. Since rare words could still be broken into character n-grams, they could share these n-grams with the common words. For example, for a model trained on a news dataset, the medical terms for instance: diseases can be the rare words.

 **Figure 2: An Instance of Common words and rare words**

It can give the vector representations for the words not present in the dictionary (OOV words) since these can also be broken down into character n-grams. word2vec and glove both fail to provide any vector representations for words not in the dictionary. For example, for a word like ***stupedofantabulouslyfantastic****,*which might never have been in any corpus, gensim might return any two of the following solutions – a) a zero vector    or      b) a random vector with low magnitude. But fastText can produce vectors better than random by breaking the above word in chunks and using the vectors for those chunks to create a final vector for the word. In this particular case, the final vector might be closer to the vectors of fantastic and fantabulous. Character n-grams embeddings tend to perform superior to word2vec and glove on smaller datasets.

1. **Process of Installation**

To make full use of the fastText library, please make sure you have the following requirements satisfied:

1. OS – MacOS or Linux
2. C++ complier – gcc or clang
3. Python 2.6+, numpy and scipy.

To install fastText, type the code below-

1. git clone https://github.com/facebookresearch/fastText.git
2. cd fastText
3. make

You can check whether fastText has been properly installed by typing the below command inside the fastText folder.

./fasttext

If everything was installed correctly then, you should see the list of available commands for fastText as the output

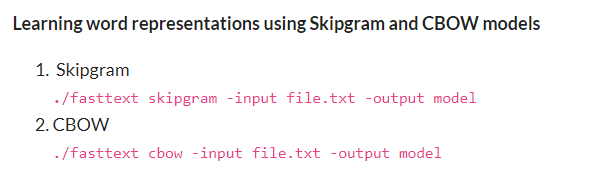
**3.Process of Implementation**

As stated earlier, fastText was designed for two specific purposes- **Word Representation Learning** and **Text Classification**. We will see each of these steps in detail. Let us get started with learning word representations.

* 1. **Learning Word Representation**

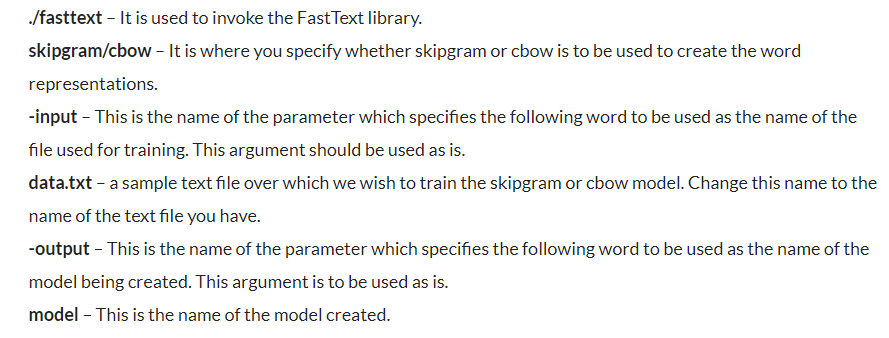
Words in their natural form cannot be used for any Machine Learning task in general. One way to use the words is to transform these words into some representations that capture some attributes of the word. It is analogous to describing a person as – [‘height’:5.10 ,’weight’:75, ‘colour’:’dusky’, etc.] where height, weight etc are the attributes of the person. Similarly, word representations capture some abstract attributes of words in the manner that similar words tend to have similar word representations. There are primarily two methods used to develop word vectors – Skipgram and CBOW.

We will see how we can implement both these methods to learn vector representations for a sample text file using fastText.



**Figure 3: Instance of Skipgram and CBOW Models**

Let us see the parameters defined above in steps for easy understanding.



**Figure 4: Commands and parameters involved for word representations**

* 1. **Print word vectors of a word**

In order to get the word vectors for a word or set of words, save them in a text file. For example, here is a sample text file named queries.txt that contains some random words. We will get the vector representation of these words using the model we trained above.

./fasttext print-word-vectors model.bin < queries.txt

To check word vectors for a single word without saving into a file, you can do

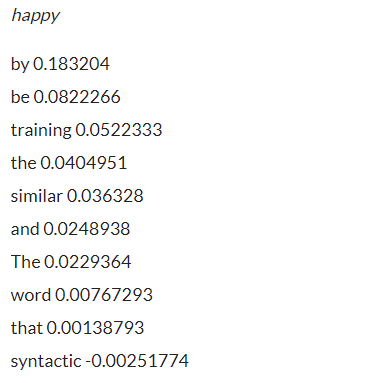
echo "word" | ./fasttext print-word-vectors model.bin

* 1. **Finding similar words**

You can also find the words most similar to a given word. This functionality is provided by the **nn**parameter. Let’s see how we can find the most similar words to “happy”.

./fasttext nn model.bin

After typing the above command, the terminal will ask you to input a query word.



**Figure 5: Words similar to happy**

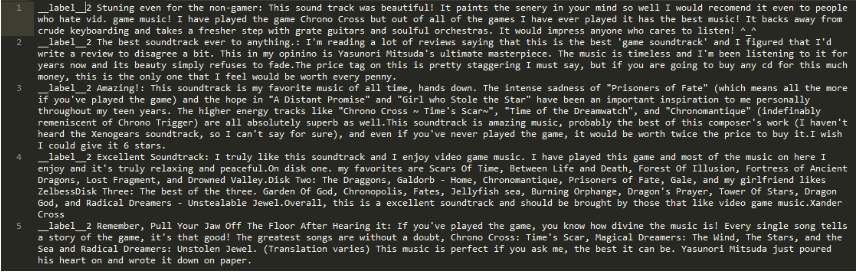
The above is the result returned for the most similar words to *happy.*Interestingly, this feature could be used to correct spellings too.

1. **Text Classification**

As suggested by the name, text classification is tagging each document in the text with a particular class. Sentiment analysis and email classification are classic examples of text classification. In this era of technology, millions of digital documents are being generated each day. It would cost a huge amount of time as well as human efforts to categorise them in reasonable categories like spam and non-spam, important and unimportant and so on. Text classification techniques of NLP come here to our rescue. Let’s see how by doing hands-on practice based on a sentiment analysis problem.

Before we jump upon the execution, there is a word of caution about the training file. The default format of text file on which we want to train our model should be    \_ \_ label \_ \_ <X>  <Text>.

Where \_ \_label\_ \_ is a prefix to the class and <X> is the class assigned to the document. Also, there should not be quotes around the document and everything in one document should be on one line.



**Figure 6: Labels and the Class**

**4.1 Training the Classifier**

Here, the parameters are same as the one mentioned while creating word representations. The only additional parameter is **-label.**This argument takes care of the format of the label specified. The file that you downloaded contains labels with the prefix *\_\_label\_\_.*

./fasttext supervised -input train.ft.txt -output model\_kaggle -label  \_\_label\_\_

If you do not wish to use default parameters for training the model, then they can be specified during the training time. For example, if you explicitly want to specify the learning rate of the training process then you can use the argument ***-lr***to specify the learning rate.

./fasttext supervised -input train.ft.txt -output model\_kaggle -label  \_\_label\_\_ -lr 0.5

**The other available parameters that can be tuned are –**

* **-lr : learning rate [0.1]**
* **-lrUpdateRate : change the rate of updates for the learning rate [100]**
* **-dim : size of word vectors [100]**
* **-ws : size of the context window [5]**
* **-epoch : number of epochs [5]**
* **-neg : number of negatives sampled [5]**
* **-loss : loss function {ns, hs, softmax} [ns]**
* **-thread : number of threads [12]**
* **-pretrainedVectors : pretrained word vectors for supervised learning []**
* **-saveOutput : whether output params should be saved [0]**

**4.2 Testing the result**

./fasttext test model\_kaggle.bin test.ft.txt

**N** 400000  
**P@1** 0.916  
**R@1** 0.916

1. **Pros and Cons of FastText**

Like every library in development, it has its pros and cons. Let us state them explicitly.

* 1. **Pros of the concerned Library**

1. The library is surprisingly very fast in comparison to other methods for achieving the same accuracy. Here is the result published by the Facebook research team in support of the argument. Sentence Vectors(supervised) can be easily computed.
2. fastText works better on small datasets in comparison to gensim.
3. fastText performs superior to gensim in terms of syntactic performance and fairs equally well in case of semantic performance.
   1. **Cons of the Library**
4. This is not a standalone library for NLP since it will require another library for the pre-processing steps.
5. Though, this library has a python implementation. It is not officially supported.