UNSW

COMP 9417
FINAL PROJECT, AUGUST 2019

FAKE NEWS DETECTION
Topic 1.5

SUBMITTED BY
Manish Manandhar : z5174497
Oscar Fan: z5114679
Smrita Pokharel : z5125134
Yuta Sato : z5186797

For our project files please visit:
https://drive.google.com/drive/folders/1_Z0kXPt8_xKkutZ8cVDreC9mzH3qIS-k?usp=sharing

INTRODUCTION

Fake news has become an increasingly prevalent issue within the age of social media, as writers try to attract clicks with massively exaggerated clickbait headings.

In this report, we explore different machine learning algorithms and features to accurately classify the stance of an article. The article stance would be determined by comparing the article body with the article heading, with four possible classifications- unrelated, agree, disagree, discuss. We originally considered using multinomial naive bayes, SGD classifier, MLP, multiclass logistic regression, SVMs, extra tree classifier, random forest classifier, gradient boosting classifier, k-nearest neighbours, decision tree classifier, and LSTM. Beyond the features provided by the baseline model, we looked at a combination of count vectoriser, TF-IDF, and cosine similarity to transform text into vectors.

DATA ANALYSIS

The initial step was analysing the data, which was done using Carrot clustering Workbench Carrot Workbench (2019). Carrot Workbench is an easy to use data visualization tool and works really well with unstructured data like text.

In order to feed data into carrot, given CSV files were first converted to XML using a self written Python script, then the XMLs created were fed into Carrot Workbench for analysis.

We tried 3 clustering algorithms namely : K means, Lingo and STC on the training dataset which helped us gain insight into the content of the dataset provided. Carrot runs on JVM and it ran out of memory when we fed it entire training data set hence we created multiple batches containing 5000 news articles each. Then we tried all three clustering algorithms on training set from which we conclude that :

1. Both the training datasets from headlines and article bodies contained news about Apple iPhones, Mac book, Apple Watch, Steve Jobs, ISIS , Bear attacking humans, Justin Bieber, Heavy weapons, Doctors and so on.
2. Carrot workbench is easy to use, provides various hyper parameters on all different algorithms such as number of clusters, title boosting , Matrix Model, Term weighing and so on. Tuning these hyper parameters gave results on the fly.
3. Three types of cluster visualisation were tried:
    1. Aduna Clusters (Appendix)
    2. Free Foam Tree visualisation
    3. Donut chart (Appendix)

For each of these cluster visualization techniques, size of the cluster is directly proportional to the number of documents(in our case news articles) that it contained. Aduna clusters was able to show if a document belonged to more than one cluster. Apart from this, all three cluster visualizations gave us basic overview of training dataset.

Out of Kmeans, STC and Lingo, Lingo provided better results. While STC is faster, Lingo is more accurate and gives better clusters. The clusters from Kmeans were similar to that of STC and it was faster than Lingo but slower than STC.

CLUSTER ANALYSIS ON 3RD BATCH OF 5000 TRAINING DOCUMENTS (LINGO) (Foam Tree visualization)

Data Preprocessing

To preprocess our data we used two methods for different features. For TF-IDF and cosine similarity features we used Sklearn's TF-IDF Vectorizer's default preprocessor. For baseline features, we used the baseline methods of first cleaning each sentence using python's regular expressions library and only keeping words in our data. Then, each word was tokenized and converted to their lemmatized form using nltk. Then, stop words were removed from the sentence using sklearn's set of english stop words.

Dataset Split: We experimented with two training and cross validation set splits (80/20, and 60/40). After comparing the results, we decided that 60/40 provided the best performance.

Baseline Features

This baseline is composed of 4 features, it is defined and used in baseline models.

Polarity features

This feature checks if the number of refuting words in headline and article body is odd or even. Then, uses this information to vectorize the input document.

Refuting features

This feature checks whether "refuting words" exist in headline or not, then returns a vector of the input length containing 0 or 1 for the respective headline.

Word overlap features

This feature contains the probability of how many words are duplicated in concatenated text of headline and body.

So, the numerator is the number of overlapped words and the denominator is the number of total words in text.

Hand features

This features is list of frequency of how many n-grams (2~6) in title appear in body text as well. It counts how many times a token in the title appears in the body text.

Added Features

Count Vectors

Sklearn's CountVectorizer was used to extract the feature of the total number of word counts within a document. This feature was used naïvely to check the baseline performance metric of the Machine Learning algorithms used in this project. As expected, count vectors retained the lowest validation and test accuracy and competition score.

TF-IDF

An extension of counting vectors, TF-IDF (Text Frequency - Inverse Document Frequency) is a metric for determining how important a word is, in the context of a document, and its corpus. This is done by multiplying the number of occurrences of

word i in document j, then multiplying it by the logarithm of the total number of documents divided by the number of documents containing the word i.

$$w_{i,j} = tf_{i,j} \times \log\left(\frac{N}{df_i}\right)$$

TF-IDF allows for more insight into a word, and their uniqueness in the context of a corpus. Moreover, using sklearn Tfidf vectorizer, it was relatively quick to compute the tfidf vectors of the training, validation and test data. The drawback with tf-idf is that it doesn't capture the index or position of the word in the corpus like the n-gram models.

Cosine similarity

Cosine similarity is a measure of similarity between two vectors by computing their dot product and normalizing them. Cosine similarity basically computes the angle between two vectors in multi-dimensional space, which is a good metric to use in high-dimensional space rather than Euclidean distance. Similarly, cosine similarity also captures orientation of a vector.

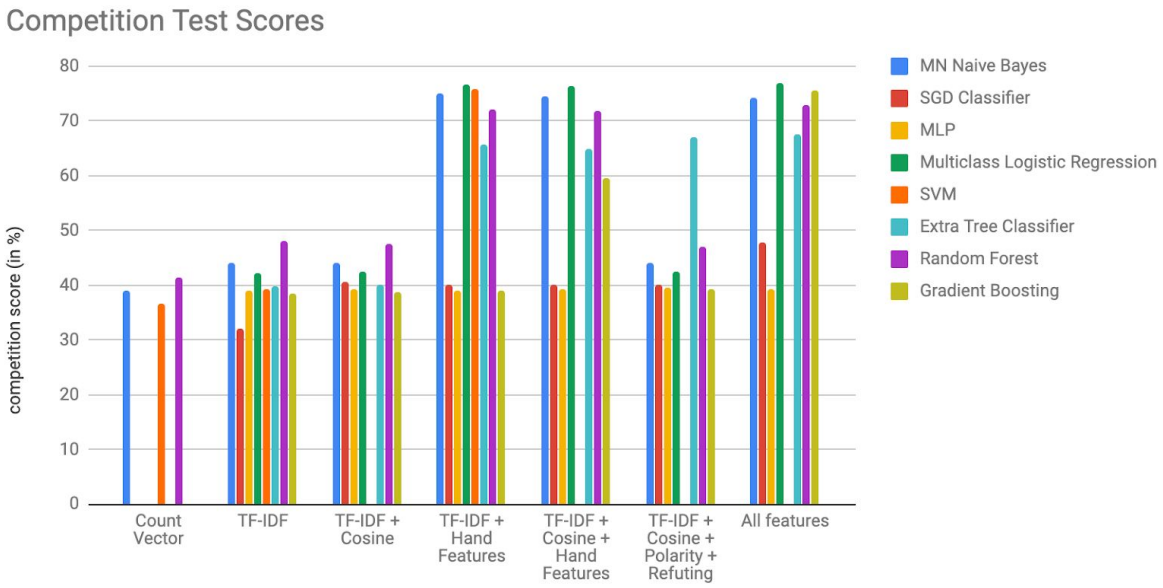
The advantage of cosine similarity is that it is extremely useful in NLP, where vectors tend to be extremely high dimensional, causing the Euclidean distance suffers from the curse of dimensionality. The drawback was the long computation time.

GloVe Word Embeddings

GloVe vectors are word vectors that were trained using unsupervised models in Stanford. The glove word vectors carry semantic information and linear substructures about the word vector space. We converted the gloVe file with 6 billion tokens, 400k vocabulary (Pennington 2014) with 50 dimensions for each word vector to a pickle for faster execution. We tried to use the word embeddings to train Multi-Layer Perceptron network, however, our computers ran out of RAM and Disk memory before the training could be complete given the large amount of data.

Glove word embeddings were also used to transform each word in the corpus to train a recurrent neural network with LSTM cells in batches of 64 instances and a droup-out wrapper of 0.60 to avoid overfitting. However, we were neither able to get any good training accuracy nor optimize the loss, using Adam and SGD optimizer.

Model Selection



We started by computing the accuracies by just using TF-IDF, and cosine similarity. However, as can be seen, this resulted in very low accuracies. Because of the unbalanced nature of the dataset we were provided with, even by solely predicting the 'unrelated' class, we would be able to almost obtain 40% (39.37%) accuracy. Therefore, it is clear that by themselves, TF-IDF and cosine similarity are not very useful, and had to be combined with baseline features in order to get the best results.

1. Multinomial Naïve Bayes

   Naïve Bayes Classifier is a classic algorithm for solving NLP problems. It uses Bayes theorem and outputs Class C for instance d to maximize the probability of words that appear in the text. (c_max= argmax P(c)P(d|c)) Compared to Bernoulli Naïve Bayse, multinomial Naïve Bayes focuses on frequency of same words in text and doesn't focus on the words which do not occur in sentences. Therefore, MNB should work better than Bernoulli because for news article there are many words not related headline and body. For parameters, we choose alpha(smoothing parameter) = 1.0, class_prior = None , fit_prior=True(Prior probabilities of the classes if this parameter is False ,it uses a uniform prior) For hand feature + tf-idf feature we change alpha to 10,but for other feature it did not show much improvement ,therefore we use default value.

2. SGD classifier

   Sklearn's Stochastic Gradient Descent is a discriminative linear classifier. We used L2 regularization and tuned the loss function parameter firstly with squared loss, then with perceptron. We also trained this model across combinations of several features listed above. However, the validation accuracy

and competition test scores were not very high as we expected, as the model overfitted very easily given the complexity of our data and the skewed distribution of news stance classes.

3. Multi-Layer Perceptron

    Multi-Layer Perceptron (MLP) from sklearn is a neural network classifier with 100 hidden units with relu activation functions and adam optimizer. The MLP classifier we used was a feed forward neural network that computed partial derivatives of the loss function to update its weights on each iteration. MLP classifier was also not able to learn the training data well, and gave below par accuracies for all the features. MLP for this training set was prone to overfitting as the validation and training score was quite high, however, the competition test score was very low.

4. Recurrent Neural Network (LSTM)

    Recurrent Neural Networks are a bit different than feedforward neural networks like the multi-layer perceptrons, as each word in a sentence depends on the appearances of the previous words and can encapsulate the information stored in the previous state. Long short term memory cells, with their input, output, memory and forget gates (Appendix Figure 3), have the capability to encapsulate larger memory than simple RNNs. Thus, we added 64 hidden LSTM units in our network with 4 output classes to predict the stance of the news article.

    Glove word vectors were first used to vectorize the words and sentences to vectors of dimensions (49972, 250, 50), where 49972 represents total training examples, 250 represents the dimensions of headline and article body and 50 represents the dimensions for each word vector. The training data were randomly picked on a batch of 500. Inorder to avoid overfitting, we also included a dropout wrapper with a dropout output keep probability of 60%. For loss optimization, we tried Adam optimizer and Stochastic Gradient Descent optimizer with various learning rates from 0.5 to 0.0001. Although we changed out hyperparameters a great deal, we couldn't stabilise the networks learning, neither could optimize the loss.

5. Multiclass Logistic Regression

    Multiclass logistic regression is a logistic regression algorithm that generalizes logistic regression to multiclass using cross entropy loss. Multiclass logistic regression was the best performer when TF-IDF and cosine similarity was combined with baseline features. We explored different solvers (lbfgs, sag, saga,

newton-cg) and varying max iterations from 100 to 1000 to try and optimise the model accuracy. What we discovered was that although almost all the solvers converged within the 200 iterations, lbfgs struggled to converge until almost 2100 iterations. This didn't result in an increase in accuracy either, as all the models had accuracies around 72% to 76% accuracy. Ultimately, the hyperparameters we decided on were the 'sag' solver with a max iterations of 1000, resulting in an accuracy of 76.88% on the competition test set.

6. SVM

SVM is also used for classification to work as margin maximization and try to get the best decision boundary.(Assume that decision boundary is f(x) = wx + b, maximize ‖ 2/w ‖ under constrain function y_i(y_i-b)≥1).SVM has been successfully applied to classify text and it should work well for fake news detection, however It takes more time and the result is not good compared to other models as a result. For one of the reasons, this fake news detection is categorized for 4 category and in general SVM is used for binary classification, so if used for more than 2 class, it would require some method (like pairwise method). The hyper parameter "C", which is the penalty parameter of the error or misclassification was tuned between 0.01 and 0.05, and was chosen to be 0.25.

7. Extra Tree Classifier

Extra Tree Classifier is an ensemble method that trains randomized decision trees on subsamples of data and uses their averages to improve training accuracy and avoid overfitting (Scikit-learn 2019). We trained our extra tree classifier with 100, 10, 1000 estimators and the split was made using gini index. Extra tree classifier was one of the best models that was able to predict the unseen data of the competition test with a score of almost 70%.

Various hyper parameters of Extra tree classifier like n_estimators (number of trees), max_depth (Depth of trees), min_sample_leaf were tuned. It was observed that accuracy and training time were directly proportional to n_estimators. However adding too many trees caused overfitting hence we decided on 100 which gave us 76.422% score on competition test data. The deeper the tree, the more information it can capture. Changing max depth from 10 to 100 with n_estimators= 10 increased the score from 41.55% to 72.16%.

8. Random Forest Classifier

Random forest is one of the ensemble learning that built a set of classification trees. One of the important attributes of random forest is to

randomly select sub data set from original train data and generate trees using bagging method.

(For example, If original data is (d1,d2,d3,d4,d5) then it can make decision trees using (d1,d2,d4,d4),(d1,d1,d1,d2) and so on).Finally, it combines each tree and shapes random forest model. Decision tree has characteristics to do overfitting easily, therefore random forest has the same characteristics. In this project, we split the original train data into sub train data and test data, so it has the possibility to overcome overfitting since in this case some train and test data is really similar compared to use new test set like competition test set.

We tuned various parameters of Random Forest like n_estimators (number of trees), max_depth (Depth of trees), min_sample_leaf. From observation we noted that increasing n_estimators increased accuracy and training time. However adding too many trees caused overfitting hence we decided on 100 which gave us 78.26% score on competition test data. The deeper the tree, the more information it can capture. Changing max depth from 10 to 100 with n_estimators= 10 increased the score from 65.11% to 74.49%.

9. Gradient Boosting Classifier

Gradient boosting is an intuitively simple to understand ensemble method, which tries to improve results by building multiple models, training solely on misclassified results. (Refer to figure 4 in appendix)

The gradient boosting classifier was used in the baseline with 200 estimators, which involved using polarity, refuting, and hand features, with an accuracy of 75.2%. When this was further combined with TF-IDF and cosine similarity, this resulted in an increase in accuracy to 75.31%. With just a 0.11% increase in accuracy, we decided to explore how tuning hyperparameters could improve performance. The main hyperparameters we tuned were the number of estimators (100 to 200), learning rate (0.01 to 10), and max depth (3 to 12). What we found was that the best performing hyperparameter was to have 120 estimators, with a learning rate of 0.1, and a max depth of 3, which managed to improve the accuracy to 75.62%. When the learning rate was too small (0.01), the classifier struggled to get more meaningful information from the training set. Whilst when the learning rate was too large (10), the classifier also struggled with accuracy, as it was overshooting the point where the loss function would be minimised. Interestingly, increasing max depth did not improve the accuracy of the model, which could be because the model has reached the limit of its capabilities. Gradient boosting had an incredibly time consuming training process which made it difficult to incrementally explore different combinations of hyperparameters.

K-Nearest Neighbors and Decision Trees

We had also originally considered potentially using K- Nearest Neighbours, but there were two main issues with using this method of classification. K- Nearest Neighbours suffers from the curse of dimensionality, meaning that in an extremely high dimensional problem like NLP, all vectors would have an extremely large Euclidean distance between them. Furthermore, although KNN requires no training, classification is done at runtime, and we perceived the classification time to be too much of a drawback.

Decision trees are a very intuitive and easy to understand form of classification. However, we decided against using a decision tree classifier, as, not only does it easily suffer from overfitting, we were also already using three tree ensemble methods, meaning it wasn't necessary to also test the decision tree model.

CONCLUSION

In final consideration, linear classifiers like SVM, SGD could not classify the high dimensional data very well. We can see that models trained on our additional features features like TF-IDF stacked with the baseline features perform better across all algorithms that we have used. Among single classifiers, Multiclass Logistic Regression and Multinomial Naïve Bayes performed relatively well. The ensemble methods of Extra Tree Classifier, Random Forests and Gradient Boosting also fared better than most non-ensemble classifiers.

Best Model: Multiclass Logistic Regression

FUTURE ENHANCEMENTS

Using word embeddings

Due to the high dimensionality of the data and limitation of resources we were unable to use word embeddings for the project. We tried using pretrained models from both Word2Vec and Glove on Colab. However we ran into various problems of it sometimes crashing or not saving our trained model because of which we decided to keep this as a future enhancement.

Rules engine

We analysed the documents that were misclassified (using Carrot workbench and confusion matrix) and were able to figure out that a lot of documents that had actual 'agree' class got misclassified into 'unrelated'. These documents belonging to 'agree' class had the first two and the last two sentences that were similar to the headline and most of them didn't have any polar or refuting words in the middle section. In the future

we can come up with a rule engine to give more weight to the first two and the last two sentences and could keep that a new feature vector for our machine learning models.

Challenges:

Dimensionality of the data: This slowed down the development process significantly. Getting cosine similarity features took a lot of time (Appendix: Table 1 : Training time for features). Similarly baseline features training took some time too. (Appendix: Table 1 : Training time for features). This was one of the major challenges for us as we carried our analysis on various dataset splits and features. However, we overcame this issue by serializing most of our features using Pickle (Python Package).

APPENDIX

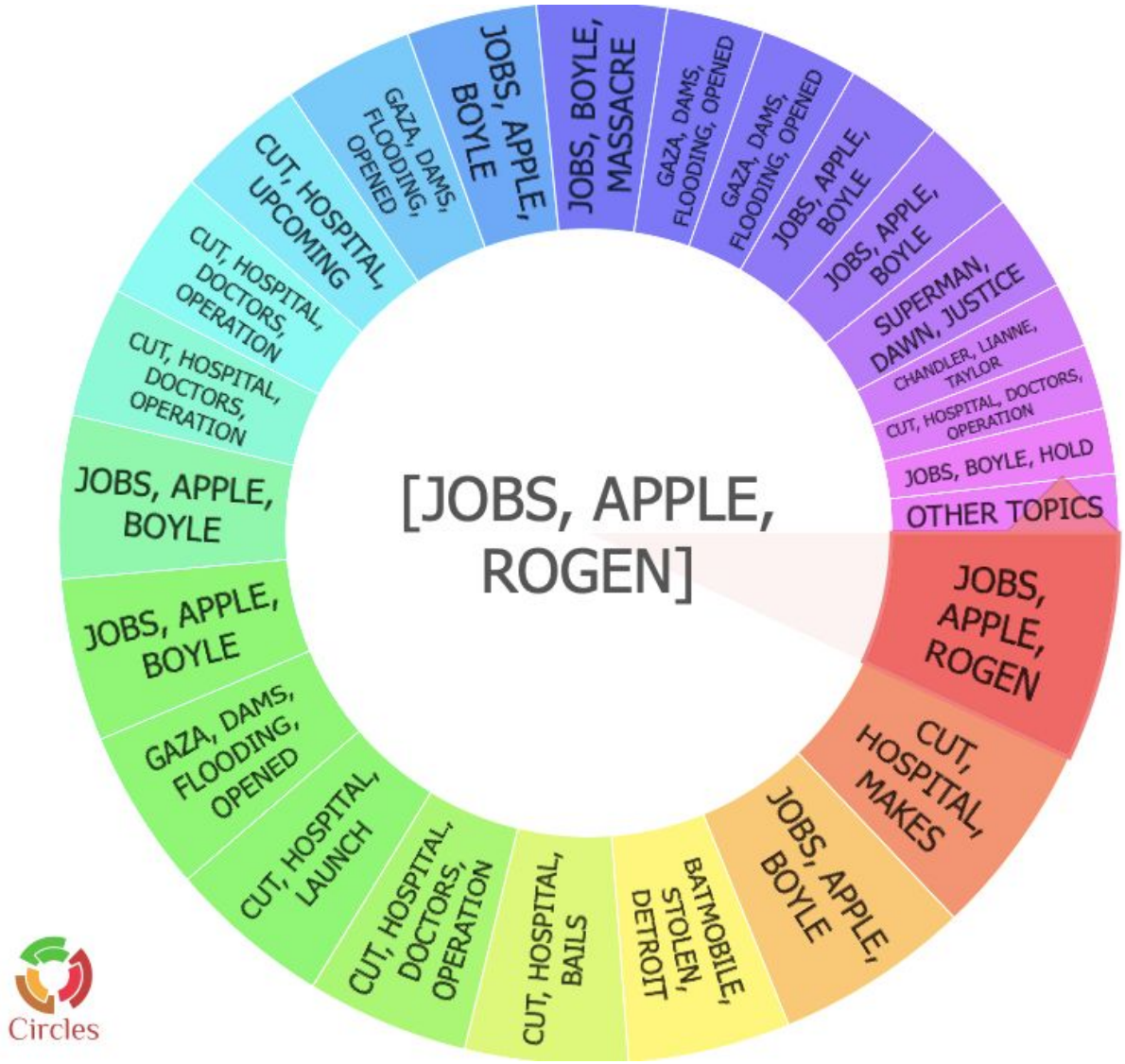


Fig 1 - Cluster Visualization on 100 documents from training set using Lingo (Donut cluster visualization)

S. N.	Feature	Training time Training set	Training time Test set

1.	Cosine features:	4:04:27	8:08:15
2.	Hand Features :	02:18	2:30
3.	Polar features :	02:48	01:59

Table 1 : Training time for features

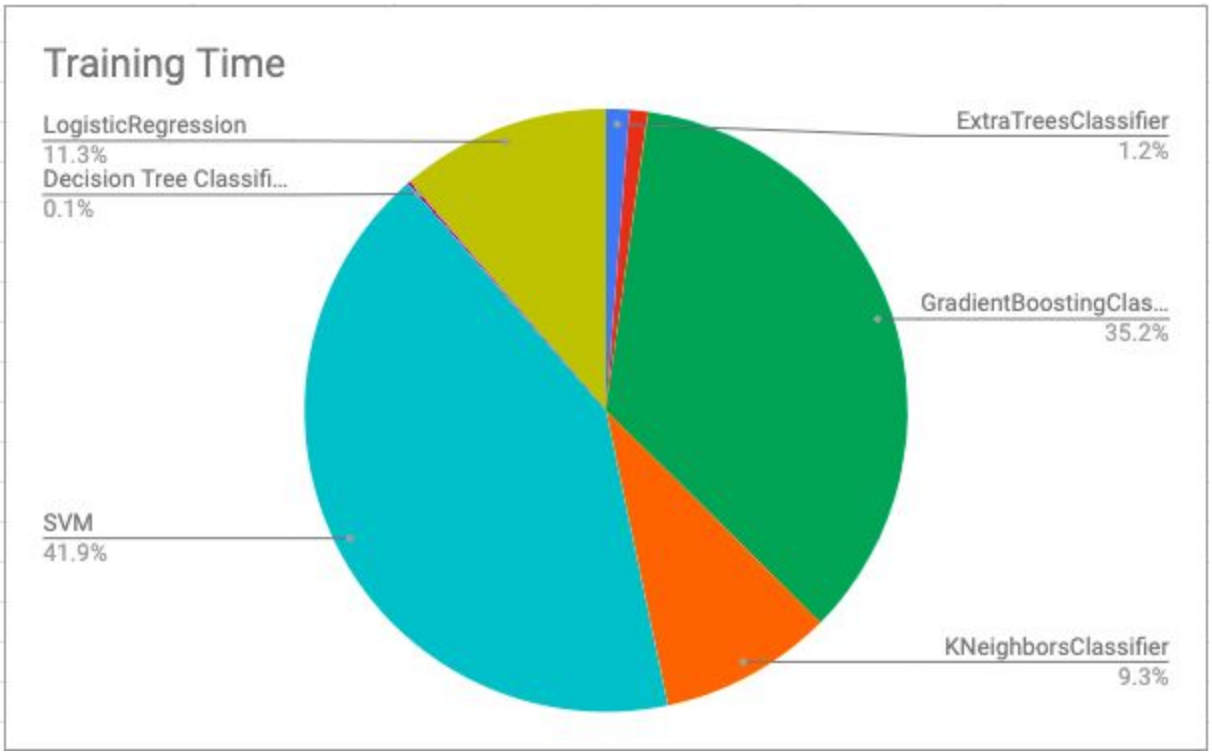


Fig 2 : Training time comparison for classification models

Carrot training data format

```
<searchresult>
   <query></query>
   <document>
      <title></title>
```

<url></url>
<snippet></snippet>
<document>
</searchresult>

LSTM Cell

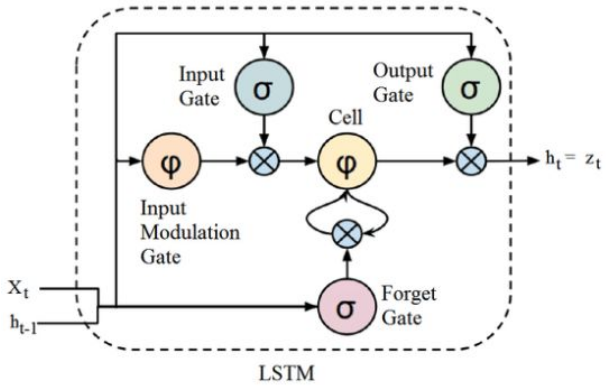


Figure 3: a single LSTM Cell (Kang, 2019)

Gradient Boosting

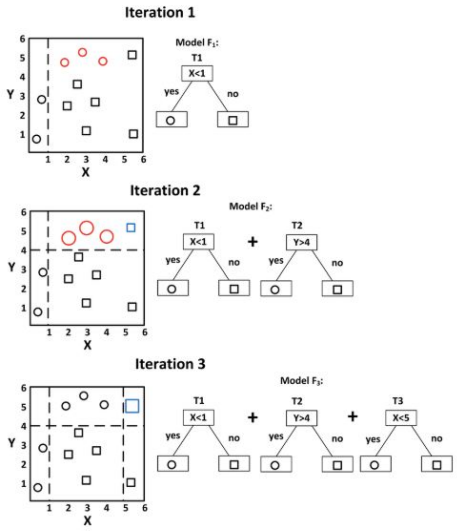
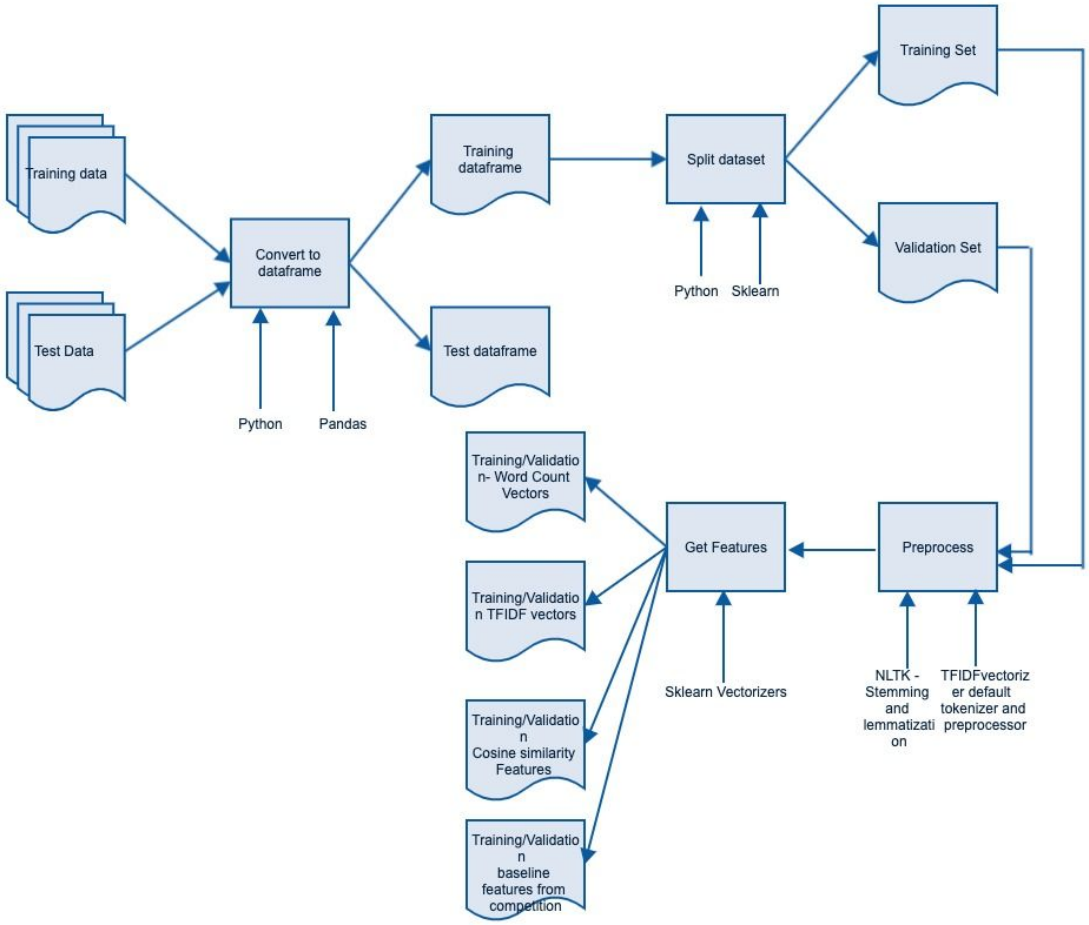
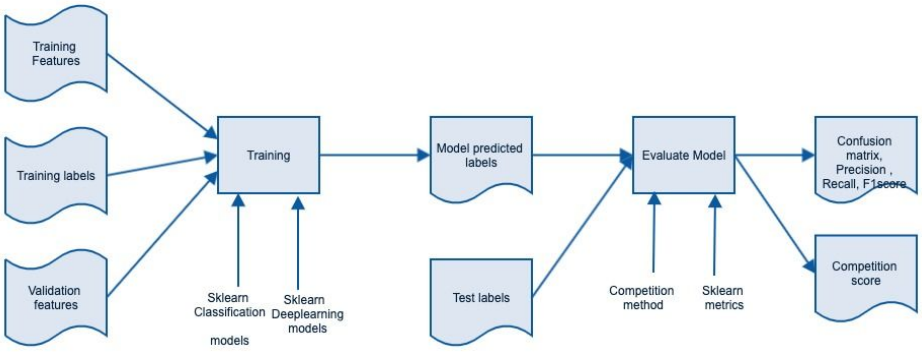


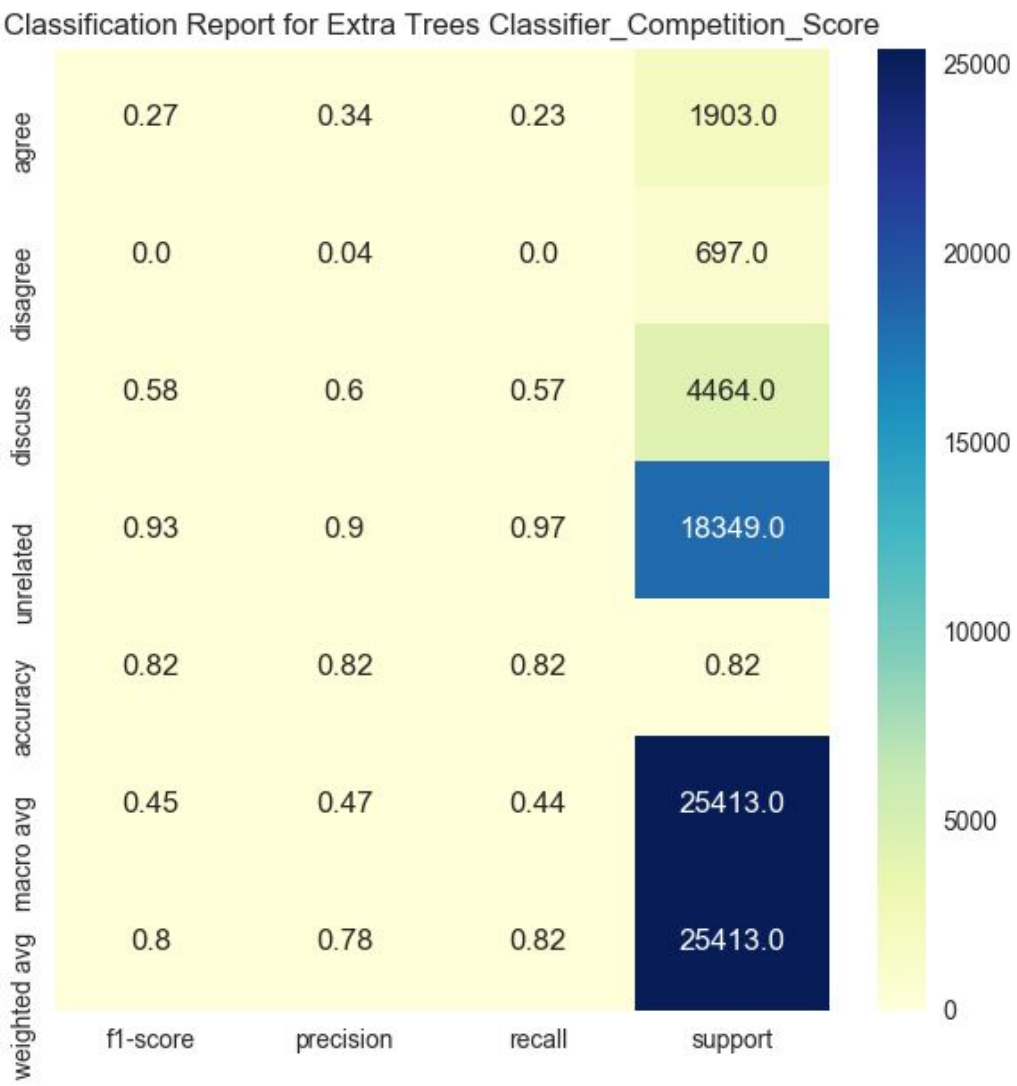
Figure 4: A simple explanation of Gradient Boosting

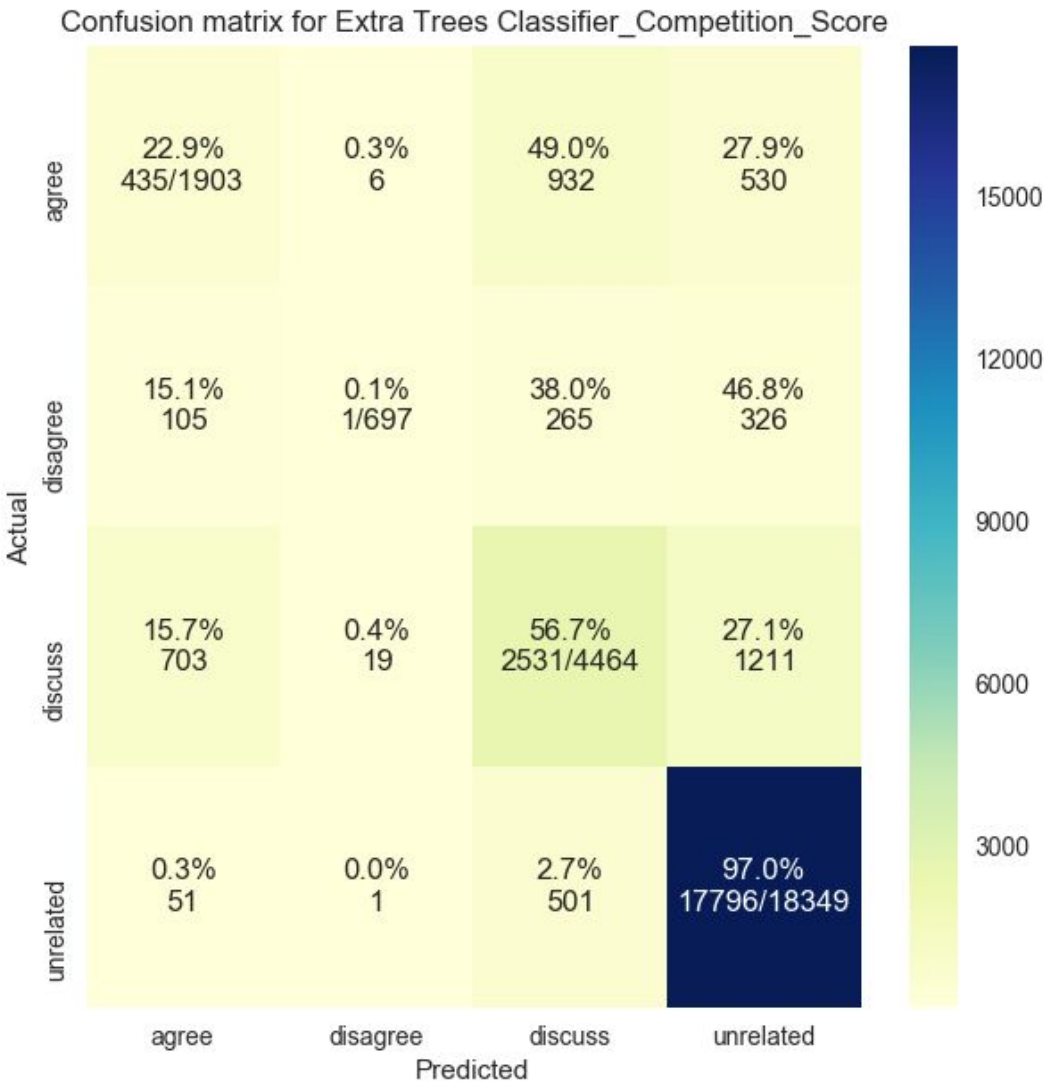
Preprocessing and feature extraction



Model training and evaluation




Classification Report for Extra Trees Classifier_Competition_Score
agree
disagree
discuss
unrelated
accuracy
macro avg
weighted avg
0.27
0.34
0.23
1903.0
0.0
0.04
0.0
697.0
0.58
0.6
0.57
4464.0
0.93
0.9
0.97
18349.0
0.82
0.82
0.82
0.82
0.45
0.47
0.44
25413.0
0.8
0.78
0.82
25413.0
f1-score
precision
recall
support
25000
20000
15000
10000
5000
0



Works Cited:

Pennington, J., Socher, R. and Manning, C. (2019). GloVe: Global Vectors for Word Representation. *Stanford University*. [online] Available at: https://nlp.stanford.edu/pubs/glove.pdf [Accessed 1 Aug. 2019].

Kang, E. (2019). *Long Short-Term Memory (LSTM): Concept*. [image] Available at: https://medium.com/@kangeugine/long-short-term-memory-lstm-concept-cb3283934359 [Accessed 10 Aug. 2019].

Baoxun Xu,Xiufeng Guo,Yunming Ye,Jiefeng Cheng.(2012). An Improved Random Forest Classifier for Text Categorization. JOURNAL OF COMPUTERS, VOL. 7, NO. 12.http://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.300.5687&rep=rep1&type=pdf [Accessed 10 Aug. 2019].

Carrot Workbench (2019). Available at: https://project.carrot2.org/download.html [Accessed 3rd July 2019]

Scikit-learn. (2019). *3.2.4.3.3. sklearn.ensemble.ExtraTreesClassifier — scikit-learn 0.21.3 documentation*. [online] Available at: https://scikit-learn.org/stable/modules/generated/sklearn.ensemble.ExtraTreesClassifier.html [Accessed 10 Aug. 2019].

Fraj, M. (2019). *In Depth: Parameter tuning for Gradient Boosting*. [online] Medium. Available at: https://medium.com/all-things-ai/in-depth-parameter-tuning-for-gradient-boosting-3363992e9bae [Accessed 11 Aug. 2019].