Natural Language Processing
Lab Week 5: FastText

fastText (https://pypi.org/project/fasttext/) is a library for efficient learning of word representations and sentence classification.

In this Lab, you will train and test FastText text classifier mainly using the `train_supervised`, which returns a model object, and call `test` and `predict` on this object.

Data

We are interested in building a classifier to automatically recognize the topic of a stackexchange question about cooking. Let's download examples of questions from the cooking section of Stackexchange (https://cooking.stackexchange.com/), and their associated tags:

```
>> wget
https://dl.fbaipublicfiles.com/fasttext/data/cooking.stackexchange.tar.gz
&& tar xvzf cooking.stackexchange.tar.gz
>> head cooking.stackexchange.txt
```

Each line of the text file contains a list of labels, followed by the corresponding document. All the labels start by the `__label__` prefix, which is how fastText recognize what is a label or what is a word. The model is then trained to predict the labels given the word in the document.
Before training our first classifier, we need to split the data into train and validation. We will use the validation set to evaluate how good the learned classifier is on new data.

```
>> wc cooking.stackexchange.txt
```

Our full dataset contains 15404 examples. Let's split it into a training set of 12404 examples and a validation set of 3000 examples:

```
>> head -n 12404 cooking.stackexchange.txt > cooking.train
>> tail -n 3000 cooking.stackexchange.txt > cooking.valid
```

Installation

FastText builds on modern Mac OS and Linux distributions. Since it uses C++11 features, it requires a compiler with good C++11 support. You will need Python (version 2.7 or ≥ 3.4), NumPy & SciPy and pybind11.
To install the latest release, you can do :

```
$ pip install fasttext
```

Calling the help function will show high level documentation of the library:

```
>>> import fasttext
>>> help(fasttext.FastText)
```

System Implementation

Preprocessing the data

Looking at the data, we observe that some words contain uppercase letter or punctuation. One of the first step to improve the performance of our model is to apply some simple pre-processing. A crude normalization can be obtained using command line tools such as `sed` and `tr`:

```
>> cat cooking.stackexchange.txt | sed -e "s/\([.\!?,'/()]\)/ \1 /g" | tr "[:upper:]"
"[:lower:]" > cooking.preprocessed.txt
>> head -n 12404 cooking.preprocessed.txt > cooking.train
>> tail -n 3000 cooking.preprocessed.txt > cooking.valid
```

Let's train a new model on the pre-processed data.

```
model = fasttext.train_supervised(input="cooking.train")
```

By default, fastText sees each training example only five times during training, which is pretty small, given that our training set only have 12k training examples. The number of times each examples is seen (also known as the number of epochs), can be increased using the `-epoch` option:

```
>>> model = fasttext.train_supervised(input="cooking.train", epoch=25)
```

You can improve the performance of a model by using word bigrams, instead of just unigrams. This is especially important for classification problems where word order is important, such as sentiment analysis.

```
>>> model = fasttext.train_supervised(input="cooking.train", lr=1.0,
epoch=25, wordNgrams=2)
```

You need to apply the different training configuration explained above and compare the results by evaluating these systems using the following command:

```
>>> model.test("cooking.valid")
```

The output are, respectively, the number of samples, the precision and the recall.