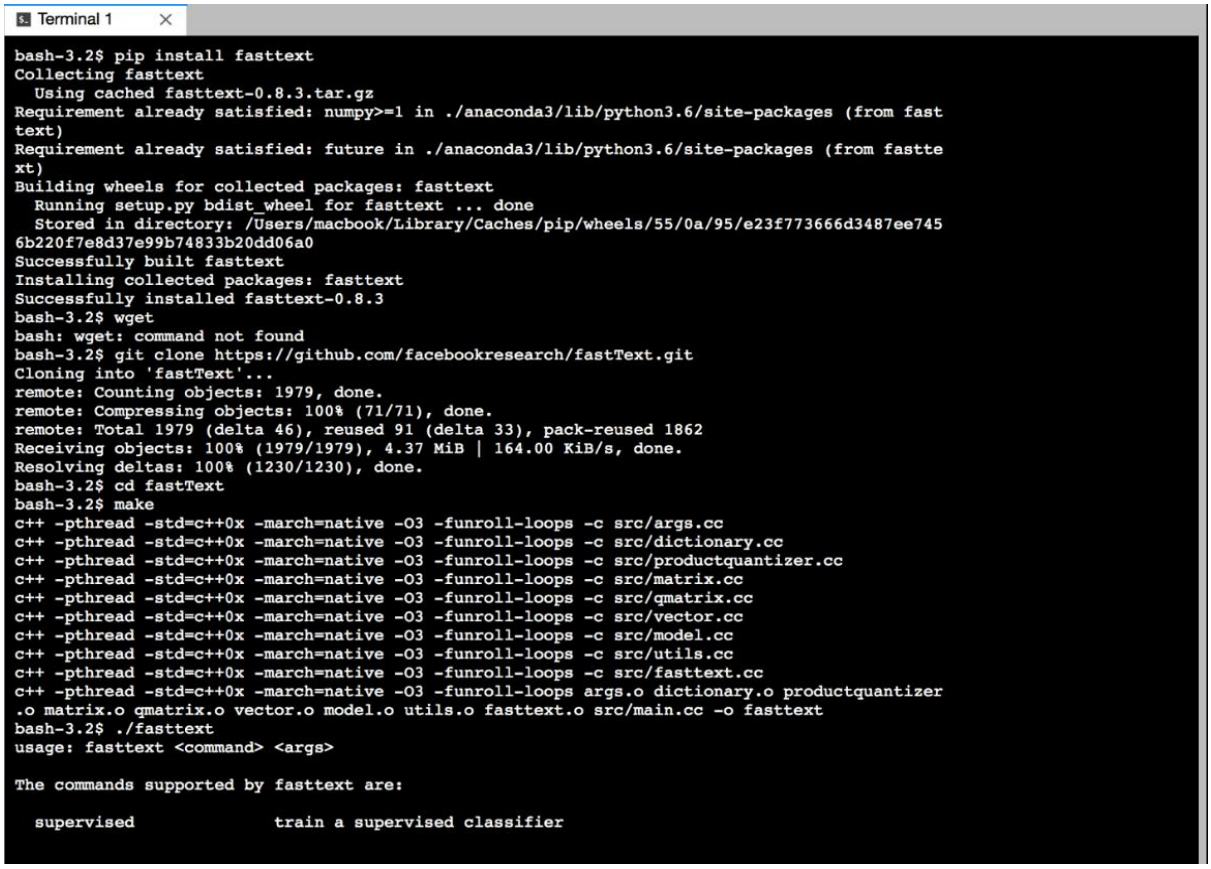
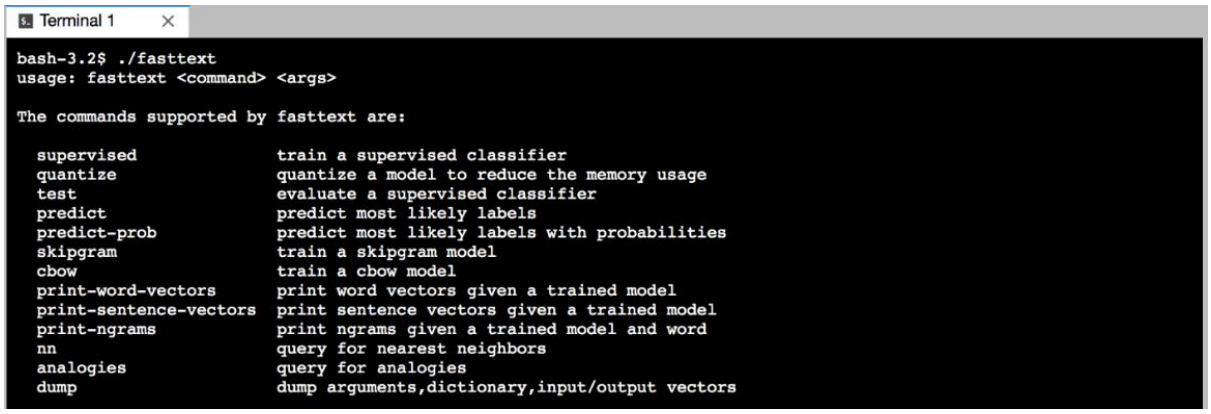
FastText for Text Classification

FastText is a library created by the Facebook Research Team for **efficient learning of word representations** and **sentence classification**. FastText differs in the sense that word vectors a.k.a 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 document is a stepwise walkthrough to study Fasttext using a cooking dataset of stack exchange ref: https://fasttext.cc .

1. Install FastText through pip in bash. If not pip, you can clone the library from git using this link: $ wget https://github.com/facebookresearch/fastText.git followed by $ cd fastText and $ make



2. Check if FastText has been installed properly and works, type the commands as: $ ./fasttext



3. FastText works on a specific type of data that needs to be labelled. Hence, we make use of such data from the cooking section of Stackexchange, and their associated tags using

$ curl -0 https://s3-us-west-1.amazonaws.com/fasttext-vectors/cooking.stackexchange.tar.gz && tar xvzf cooking.stackexchange.tar.gz

```
bash-3.2$ curl -O  https://s3-us-west-1.amazonaws.com/fasttext-vectors/cooking.stackexchange.
tar.gz && tar xvzf cooking.stackexchange.tar.gz
  % Total    % Received % Xferd  Average Speed   Time    Time     Time  Current
                                 Dload  Upload   Total   Spent    Left  Speed
100  446k  100  446k    0     0  65372      0  0:00:07  0:00:07 --:--:-- 84396
x cooking.stackexchange.id
x cooking.stackexchange.txt
x readme.txt
bash-3.2$ head cooking.stackexchange.txt
__label__sauce __label__cheese How much does potato starch affect a cheese sauce recipe?
__label__food-safety __label__acidity Dangerous pathogens capable of growing in acidic environments
__label__cast-iron __label__stove How do I cover up the white spots on my cast iron stove?
__label__restaurant Michelin Three Star Restaurant; but if the chef is not there
__label__knife-skills __label__dicing Without knife skills, how can I quickly and accurately dice vegetables?
__label__storage-method __label__equipment __label__bread What's the purpose of a bread box?
__label__baking __label__food-safety __label__substitutions __label__peanuts how to seperate peanut oil from roasted
 peanuts at home?
__label__chocolate American equivalent for British chocolate terms
```

Thus, the above data has been labelled with the text.

4. To classify this data, we need to split into train and validate. We count the number of examples using 'wc' command and split examples into train and validate. Thus 12000 examples are in train and 3000 in validate.

```
bash-3.2$ wc cooking.stackexchange.txt
   15404  169582 1401900 cooking.stackexchange.txt
bash-3.2$ head -n 12404 cooking.stackexchange.txt > cooking.train
bash-3.2$ tail -n 3000 cooking.stackexchange.txt > cooking.valid
```

5. Train this classifier using $ ./fasttext supervised -input cooking.train -output model_cooking

```
bash-3.2$ ./fasttext supervised -input cooking.train -output model_cooking
Read 0M words
Number of words:  14543
Number of labels: 735
Progress: 100.0% words/sec/thread:   66078 lr:  0.000000 loss:  9.966440 ETA:   0h 0m
```

6. Let us run this over validation data to test if this model works well.

```
bash-3.2$ ./fasttext test model_cooking.bin cooking.valid
N       3000
P@1     0.152
R@1     0.0656
Number of examples: 3000
```

One major point to be noted here is that this data was not pre-processed and thus we get precision and recall as above. We can improve the performance of the model by applying some pre-processing techniques like converting into lowercase etc and again repeat the same process.

```
bash-3.2$ cat cooking.stackexchange.txt | sed -e "s/\([.\!?,'/()]\)/ \1 /g" | tr "[:upper:]" "[:lower:]" > cooking.p
reprocessed.txt
bash-3.2$ head -n 12404 cooking.preprocessed.txt > cooking.train
bash-3.2$ tail -n 3000 cooking.preprocessed.txt > cooking.valid
bash-3.2$ ./fasttext supervised -input cooking.train -output model_cooking
Read 0M words
Number of words:  8952
Number of labels: 735
Progress: 100.0% words/sec/thread:   71676 lr:  0.000000 loss:  9.961704 ETA:   0h 0m
bash-3.2$  ./fasttext test model_cooking.bin cooking.valid
N       3000
P@1     0.178
R@1     0.0771
Number of examples: 3000
```

Precision has been improved by 4% approx. Let us see if further this can be improved by adding epochs. By default, fastText sees each training example only five times during training, which is small, given that our training set only have 12000 training examples. These number of examples can be increased using an epoch (set to 25) option as follows:

```
bash-3.2$ ./fasttext supervised -input cooking.train -output model_cooking -epoch 25
Read 0M words
Number of words:  8952
Number of labels: 735
Progress: 100.0% words/sec/thread:   72038 lr:  0.000000 loss:  7.195977 ETA:   0h 0m
bash-3.2$ ./fasttext test model_cooking.bin cooking.valid
N       3000
P@1     0.517
R@1     0.223
Number of examples: 3000
```

This gives us surprising results as precision goes up by almost 60%.

7. Another way, learning rate can also be used to improve the performance. This corresponds to how much the model changes after processing each example. A learning rate of 0 would means that the model does not change at all, and thus, does not learn anything. Good values of the learning rate are in the range 0.1 - 1.0.

```
bash-3.2$ ./fasttext supervised -input cooking.train -output model_cooking -lr 1.0 -epoch 25
Read 0M words
Number of words:  8952
Number of labels: 735
Progress: 100.0% words/sec/thread:   72322 lr:  0.000000 loss:  4.320680 ETA:   0h 0m
bash-3.2$  ./fasttext test model_cooking.bin cooking.valid
N       3000
P@1     0.585
R@1     0.253
Number of examples: 3000
```

The precision goes on increasing with change in parameters.
The other available parameters that can be tuned are –

- **-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]

This way we can improve the performance of this model by adding parameters. FastText outputs results quickly but it has a limitation of data to be in a very specific way i.e. it should be in the labelled format as

_ _ label _ _ <X> <Text>