Introduction

Making computers understand different human languages has been an important area of research in machine learning community since long. From bag of words to LDA and now highly celebrated word2vec,people evoluted different methods in order to make computers “intelligent” enough so that they can understand human language and have been successful to some extent in this quest. Although it’s a tough target to achieve, but it is essential and can be extremely useful in many places like spam e-mail classification or job recommendation and so on. So, efforts put to achieve this task are worth it. But, developing algorithms is just half the battle. We can’t implement these techniques in real life situations manually because of huge amount of data. So, developing tools and learning to implement them is a really important task.

Many tools have been developed to implement NLP over the time. Each of them has its own pros and cons. In the line of these tools, Facebook AI Research launched a library named ‘fastText’. Its speciality lies in the fact that it can be used efficiently on large datasets for text classification. It can also be applied to perform advanced machine learning tecniques like learning vector representations. Also, it has been designed to work on different languages.

In this article, we will learn to implement some of the NLP techniques in fasttext, a python interface of Facebook’s fastText. Without anymore theoretical information, let’s get started with installation and then implementation.

Note:- This article focuses on implementation and assumes that the reader is familiar with concepts of Natural Language Processing. If you are new to the field, refer this [article](https://www.analyticsvidhya.com/blog/2017/06/word-embeddings-count-word2veec/) first before moving ahead.

Table of contents-

1. Installation
2. Implementation
   1. Learning word representations
   2. Text Classification
3. Important parameters

Installation-

Machine pre-requisites:- fastText builds on modern Mac OS and Linux distributions. fasttext supports Python 2.6 or newer. Cython is required for building C++ extension.

If you have all the required packages mentioned above, you can install fasttext by using the following command line

pip install fasttext

Implementation-

We will learn to use fasttext for two specific purposes- word representation learning and Text classification. Let’s get started with word representation learning.

Learning word representations-

I assume you have prior knowledge about word representation learning and related concepts like word2vec and pretrained word vectors and implemented them at least once using any library. If not, it is highly recommended to go through the article mentioned above before moving ahead.

**Learning word representations using skipgram and cbow models-**

**import fasttext**

**# implementing skipgram model**

**model = fasttext.skipgram("/home/nss/Documents/fasttext.txt",'model')**

**# implementing cbow model**

**model\_cbow = fasttext.cbow("/home/nss/Documents/fasttext.txt",'model\_cbow')**

Let me explain the two arguments one by one-

1. The first argument we pass to the function should be the path of the utf-8 encoded text file we want to use for training word vectors.
2. **Second argument serves as path for saving output files. In this case, .skipgram()** method saves two files in my current working directory named after the second arguments I have passed : model.bin and model.vec . You should check in your directory what is contained in the saved files. It will make things clear and intuitive. Basically, model.vec is a text file containing the word vectors for each word in the dictionary created, one word per line. We will see how to see the available words in dictionary shortly. On the other hand, model.bin is a binary file. It contains the parameters of model along with the dictionary and all hyper-parameters.

I will mention other arguments accepted by above functions later in the article. Let’s see what are the words present in dictionary.

**# printing words present in dictionary**

**print(model.words)**

**print(model\_cbow.words)**

Also, you can find word vectors for out of dictionary words.

**# printing vector for ‘king’**

**print(model['king'])**

**print(model\_cbow['king'])**

I trained my models on a very small text data and certainly got very inaccurate results because purpose in this case is demonstration. If you want to apply these NLP concepts in some real life problems, you will need more accurate vectors. It’s possible only when you train your model on a huge corpus. But doing so can be time costly and there could be machine constraints. To avoid such problems, almost every NLP package has methods to use pre-trained models and fasttext is no different in this case.

You can use pre-trained model released by Facebook and trained using fastText. Follow this [link](https://github.com/facebookresearch/fastText/blob/master/pretrained-vectors.md) to download the models. For now, I will use the models trained in previous section because the fastText model download is 6.1 GB!

**#loading the pre-trained model**

**model\_loaded = fasttext.load\_model('model.bin')**

**# using the loaded model to find vector for ‘king’**

**model\_loaded['king']**

Text Classification-

As suggested by name, text classification is tagging each document in text with a particular class. Sentiment analysis and email classifaction are classic examples of text classification. In this era of techology, millions of digital documents are being generated each day. It would cost a huge amount of time as well as human efforts to categorize them in reasonable categories like spam and non-spam, important and un-important 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. I have taken the data for this analysis from [kaggle](https://www.kaggle.com/bittlingmayer/amazonreviews).

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 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.

If you can’t visualise the format I am talking about, download the data set from the mentioned link and have a look at it. Everything will be clear in a moment once you see the data and relate to the mentioned format.

Infact, the reason why I have selected this data for this article is that the data is already available exactly in the required default format.If you are completely new to fasttext and implementing text classification for very first time in fasttext, I would strongly recommend to use the data mentioned above.

In case your data has some other formats of label, don’t be bothered. fasttext will take care of it once you pass suitable argument. We will see how to do it in a moment. Just stick to the article.

After this briefing about text classification, let’s move ahead and land on implementation

**#training the classifier**

**classifier = fasttext.supervised("/home/nss/Documents/train.ft.txt",'text\_classifier', label\_prefix='\_\_label\_\_')**

**#running the model on test file**

**result = classifier.test("/home/nss/Documents/test.ft.txt")**

**#checking precision and recall**

**print(result.recall)**

**print(result.precision)**

Again, I will explain the arguments in the .supervised() method.

1. First argument passes the path of the text file on which you want to train the model.
2. Second argument specifies another path where the function saves two files ‘text\_classifier.vec’ and ‘text\_classifier.bin’.
3. Third argument I have mentioned takes the format of label prefix in the training file. I have passed the default format. You can pass whatever format in yourcase.

In .test() method, the passsed argument contains the path of test file on which you want to test the model. The default format of file should be [utf- 8] encoded text file.

Once trained and tested, let’s run the model on a sample document for prediction and see what happens.

**#running model on example document**

**texts = [ 'this is an example text']**

**labels = classifier.predict(texts)**

**print (labels)**

It should print the label assigned by the model to the given text document.

Also, you can print the probability of falling the document in ‘k’ best classes by using following code

**#printing probabilities**

**classifier.predict\_proba(texts,k=2)**

Important parameters**-**

**Skipgram and CBOW model-**

1. lr = learning rate . Default value is 0.05 .
2. dim = size of word vectors. Default size is 100.
3. ws = context window size. Default window size is 5.
4. epoch =number of epochs. Default value is 5.
5. encoding = encoding of input file. Default is [utf- 8]
6. word\_ngrams = maximum length of word ngram. Default length is 1.

**Text classification model-**

1. lr = learning rate. Default value is 0.1
2. label\_prefix = label prefix for input text file. Default format is already mentioned.
3. loss = loss function used in trainning.
4. dim = size of word vectors. Default size is 100.
5. pretrained\_vectors = pre-trained word vectors file(.vec file) for supervised learning.

End Notes-

Although the article assumes many pre-requisites, I have tried to explain all the required points so that everybody can understand how to implement word representation learning and text classification using fasttext. Still if you get stuck somewhere, write in comments below or post on [Discussion portal](https://discuss.analyticsvidhya.com/) .