Machine Learning

Project Report

Task : Help Boost Our Online Reach
An NLP Based Task

By Team Judo,
- Dhamodhar Reddy (IMT2019026),
- Jayanth Ayyadevara (IMT2019038).

CONTENTS

1 PROBLEM STATEMENT

Consider yourself to be a consultant for an online advertising agency. The agency spends a considerable amount of time and money to find the best web pages to publish their ads on. They select web pages that will generate prolonged online traffic so that their ads can have a long-lasting reach.

Now, wouldn't it be great if you could somehow automate this process and save company resources? To facilitate this, the agency has created a dataset of raw html, meta statistics and a binary label for each webpage. The binary label represents whether the webpage was selected for ad placement or not.

The aim of this task is to identify the relevant, high-quality web pages from a pool of user-curated web pages, for the identification of "ad-worthy" web pages. The challenge requires you to build large-scale, end-to-end machine learning models that can classify a website as either "relevant" or "irrelevant", based on attributes such as alchemy category and its score, meta-information of the web pages and a one-line description of the content of each webpage. This task aims to gently introduce you to the domain of NLP, as you would be required to convert the string attributes of the dataset to some form of numerical data, and then construct your ML models on this numerical data.

2 EXPLORATORY DATA ANALYSIS

There are a total of 5915 data points and 27 features in the given data of which 3 features (url, webpageDescription and alchemy_category) are text based and the remaining features are numerical.

Also we have a pretty balanced data i.e the label/target feature is almost equally distributed (48% zeroes and 52% ones). Hence we need not take care of any imbalance in the data.

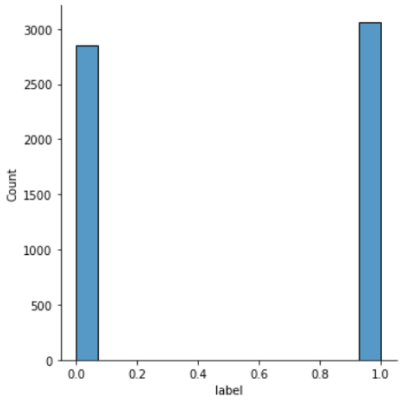


Figure 1: Distribution of labels in the given data

Of the 3 text based features, alchemy_category is a categorical feature, and hence is converted to numerical columns by one hot encoding.

3 FEATURE SELECTION AND ENGINEERING

3.1 Extracting the body and title features

Now of the remaining two features the webPageDescription feature is an object consisting of title, body and url as its constituent properties. So we can separate and treat title and body as individual data features. (url is already present as a feature beforehand). So together, we applied the NLP techniques to 3 features namely url, title and body.

3.2 Dropping unimportant features

3.2.1 Drop WebPageDescripation

Now that we have extracted the title and body from the webPageDescription feature, we no longer need the webPageDescription feature. Hence we drop it.

3.2.2 Drop framebased

The given data consists of a numerical feature called framebased which consists of only zeros for all the data points. Since this gives us no useful information about the datapoints, we simply drop the framebased feature.

3.2.3 Drop id

The given data conists of an id feature which can be useful for fetching the html content that was provided along with the data, but other than that we have no use with it while training the model as it does not give any relevant information about a website. So we drop the id feature.

3.3 Replacing Null/Missing values

3.3.1 Null values in alchemy_category and alchemy_category_score

The feature 'alchemy_category' has 459 missing values, which are replaced with the pre-existing category 'unknown'. Same with the 'alchemy_category_score' , except this is a numerical value and hence its missing values are replaced with corresponding scores of 'unknown' (0.40001) alchemy_category feature.

3.3.2 Missing values in 'isNews'

The feature 'isNews' has 2269 missing values. The 'isNews' values for data points which have 'isNews' as missing values and 'isFrontPageNews' as 1, are replaced with 1.(If it is a Front page news, then it is news). Remaining missing values are replaced with 0's.

3.3.3 Missing values in 'isFrontPageNews'

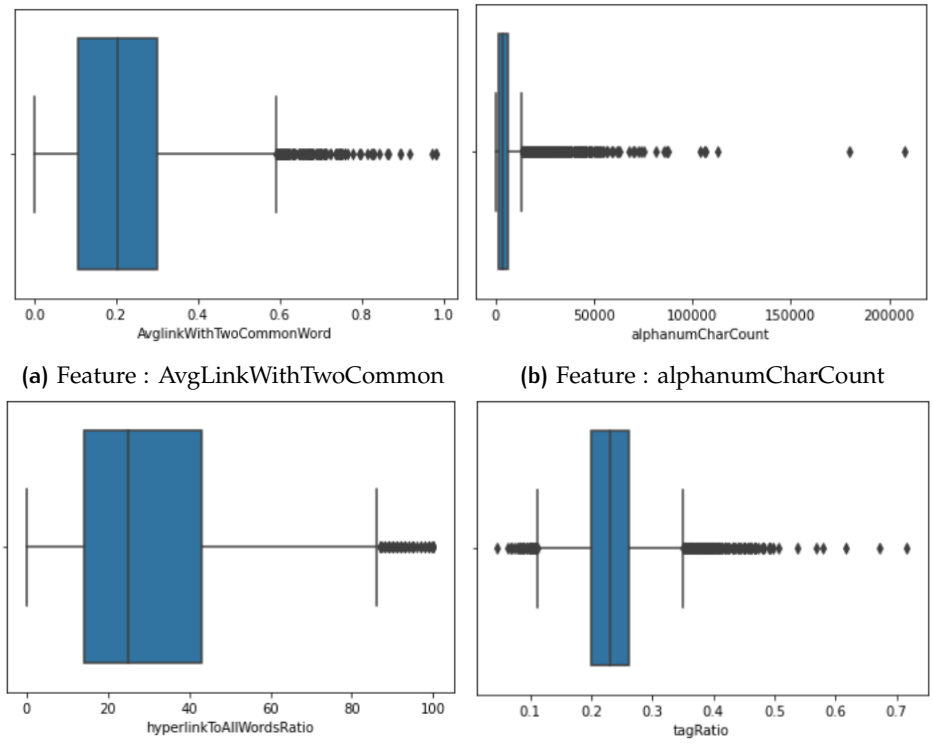
The feature 'isFrontPageNews' has 1006 missing values. The 'isFrontPageNews' values for data points which have 'isFrontPageNews' as missing values and 'isNews' as 0, are replaced with 0. (If it is not news, then it can't be Front page). Remaining missing values are replaced with 0's.

3.3.4 Missing values in 'title' and 'body'

There are also missing values in 'title' and 'body' features. We have replaced them with their corresponding domain name sliced from their url, as it will give a better idea about the feature than a null value.

3.4 Outliers

We visualized the data through a boxplot, We then found that few numerical feauters had outliers. We dropped few datapoints of some of the the features.



(a) Feature : AvgLinkWithTwoCommon

(b) Feature : alphanumCharCount

(c) Feature : hyperlinkToAllWordsRatio

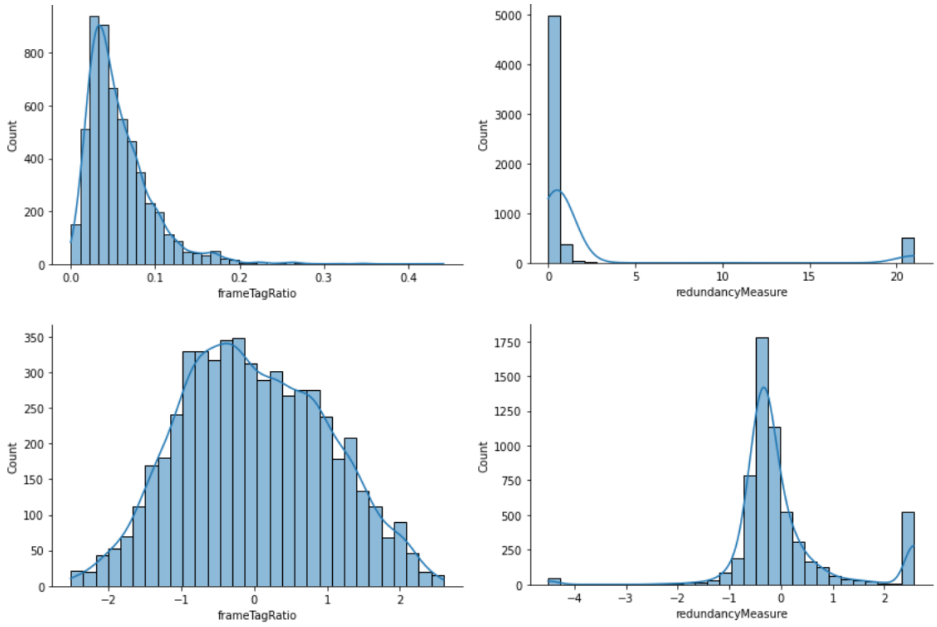
(d) Feature : tagRatio

Figure 2: Outliers in few feautures

Few more features had outliers but there were many of them, so we were not sure whether these were genuine errors in the data or not. That led us to believe it was the skewness that resulted in these many outliers and went in to check the skewness of the features.

3.5 Removing Skewness from few features

Few numerical features were 'right skewed' when visualized through a histogram. We used $'yeo-johnson'$ (since we have a non-negative i.e there were zeros present in the data) transformation using sklearn's power transform to get rid of the skewness. The features which were left skewed and can be transformed were $AvglinkWithTwoCommonWord$, $AvglinkWi$ $AvglinkWithFourCommonWord$, $redundancyMeasure$, $frameTagRatio$, $parametrizedLinkRatio$.



(a) Feature : FrameTagRatio **(b)** Feature : redundancyMeasure

Figure 3: Skewness in few feautures

3.6 Normalizing numerical data

All the numerical features were then normalized using the standard scaler from sklearn.preprocessing package.

4 NATURAL LANGUAGE PROCESSING

So now that we have done the basic preprocessing, let's move into the more interesting part, that is working with text based data.

4.1 Removing punctuations, lowercase every character

For the better processing of data, we remove any form of punctuations from the data since we want only the words along with lowercasing any character we see to maintain uniformity between each word and classify both the uppercase and lowercase words as a single word. Also the tokenization (converting string of words to list of words for ease in accessibility of each word) is done along with this.

4.2 Splitting alpha-numeric words

The data had many alphanumeric words which were present as a single word. For example, 100grams instead of 100<space>grams. If we do not separate these words, then our

model cannot see any resemblance between 100grams and 200grams since they appear as two completely different words for the vectorizer.

4.3 Removing stopwords

We must remove the words which are highly abundant in every document. By removing these words, we remove the low-level information from our text in order to give more focus on the important information.

4.4 Lemmatization

We want to map the words 'created' and 'creates' as a single entity 'create'. To achieve this we need lemmatization. There are a bunch of different vectorizers to start working with. We started with a basic wordnet lemmatizer without using any of the parameters. This was not doing the job upto the mark.

Many words are not yet classified into a single group. Later we used a lemmatizer from the Spacy library. But this lemmatizer took a lot of time to process. Hence we just stick to the wordNet lemmatizer. But we figured out that we cannot use the same lemmatization techniques for all parts of speech. Hence we lemmatize each parts of speech accordingly. (Achieved using WordNet lemmatizer and setting the params accordingly)

4.5 Vectorization

Now the final step is to vectorize these processed words. Again we started with a basic count vectorizer but later we figured out that TF-IDF vectorizer is a better way to vectorize.

4.5.1 TFIDF Vectorizer

With TF-IDF vectorizer we can give importance to each word such that it gives us a way to associate each word in a document with a number that represents how relevant each word is in that document.

Before vectorizing, we set the parameters of the vectorizer such that in the final vectors there are only words which appear in at least 2 documents (min_df = 2) (If it appears in only one document, then this will not be any important info provider to train) and do not appear in more than half of the documents as these will be redundant info (max_df = 0.5).

We apply the same preprocessing methods mentioned above for each of the 3 text based features (body,url,title).

4.5.2 *Word2Vec Vectorizer*

Later we discovered a different vectorizer called the Word2Vec vectorizor is a group of related models that are used to produce so-called word embeddings. After training, word2vec models can be used to map each word to a vector of typically several hundred elements, which represent that word's relation to other words. We used a pre-trained model by google and mapped the words from each of our text features.

4.6 Principal Component Analysis

When vectorizing with the TF-IFD vectorizor, we are left with around 40,000 words combined after transformation of the three initial text features. So we used PCA to reduce the dimensionality to various different features sizes ranging from 100 dimensions for each original feature to 3000 dimensions for each feature.

5 MODELS USED

We started with basic models first and later trained the data with complex models. Also for each of these models we later used the gridsearchCV for hyperparameter tuning and selecting the best hyperparameters.

5.1 Logistic Regression

Training the data with basic logistic regressor.

```
pred,acc,roc = cross_validator1(lr,word2vec_train,train_data)
print("Accuracy :", acc)
print("Roc Auc Scores :", roc)
```

```
Accuracy : [0.7846283783783784, 0.8098055790363483, 0.7962806424344886, 0.7996618765849535, 0.8055790363482671]
Roc Auc Scores : [0.8560813587179839, 0.8759318698042715, 0.8697651483876886, 0.8666025993044115, 0.8687674877299207]
```

Figure 4: Logistic Regression with default hyper parameters

Trying to train with different hyper parameters with same model.

```
lr = LogisticRegression(C = 2,max_iter = 10000,solver = 'liblinear')
pred,acc,roc  = cross_validator1(lr,word2vec_train,train_data)
print("Accuracy :", acc)
print("Roc Auc Scores :", roc)
```

```
Accuracy : [0.777027027027027, 0.7954353338968724, 0.7878275570583263, 0.798816568047
3372, 0.7996618765849535]
Roc Auc Scores : [0.8506511665068712, 0.8724046388950871, 0.86305674051649, 0.86170602
2332052, 0.8640744690610522]
```

```
lr = LogisticRegression(penalty='l2',C = 2,max_iter = 10000,solver = 'newton-cg')
pred,acc,roc  = cross_validator1(lr,word2vec_train,train_data)
print("Accuracy :", acc)
print("Roc Auc Scores :", roc)
```

```
Accuracy : [0.7753378378378378, 0.797971259509721, 0.7886728655959425, 0.797125950972
1048, 0.8005071851225697]
Roc Auc Scores : [0.8505684152855774, 0.8723445644027165, 0.8631484794275492, 0.861734
6238330588, 0.8640458006513463]
```

Figure 5: Logistic Regression with different penalties, C values and solvers

Later with the use of gridsearch and PCA over TF-IDF vectorizor, the logistic regression model gave the best results for the kaggle public test dataset. (not for private dataset though!)

5.2 Support Vector Machine

Training the data with linear Support Vector machine

```
svc = svm.SVC(kernel = 'linear', C = 1) # Linear Kernel

pred,acc,roc  = cross_validator2(svc,word2vec_train,train_data)
print("Accuracy :", acc)
print("Roc Auc Scores :", roc)
```

```
Accuracy : [0.7778716216216216, 0.8021978021978022, 0.7971259509721048, 0.79543533389
68724, 0.7996618765849535]
Roc Auc Scores : [0.8468075149522896, 0.8705137227304715, 0.8607948029906886, 0.861085
369760205, 0.8638995917618457]
```

Figure 6: Logistic Regression with different penalties, C values and solvers

Training the data with rbf Support Vector machine

```
svc1 = svm.SVC(kernel='rbf',gamma='scale',C = 1) # RBF Kernel

pred,acc,roc  = cross_validator2(svc1,word2vec_train,train_data)
print("Accuracy :", acc)
print("Roc Auc Scores :", roc)
```

```
Accuracy : [0.7896959459459459, 0.7971259509721048, 0.7819103972950127, 0.7996618765849535, 0.7852916314454776]
Roc Auc Scores : [0.8487821303017851, 0.8672382325512207, 0.861867001513692, 0.8624181997071207, 0.8595190587587724]
```

```
svc3 = svm.SVC(kernel='rbf',gamma='scale',C = 0.1) # RBF Kernel

pred,acc,roc  = cross_validator2(svc3,word2vec_train,train_data)
print("Accuracy :", acc)
print("Roc Auc Scores :", roc)
```

```
Accuracy : [0.745777027027027, 0.7700760777683855, 0.768385460693153, 0.7743026204564666, 0.7590870667793744]
Roc Auc Scores : [0.8307024151942657, 0.8393608074011774, 0.8454686711618733, 0.8437242586490938, 0.8442330627035457]
```

Figure 7: Logistic Regression with different penalties, C values and solvers

Training the data with polynomial Support Vector machine

```
svc2 = svm.SVC(kernel='poly',gamma='scale',C = 0.5) # Polynomial Kernel

pred,acc,roc  = cross_validator2(svc2,word2vec_train,train_data)
print("Accuracy :", acc)
print("Roc Auc Scores :", roc)
```

```
Accuracy : [0.7609797297297297, 0.7616229923922232, 0.7480980557903635, 0.7802197802197802, 0.7633136094674556]
Roc Auc Scores : [0.8169600054787015, 0.8320917938243422, 0.8238068207880374, 0.8334277182866556, 0.8367162056786386]
```

Figure 8: Logistic Regression with different penalties, C values and solvers

5.3 K Nearest Neighbour Classifier

Next we trained the data using the KNN classifier, but it as we can see here that the roc scores were drastically reduced compared to the previously trained models

```
knc = KNeighborsClassifier()

pred,acc,roc  = cross_validator1(knc,word2vec_train,train_data)
print("Accuracy :", acc)
print("Roc Auc Scores :", roc)
```

```
Accuracy : [0.700168918918919, 0.7117497886728656, 0.702451394759087, 0.7083685545224
007, 0.7032967032967034]
Roc Auc Scores : [0.7598160069396886, 0.7718857097086101, 0.7655182101738452, 0.769918
0853011166, 0.7636834319526629]
```

Figure 9: Logistic Regression with different penalties, C values and solvers

5.4 Bagging Techniques

5.4.1 *Decision Trees*

We tried to train the data with decision trees. Again, as the KNN classifier this model was also not giving good results

```
dtc = tree.DecisionTreeClassifier()

pred,acc,roc  = cross_validator1(dtc,word2vec_train,train_data)
print("Accuracy :", acc)
print("Roc Auc Scores :", roc)
```

```
Accuracy : [0.6900337837837838, 0.7109044801352493, 0.7075232459847844, 0.6973795435333897, 0.702451394759087]
Roc Auc Scores : [0.6900824088024471, 0.7101005246505666, 0.7069171138938581, 0.6968612712795167, 0.7021094215861657]
```

Figure 10: Logistic Regression with different penalties, C values and solvers

5.4.2 *Random Forest Classifier*

Later we shifted to use the ensemble models.We started training RandomForestClassifier which is a decision tree ensemble which used bagging technique.

```
rfc = RandomForestClassifier(max_iter=10000)
pred,acc,roc  = cross_validator1(rfc,word2vec_train,train_data)
print("Accuracy :", acc)
print("Roc Auc Scores :", roc)
```

```
Accuracy : [0.7846283783783784, 0.8098055790363483, 0.7962806424344886, 0.7996618765849535, 0.8055790363482671]
Roc Auc Scores : [0.8560813587179839, 0.8759318698042715, 0.8697651483876886, 0.8666025993044115, 0.8687674877299207]
```

Figure 11: Logistic Regression with different penalties, C values and solvers

Though RandomForestClassifier was giving good results but these scores were not enough to surpass the logistic regressor/svm scores.

5.5 Boosting Techniques

Then we shifted to boosting techniques which are ada boost and gradient boost which the professor has discussed about in the class.

5.5.1 *Ada Boost*

Training the data with Ada Boost Classifier

```
abc = AdaBoostClassifier()

pred,acc,roc  = cross_validator1(abc,word2vec_train,train_data)
print("Accuracy :", acc)
print("Roc Auc Scores :", roc)
```

```
Accuracy : [0.7584459459459459, 0.7717666948436179, 0.7844463229078613, 0.7624683009298394, 0.7869822485207101]
Roc Auc Scores : [0.8218095123955623, 0.8543708484234737, 0.8520753061786157, 0.8400775672707304, 0.8422191069217009]
```

Figure 12: Logistic Regression with different penalties, C values and solvers

5.5.2 *Gradient Boost*

Though ada boost was giving satisfactory results, when trained with gradient boosting technique, we are getting slightly improved scores.

```
gbc = GradientBoostingClassifier()
pred,acc,roc  = cross_validator1(gbc,word2vec_train,train_data)
print("Accuracy :", acc)
print("Roc Auc Scores :", roc)
```

```
Accuracy : [0.7846283783783784, 0.8055790363482671, 0.8047337278106509, 0.8021978021978022, 0.8030431107354185]
Roc Auc Scores : [0.8506354723097292, 0.8720213064199608, 0.8850368102380624, 0.8675864909390445, 0.8737901931104077]
```

Figure 13: Logistic Regression with different penalties, C values and solvers

Later we used gridsearchCV to find the best hyperparameters but it did not give ant better results. We also used the TF-IDF with PCA.

5.5.3 *XG Boost*

Therfore we also decided to try out the XG boost (also called xtreme gradient boost) ensemble technique to train the model.

```
xg_cls = xgb.XGBClassifier()
pred,acc,roc  = cross_validator1(xg_cls,word2vec_train,train_data)
print("Accuracy :", acc)
print("Roc Auc Scores :", roc)
```

```
Accuracy : [0.7837837837837838, 0.8081149619611158, 0.8081149619611158, 0.8038884192730347, 0.8064243448858833]
Roc Auc Scores : [0.8563567205405652, 0.8715807601425768, 0.8814976950598598, 0.8601601112026358, 0.8731766891426999]
```

```
xg_cls = xgb.XGBClassifier(min_child_weight = 0.6, max_depth = 10)
pred,acc,roc  = cross_validator1(xg_cls,word2vec_train,train_data)
print("Accuracy :", acc)
print("Roc Auc Scores :", roc)
```

```
Accuracy : [0.785472972972973, 0.812341504649197, 0.812341504649197, 0.8021978021978022, 0.8030431107354185]
Roc Auc Scores : [0.8576564854129571, 0.8757716711579502, 0.8847386587771203, 0.8661521256635549, 0.8802921884317233]
```

Figure 14: Logistic Regression with different penalties, C values and solvers

As expected, the xg boost was also giving a slightly better scores to our data. Which in turn is so far our best model of all the models we have tried with the **word2vec vectorizor** (not the best overall model though)

5.6 Neural Networks

Finally we tried training with the neural networks with only a few variation of layers. Due to the time constraint we were not able to completely explore the neural networks with our project. So the basic neural network models did not give any satisfactory results.

6 TABULATION OF THE SCORES

Model Used	Kaggle Private Score	Kaggle Public Score	Parameters
Logistic Regression	0.86605	0.88352	C: 0.5, penalty: l2, solver: liblinear
Support Vector Machines	0.89194	0.86969	C : 1, kernal : linear,
K Nearest Neighbours	0.79997	0.80519	Default parameters
Decision Trees	0.87461	0.86179	Default parameters
Random Forest Classifier	0.88441	0.85489	Default parameters
AdaBoost	0.84276	0.86632	Default parameters
Gradient Boost	0.86160	0.87234	criterion: friedman mse, learning rate: 0.1, max depth: 8, max features: sqrt, min samples leaf: 0.1, min samples split: 0.3, n estimators: 10, subsample: 1.0
XG Boost	0.86557	0.87704	min_child_weight : 0.6, max_depth : 10
Neural Networks	0.70142	0.72301	With 3 hidden layers

7 EXPERIMENTS CONDUCTED

- Tried to find out any patterns if any by visualizing the text vectors with T-SNE (t-Distributed Stochastic Neighbor Embedding) as professor has suggested in one of the classes. But we were not able to gain any sense from the visualization.
- Tried GridSearchCV on different models, but few models were taking huge amount of times to execute so we were able to try it out only on few models, but the results were good.
- Tried SGD dimensionality reduction in the place of PCA but we were not able to find any improvements in the score.

8 CHALLENGES FACED

8.1 Memory Limit Reached

After the TF-IDF vectorization, intially we were left with around 70,000 features. Thus it became difficult for us to apply PCA on it as we first tried to convert it to a dense matrix for which we did not have sufficient memory.

But later we found out that we can still reduce the number of vectors from the TF-IDF vectorizor by setting the max_df = 2 (which means that neglect all the words which appear in more than half of the documents document)and min_df (which mean that neglect all the words which appear in less than 2 documents) parameters. Then the features space dimension reduced to around 30,000

8.2 Word2Vec Vectorizor

Finding the right references for Word2Vec vectorization process has been quite a challenge for us in the initial stages. However, we later managed to find a reasonably well documented article on one of the references mentioned below which kept us going.

9 CONCLUSIONS

- We got two best models. One with the TF-IDF vectorizor along with dimensionalality reduction using PCA on Logistic Regression (using GridSearchCV for hyper parameter tuning). This model gave the best kaggle **public** score.
- The second model is Support vector machine with Word2Vec vectors and linear kernal. This model gave the best kaggle **private** score.
- We intially thought that the more complex models such as Boosting techniques or the ensembles would give us higher results compared to the simpler models such as logistic regression and svms. But with this project we are now sure that every model is as powerful as any other model and we have to carefully select which one to use.

10 INDIVIDUAL CONTRIBUTIONS

The contributions towards the project have been majorly collaborative.There were inputs and ideas from both of us on a time to time basis. There was no division of tasks but ideas for different approaches were given by both of us. For example, the idea to implement gridsearch and xgboost was suggested by Dhamodhar and the idea to make use of word2vec was given by Jayanth.

11 REFERENCES

- Basic flow followed in NLP tasks: (Referred by TAs) :External Link
- Basic flow followed in NLP tasks: (Referred by TAs) :External Link 2
- Yeo-johnson transformation : External Link
- Parameters and syntactical help for various models: External Link
- Word2Vec-Intoduction: External Link
- Word2Vec-Visualizations: External Link
- T-SNE Introduction: External Link
- XGBoost documentation and installation: External Link
- Materials from TAs: External Link