A REPORT

ON

“FASTTEXT PROJECT - GROUP E”

CONTRIBUTORS

Nandish Patel	45539510
Sai Kiran Ravula	45539146
Tarun Bandarupally	45534632

UNDER THE GUIDANCE OF
PROF. MARK DRAS

FACULTY OF SCIENCE AND ENGINEERING

MACQUARIE University
SYDNEY · AUSTRALIA

2019

ABSTRACT

In this report, we present a customized classifier for topic(s) detection based on the data provided from multiple domains. This classifier framework is based on a state-of-the-art text classifier; FastText, to label a set of questions requested by a user. The FastText-based entity has been trained, validated and tested with the help of a pre-labelled dataset outsourced from StackExchange in the environment of an Ubuntu 16.04 terminal. Information Security, Physics and Statistics were the subjects of analysis as they have a clear domain-distinction among themselves. This variety has been considered to verify whether the algorithm can handle them. A hundred and fifty thousand questions (fifty thousand from each) were stacked together, shuffled, and split into train, validation and test datasets. The StackAPI package in Python ensured efficient and successful extraction of the data. As a precautionary measure, the data has been pre-processed and dressed up for analysis by first restoring the identity of HTML escape characters; next isolating punctuations and finally, stripping questions with LATEX formulae. This yielded in a model with the following measures of accuracy: Precision :- 0.366 and Recall :- 0.122. Such low magnitudes of accuracies called out for the model to be fine-tuned. Automatic hyperparameter optimization didn't bring about a considerable change in the accuracies. With several combinations of tuning the parameters such as epoch, Learning Rate, size of word vectors, bucket, wordNgrams and the Loss function, the optimal model turned out to be the one with One vs All as the Loss function. The other parameter values were an epoch of 5, Learning rate of 0.5, Size of word vectors (dim) as 50, bucket value as 300000, and wordNgrams of 2 (Bigram). This catapulted the precision and recall of the test dataset to 0.619 and 0.207 respectively. While the model performed exceedingly well in labelling questions from individual domains, it slightly faltered when it encountered a mixed-domain question as it wouldn't have come across such kind of a question while training. These results indicate that tuning improves the accuracy considerably and its better to train the topics individually. This framework finds its direct application as a time- saver in acquiring labels of a question to quickly locate the topics when attempting an open-book Physics/Statistics examination.

Contents

List of Figures

Chapter 1

Introduction

Classification of text provides a fundamental solution to a variety of applications, such as identifying spam messages, analysing the sentiment of a certain text/message, web search and information retrieval etc. The main objective of it is to allocate a set of pre-defined multiple categories to a given document (like commodity reviews, electronic mails, text messages and so on). For instance, new blogs can be organized by their topics. While human annotation can be banked upon for precise labelling, machines cut down the expenditure and work efficiently on bigger datasets. The latter form of learning involves Natural Language Processing, which uses pre-labelled topics as examples while training. Fasttext, as a text classifier, stands out as it is an amalgamation of state-of-the-art algorithms in the field of NLP. It works quite well with huge data sources and a variety of languages. This classifier algorithm, for our analysis, has been customized to verify whether it successfully labels the topics from multiple domains such as Information Security, Physics and Statistics. These subjects are pre-labelled and available in StackExchange. [Code and other material can be found on our Github repository].

1.1 Replication Of Original Work

FastText has been built on modern Mac OS and Linux distributions. Since it uses C++11 features, it requires a compiler with good C++11 support. These include: (gcc-4.6.3 or newer) or (clang-3.3 or newer). We replicated the environment for Fasttext by running it on Ubuntu 16.04 and we used amazon ec2 instance to match the speed requirements; we installed gcc as a compiler with good C++11 support. As the environment is taken care of let's dive into data. Original tutorial was demonstrated on cooking data from StackExchange. So to replicate the work we used the same data source.

Data has been extracted from: Source using wget command. Data was first split into training and validation. The first classifier was built on the training data with default parameters. Next, we predicted a few labels from our model to compare the prediction with the original work for we achieved the same results. While testing, , we achieved the same scores for precision and recall for both single label and five labels.We then pre-processed the data as mentioned in the tutorial to improve the performance of the classifier. It includes clearing a few issues like punctuation marks, uppercase text. The pre-processed data was then divided to training and validation.

New classifier was built on the processed data with default parameters (as in the tutorial) for which we achieved results that were slightly better than the original work. Enhancement of the classifier was continued by modifying parameters like epoch, learning rate and both at the same time. We achieved results very similar to the original work at every step.

When we tried to enhance the classifier performance by altering wordNgrams feature we had a problem where the model was terminated throwing an error stating 'std: : bad alloc'. So, we skipped this part of replication in our project. Next step involved introducing a loss function to make the computation faster and this was exactly replicated as stated in the tutorial.

The final step in the tutorial was to predict labels which have a probability higher than 0.5, although our replication yielded the same labels, the probability of the labels was slightly higher in our case. We tested the precision and recall scores and they remained the same as they are in the original work.

Chapter 2

Data Collection

2.1 Problems Faced

The crucial part of this project was to gather new data using tools and technologies introduced to us in the lecture notes. The data has been collected with the help of web scraping technology. To begin with, we decided to gather data using BeautifulSoup – a Python package that provides multiple functionalities such as scraping data from StackExchange group of websites. This initial attempt to collect data was dropped, as web scraping the content from any website is not legal and many websites forbid it. Next, we researched for alternatives to collect data from StackExchange group of websites during which we discovered an ethical way to do it – Using StackAPI package available in Python. While collecting data we encountered several problems, some of them were as follows:

- **Problem 1** – Using StackAPI, we were only able to get limited number of questions along with their tags(labels). The core reason was limited number of hits to the API, to be precise 300.

- **Problem 2** – Our limited knowledge on StackAPI. There were several parameters to be explored in an API call for example fromdate, todate and many more.

- **Problem 3** – Terms and conditions for collecting the data using StackAPI were a plenty.

These problems have been addressed with the help of solutions provided in the StackAPI Documentation:

- **Solution 1** – We signed up for a developer account and created an APP on StackExchange. Using appropriate tokens and keys for authentication purposes, we were able to get 10,000 hits to the API.

- **Solution 2** – We have followed the above-mentioned tutorial to learn more about StackAPI. After exploring all the variables, we decided to use default settings of the API, as 'dates' could introduce a bias in the data.

- **Solution 3** – StackExchange allows its users to use their data for educational purposes.

2.2 Data Collection

We gathered 50,000 questions(text), along with their tags(labels) for each domain: Information security, Statistics and Physics from their respective StackExchange website. The data from the API was intially collected into separate .CSV files. Then, the individual .CSV files have been merged into a single .CSV file using Pandas library in Python, thereby containing 150,000 questions. This .CSV file was then shuffled using Sklearn library in Python to introduce randomness in the data. Next, the .CSV file was converted into a text file in a particular format as required by the FastText(Format - Labels of the text must be appended to the string "__label__"). For example:

```
__label__thermal-field-theory What are thermal photons?
```

Finally, the data was split into training(70%), validation(20%) and testing(10%) using simple Unix commands which are shown below.

```
head -n 105000 final_data.txt > data.train
tail -n 15000 final_data.txt > data.test
tail -n 45000 final_data.txt | head -n 30000 > data.valid
```

2.3 Data Preprocessing

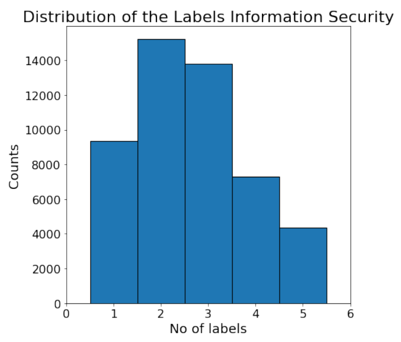
Data Preprocessing is one of the most crucial part of the data analysis. Real-world data is often incomplete, inconsistent, and/or lacking in certain behaviors or trends, and is likely to contain many errors. There is a saying in the data analysis domain that goes like "GARBAGE IN GARBAGE OUT", which roughly means if data on which you are applying analysis techniques is not up to quality standards, results from the analysis techniques will also be unreliable. We have done following preprocessing tasks:

- **Task 1** – The first task was to handle HTML escape characters because APIs often scrape data from the web pages. For example HTML escape characters like &#39; &gt; &lt; which stands for ' (single quote), $>$ and $<$. For this task we have used function unescaped() from the "html" package available in Python.
- **Task 2** – The second task was to remove all the punctuation and special characters found in the question except "-", because it can be used in words like well-being, on-site and more. These words are called hyphenated compounds. To deal with this task we have used Regular Expression(regex package) in the Python.

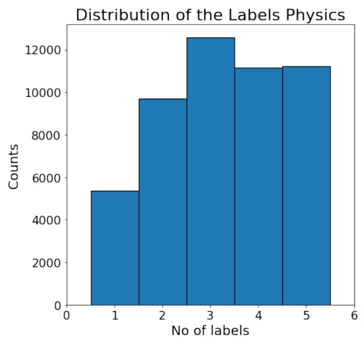
- **Task 3** – The third task was to remove all the LATEX formulas found in the questions from the STATISTICS as well as in the PHYSICS domain. For example, \$x^2\$ which means x^2. We have used Regular Expression(regex package) in the Python.
- **Task 4** – We have also converted all the questions into lower case.

2.4 Data Characteristics

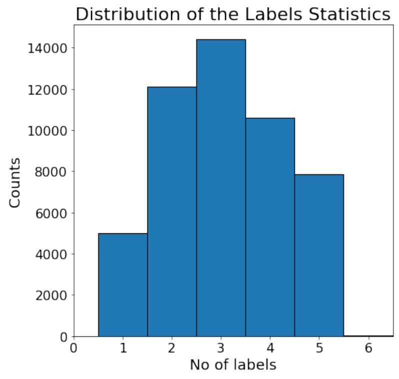
Our data has two parts – 1. Question(text) 2. Tags of the question(labels). Our data is multi-label data which means one question can have more than one label.
Distribution of the labels for each domain as well as combined data:



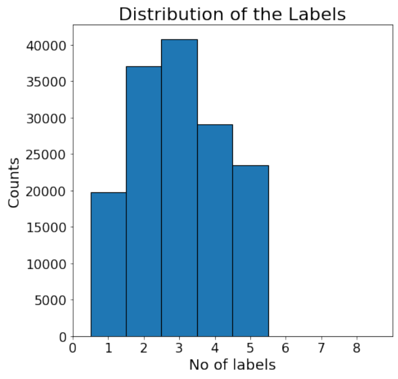
(a) Information Security



(b) Physics



(c) Statistics



(d) All data

Figure 2.1: Distribution of the labels

As we can see that for Physics, Statistics and All data, most of the questions have 3 labels(tags) while for Information Security most common number of labels are 2.

Chapter 3

Modelling

The most integral phase of our project was to train the model with the multi-domain stacked dataset. In our analysis, we mainly used ’supervised’, ’test’ and ’predict’ sub-commands; which correspond to learning (and using) the FastText text classifier. We built our first classifier on raw data using default parameters to mark it as a benchmark score for precision and accuracy.

The processed data was then fed to our classifier with the same default parameters and it yielded results which were higher than the benchmark score. This increase in statistical measures caught our interest and motivated us to check if the classifier could outperform its previous outcomes. So, we experimented by tuning various model features such as epoch, learning rate, wordNgrams, bucket, dim and loss function. Parameter tuning was performed in the similar lines of the FastText tutorial.

First trial was conducted by altering the epoch. We decided to test five levels of epoch varying from 5 to 25. Epoch is the total number of times a classifier views each training sample. By default, epoch value is set to five. There was a definite increase in precision and recall by altering the epoch.

Second trail was conducted on learning rate alteration. It varies from (0.0 to 1.0) where ’1.0’ corresponds to the classifier learning and an lr value of ’0.0’ suggests that there is nothing left for the model to learn. We tested five levels of learning rate; it was quite effective as the measures of accuracy improved with higher learning rates.

Next trail was conducted by altering both epoch, learning rate. An optimal trade-off between these parameters can be achieved by continuously monitoring the scores of precision and recall. There is a slight improvement in their values by this approach. Although we achieved significant scores, we still believed that we could improve the performance of the classifier by modifying the the other hyperparameters such as wordNgrams, bucket, dim and loss function.

Next iteration of performance enhancement was conducted by altering the values of word-Ngrams which refers to concatenation of any n consecutive tokens. Its value ranges from 1 to 5. We further continued by modifying the ranges of bucket from (100000 to 200000) and dim from (50 to 250). We finally arrived at the best classifier with great measures of precision and recall; significantly better than the benchmark scores. But it still felt mediocre, so we continued the enhancement process.

Since we have a huge dataset, we tried to improve the classifiers' computing capability by using the loss function argument. We firstly used hierarchical softmax (hs) as the loss function as it proved to approximate softmax with a much faster computation speed. We achieved results quicker than usual and there was a slight increase in the scores of precision and recall.

We used the last trick in the bag and tried enhancing the classifier by using the loss function 'one – vs – all'. To our blessing, our classifier achieved a precision rate of 0.619 and recall of 0.207, which proved to be the best figures our classifier could achieve.

We tried to compare the results of our best model with the results generated by using Fast-Text Automatic Hyperparameter Optimisation feature. This feature automatically chooses the optimal parameters for the classifier. Few problems using this autotune feature are:

- **Problem 1** - We couldn't find the autotune-validation parameter in the zip file provided from the tutorial. We had to download the latest version of the FastText from their Github Page as mentioned in Reference [3].
- **Problem 2** - Modelling failed initially due to insufficient time allocation. This was altered and we allocated 1800 seconds(30 minutes) for the model to run successfully by using auto-duration parameter.

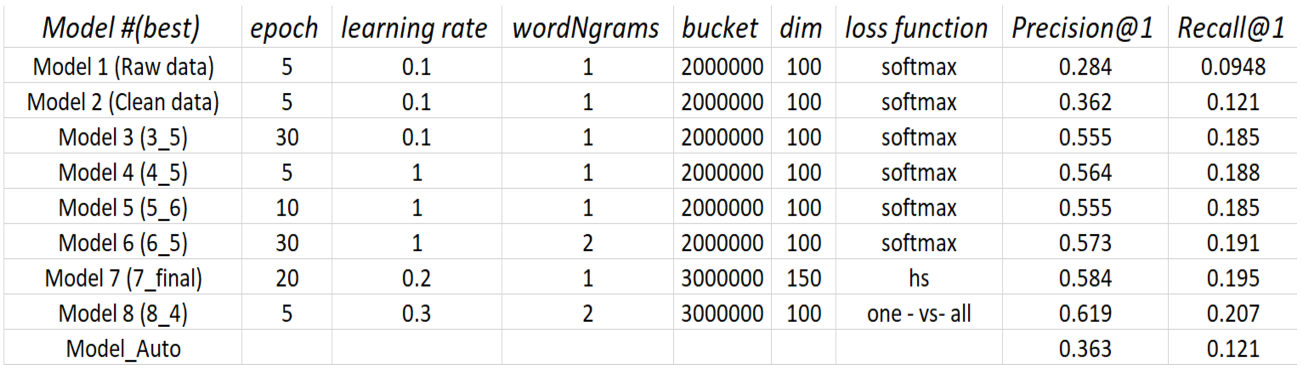
The results were obtained with precision of 0.363 and recall of 0.121. It proved that Automatic hyperparameter optimization feature doesn't work well with our dataset.

Chapter 4

Evaluation

A total of 9 models have been executed, for eight of which we tried a minimum of 6 to 7 variations by shuffling the hyper parameters (which can be found in our Github repository) and a model with autotune features was executed.

The results of the best variation in each model have been recorded and the results are tabulated as follows:



Model #(best)	*epoch*	*learning rate*	*wordNgrams*	*bucket*	*dim*	*loss function*	*Precision@1*	*Recall@1*
Model 1 (Raw data)	5	0.1	1	2000000	100	softmax	0.284	0.0948
Model 2 (Clean data)	5	0.1	1	2000000	100	softmax	0.362	0.121
Model 3 (3_5)	30	0.1	1	2000000	100	softmax	0.555	0.185
Model 4 (4_5)	5	1	1	2000000	100	softmax	0.564	0.188
Model 5 (5_6)	10	1	1	2000000	100	softmax	0.555	0.185
Model 6 (6_5)	30	1	2	2000000	100	softmax	0.573	0.191
Model 7 (7_final)	20	0.2	1	3000000	150	hs	0.584	0.195
Model 8 (8_4)	5	0.3	2	3000000	100	one - vs- all	0.619	0.207
Model_Auto							0.363	0.121

Figure 4.1: Model Evaluation

Chapter 5

Results & Conclusion

5.1 Results

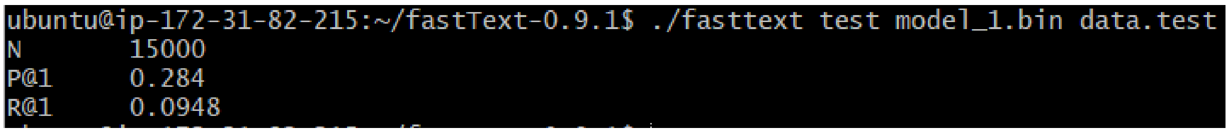
Among the nine models, Model 8 proved to be the best model with feature values: epoch = 5, learningrate = 0.3, wordNgrams = 2, bucket = 200000, dim = 100 and loss function = one - vs - all. The precision and recall scores turned out to be 0.619 and 0.207 respectively.

5.2 Conclusion

In this report, we proposed a simple baseline customization for text classification. The FastText algorithm when customized to handle a huge bank of data, turned out to be quite an efficient and swift text classifier. Multi-labelling has been successfully achieved with a precision of 61%. In spite of achieving such a high labelling-accuracy rate, the customized model performance slumped for questions with a mix of domains. Individual domain training and tuning apparently catalyses the accuracy measures. It can be applied in various fields; this entity can be utilized as a time- saver in labelling questions to quickly locate the topics when attempting an open-book Physics/Statistics examination.

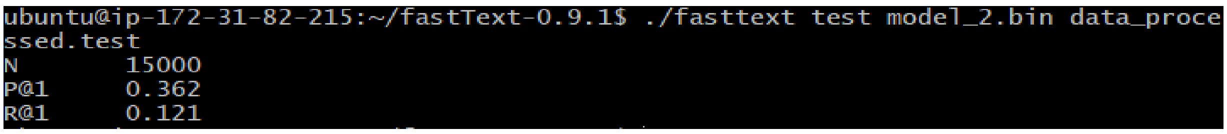
Chapter 6

Appendix



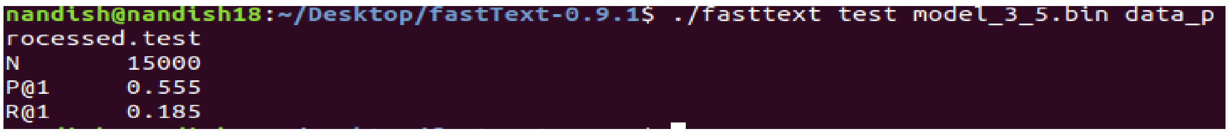
```
ubuntu@ip-172-31-82-215:~/fastText-0.9.1$ ./fasttext test model_1.bin data.test
N       15000
P@1     0.284
R@1     0.0948
```

Figure 6.1: Model 1 Before Preprocessing Default Parameters



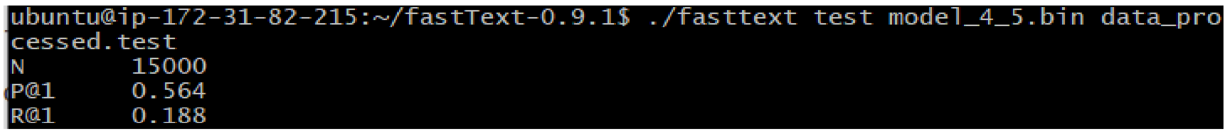
```
ubuntu@ip-172-31-82-215:~/fastText-0.9.1$ ./fasttext test model_2.bin data_proce
ssed.test
N       15000
P@1     0.362
R@1     0.121
```

Figure 6.2: Model 2 After Preprocessing Default Parameters



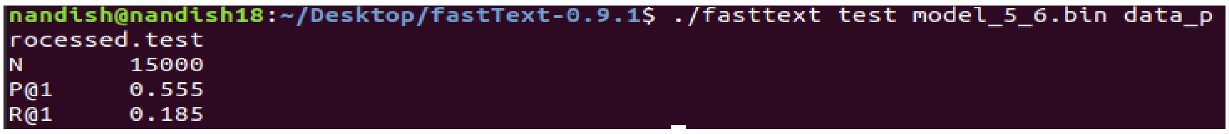
```
nandish@nandish18:~/Desktop/fastText-0.9.1$ ./fasttext test model_3_5.bin data_p
rocessed.test
N       15000
P@1     0.555
R@1     0.185
```

Figure 6.3: Model 3 => Epoch 30 and Default Parameters



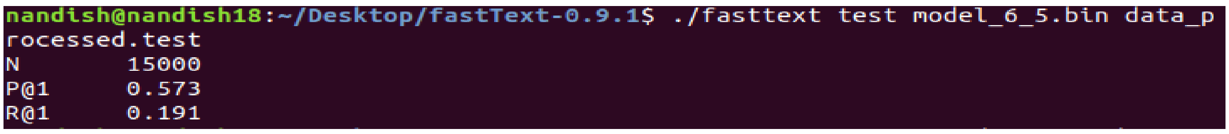
```
ubuntu@ip-172-31-82-215:~/fastText-0.9.1$ ./fasttext test model_4_5.bin data_pro
cessed.test
N       15000
P@1     0.564
R@1     0.188
```

Figure 6.4: Model 4 => lr 1.0 and Default Parameters



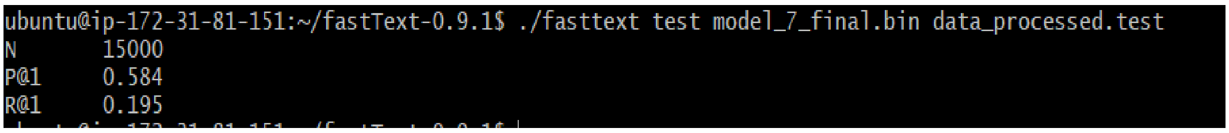
```
nandish@nandish18:~/Desktop/fastText-0.9.1$ ./fasttext test model_5_6.bin data_p
rocessed.test
N       15000
P@1     0.555
R@1     0.185
```

Figure 6.5: Model 5 => Epoch 10, lr 1.0 and Default Parameters



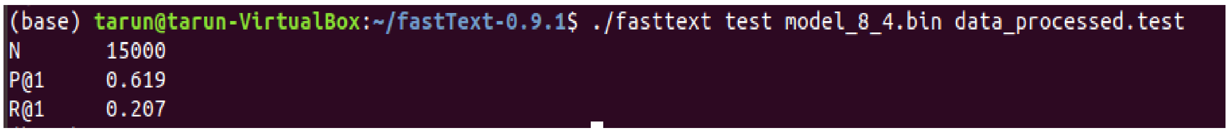
```
nandish@nandish18:~/Desktop/fastText-0.9.1$ ./fasttext test model_6_5.bin data_p
rocessed.test
N       15000
P@1     0.573
R@1     0.191
```

Figure 6.6: Model 6 => Epoch 30, lr 1.0, wordNgrams 2 and Default Parameters



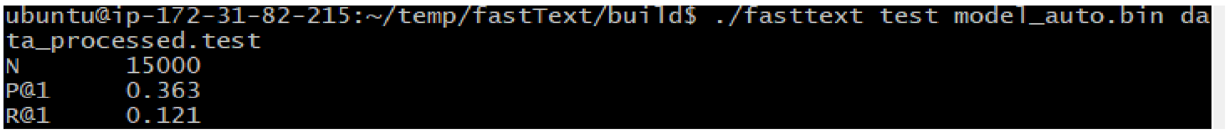
```
ubuntu@ip-172-31-81-151:~/fastText-0.9.1$ ./fasttext test model_7_final.bin data_processed.test
N       15000
P@1     0.584
R@1     0.195
```

Figure 6.7: Model 7 => Epoch 20, lr 0.2, Bucket 300000, dim 150, loss hs and Default Parameters



```
(base) tarun@tarun-VirtualBox:~/fastText-0.9.1$ ./fasttext test model_8_4.bin data_processed.test
N       15000
P@1     0.619
R@1     0.207
```

Figure 6.8: Model 8 => Epoch 5, lr 0.3, wordNgram 2, Bucket 200000, loss ova and Default Parameters



```
ubuntu@ip-172-31-82-215:~/temp/fastText/build$ ./fasttext test model_auto.bin da
ta_processed.test
N       15000
P@1     0.363
R@1     0.121
```

Figure 6.9: Model 9 Automatic hyperparameter optimization

References

[1] *https://fasttext.cc/docs/en/supervised-tutorial.html*

[2] *https://stackabuse.com/python-for-nlp-working-with-facebook-fasttext-library/*

[3] *https://github.com/facebookresearch/fastText*