Predicting the reception of comments on the New York Times Articles

Team Members:

- Megha Mishra; Email: mmishra@seas.upenn.edu
- Saumya Shah; Email: saumya97@seas.upenn.edu
- Sottihat Winyarat; Email: winyarat@seas.upenn.edu

Abstract

This project aims to analyse and predict the reception of comments on the New York Times articles by the New York Times community. It also aims to analyse the features that play a key role in determining the reception, which is indicated by the number of recommendations a comment receives. Some of the main applications of this work would be to automate the process of editors' selection for the comments as well as to highlight the comments which shall potentially be the top comments on the given article. This prediction can be useful for identifying the top comments with the potential to âtake offâ and can be possibly ranked higher in the comments section. We also examine the feasibility of online prediction of the number recommendations on the comments. The exploratory data analysis was performed and new features were engineered to help the cause. Topic modelling using Latent Dirichlet allocation was also performed for extracting 20 topics from the articles and the comments as a measure of relevance and also to understand which topics are primarily popular among the users. Various models were used and comparison of performance is reported.

1 Motivation

The New York Times is an influential newspaper with a wide reader base around the world. It has a very active comment section. Well received comments on a particular article are given high number recommendations by other readers. Our goal is to predict a number of recommendations a comment would receive and thereby analyse features that affect the level of such reception. We hope to find underlying patterns that could explain why, given an article, certain comments are well received while others do not. And therefore, we consequently wish to provide insight into the general trends and characteristics of the New York Times community. It may also help ranking the top comments beforehand and may also be useful to automate the process of Editor's selection. The primary motivation to take up this problem was the subjectivity of the issue and difficulty of effectively capturing the factors which govern the comment space dynamics. A lot of work has been done on these lines but the results haven't been as impressive as we desire them to be. Machine Learning has a lot of potential but we haven't been able to exploit it as well. The problem is very subjective and the results of predicting recommendations may be very biased if done by a particular person. We thus wish to see if we can utilize the potential of Machine Learning to solve the problem and also understand the underlying factors which influence the reception of comments.

2 Related Work

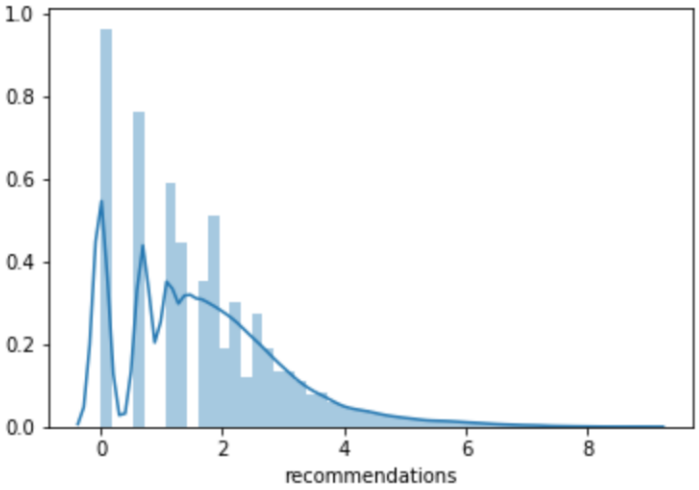
Different aspects of the comment space dynamics have been explored in the past. There are a few Kaggle projects done both with the NYT article data and comment data, although none of them have worked on prediction of the number of upvotes on a given comment. One related project is Aashita Kesawani 's [4] *Predicting the selection of comments on NYT articles as editor's picks*, a classification problem. From the comment bodies and the IDs of articles on which comments were made, Kesawani used logistic regression

model, coupled with Latent Sentiment Analysis to generate an extra 'sentiment' feature, to predict whether or not a comment will be picked by the editor. According to her conclusion, most of the comments picked by the editors had neutral sentiment. The data used in the editor's selection classification problem was highly unbalanced with less than 6% probability of a comment being picked by an editor. The model is evaluated using ROC curve and Precision vs Recall. Since the original data contains high skew of negative samples, undersampling from the negative samples was also performed to achieve class balance. We anticipate a similar issue with our training set containing outlier comments with unusually high number of recommendations. One of the possible cons of her approach is undersampling the data to cure the imbalance. This may not be a good idea as we would lose information in the process. Also, the project was mainly done to set a good baseline and thus, used basic Naive Bayes logistic regression only.
Apart from this, Tsagkias et al. (2009)[7] classified the articles receiving high or low number of comments. They solved it in two stages. Firstly, they solved the binary classification problem to predict whether a given article will receive any comments or not. Later they classified the articles as receiving high or low number of comments. To achieve the second target, they considered the number of comments received in the top ten hours and later proposed linear model to predict the total number of comments that the article will receive. They also performed an exploration of temporal cycles governing the news comment space. Though the performance of the first stage of classification was plausible, the performance for their second stage classifier degraded as the comment dynamics were much more complex. Their failure analysis revealed that the features used were not the only factors that governed the comment space and thus, more features related to context, different encoding for the current features, and entity relations were possibly required. We shall thus consider having more context related features, for example, the ones generated by performing LDA.
Ambroselli et al (2018)[6] worked on classifying the top 10% articles receiving maximum number of comments. They have used traditional logistic regression to find the baseline accuracy and have further worked on improving it using Random forest and other tree regressors. They analysed that using new metadata features such as other competing published articles at the same time, temperature at the time of publishing, promoting the article on FaceBook page, source of the article i.e news agency showing that news etc enhanced the model performance. They further performed topic modelling using Latent Dirichlet Allocation (Blei et al, 2003)[5], document embedding, to capture the semantics of the article. They analysed the comments received on the articles and observed that early comments receives lots of attention and articles getting controversial comments after the first few minutes of publishing are more likely to have a large comment section. Thus classifying the tone of the initial comments further helped to boost up the accuracy.
Also, Jordan S. et. al.[8] had worked on a similar problem. They had tried to predict the popularity of the REDDIT posts. They present a comparative study between various models including linear regression, multi-class Naive Bayes and Multi-class Support Vector Machines by dividing the regression problem into multiple bucket for classification. They claim that they could have achieved better results if they had more data and processing power. Also using extra features outside of Reddit API such as categorizing images, examining the link contents etc could have further helped to improve performance.

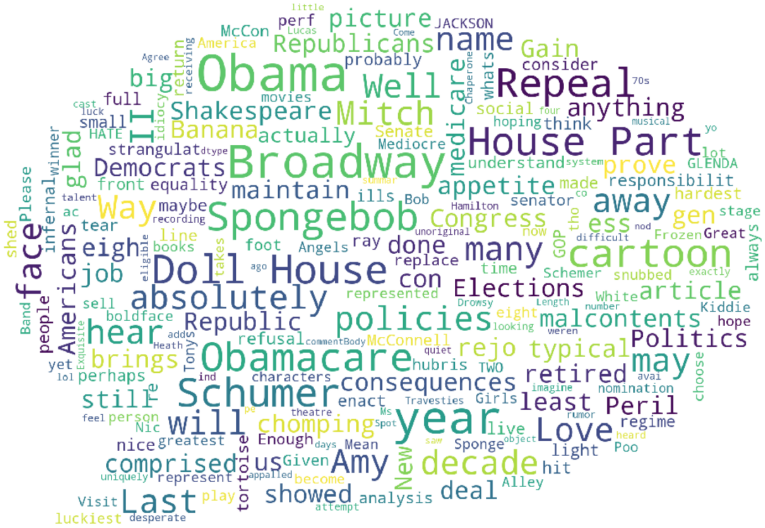
3 Data Set

The data set for the given problem was found on Kaggle.[3] The data set contained information about the comments on the New York Times articles over the period of Jan-May 2017 and Jan-April 2018 in a csv file for each of the months. Also, the data set about the information of the articles over which the comments were made was available. The comments data set contained over 2 million comments with 34 features and the article data set contained over 9000 articles with 16 features. Owing to the realization difficulties, we had sub-sampled 194,632 comments from the total data set after excluding some data from the last month (for the purpose of ultimate testing over future comments). From the sub-sampled set, 75% of the data was reserved for training, 15% for validation and 10% for testing. The 2 testing sets were mainly to analyze how the model would perform on the data from the distribution it is trained on (distribution in the sense of the timeline of the comments) and on the data which is new (more recent comments, present in the last month of the data set). Below are some visualizations of the data we used.

1. The distribution of the target y (log of the recommendations):



2. The common words in the comments



3. Preprocessing includes removal of many irrelevant features: approveDate, commentID, commentSequence,commentTitle, editorsSelection, parentID, parentUserDisplayName, permID, picURL, recommendedFlag, reportAbuseFlag, sharing, status, timespeople, trusted, updateDate, userDisplayName, userID, userLocation, userTitle, userURL, inReplyTo, articleID, sectionName, newDesk, printPage, and typeOfMaterial.

   We also converted the following categorical features into one-hot encodings: commentType, userReply and reporterReply.

   After prepossessing and feature engineering(See below in Problem Formulation section), there are p= 63 features. Training set contains 144,227 samples, Test set contains 20,162 samples, and Validation set contains 30,243 samples.

   NOTE: Detailed exploratory data analysis on the features and other engineered features are discussed in the Problem Formulation Section below.

4 Problem Formulation

Target Variable Y:
Our goal is, given a comment and the corresponding information about the article and the comment, predict y i.e. log of the number of recommendations the comment will receive. Since we do not aim to predict the exact number of upvotes but rather the scale of how well the comments shall be received and also to deal with the skew in recommendations, we take y as $log(recommendations)$
The descriptive statistics of y is as shown below:

mean	2.128227
std	1.280272
min	0.693147
25%	1.098612
50%	1.791759
75%	2.708050
max	8.873888

Loss Function:
As seen from the dataset, the number of recommendations the comments receive is highly skewed towards the right as most of the comments do not receive any recommendations, using RMSE may not be a good choice as it shall largely weight the errors made for the higher number of recommendations and may potentially bias the results towards higher number of upvotes. Thus, we will work with mean absolute error as the evaluation metric.

Features X:
Our training data set X contains the following features:

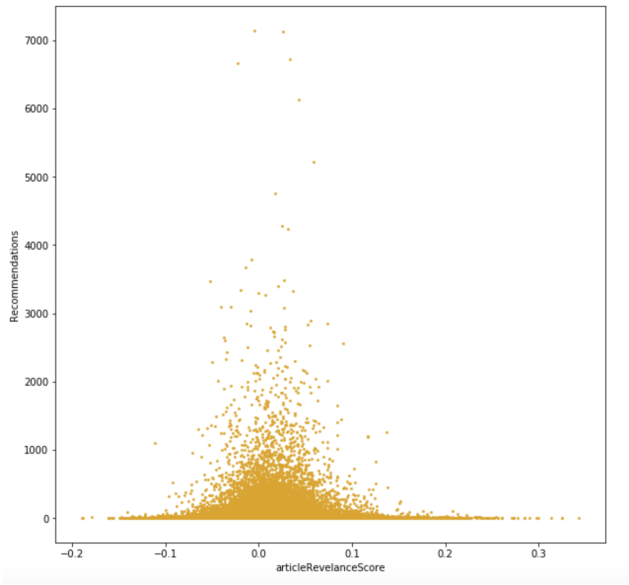
1. Article Relevance Score (ARS):
   Let C_i be the set of all non stop word vectors in comment i. Let A_i be the set of all key word vectors $\vec{a}'s$ of article A on which comment i is on. Let k be the number of key word vectors in A_i. Compute for each $\vec{c_m} \in C_i$,

$$W_m = \frac{1}{k} \sum_{j \in range(k)} cosineSimilarity(\vec{c}, a_j)$$

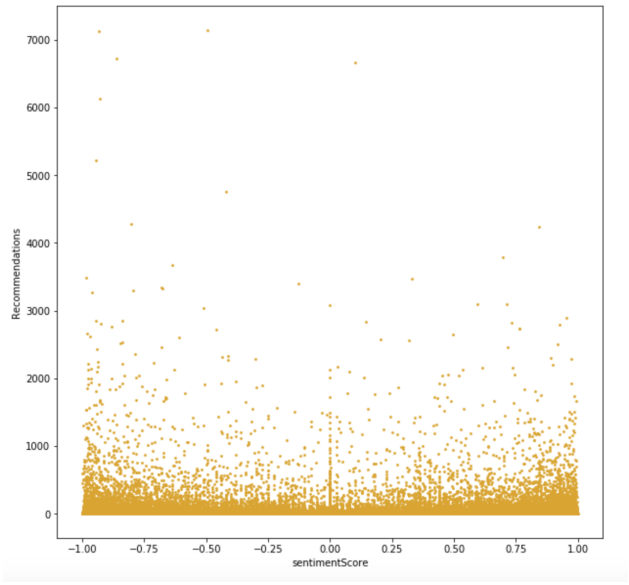
   For each comment sample $x_i \in X$, its Article Relevance Score,

$$ARS_i = \frac{1}{\|C_i\|} \sum_{m \in range(\|C_i\|)} W_m$$

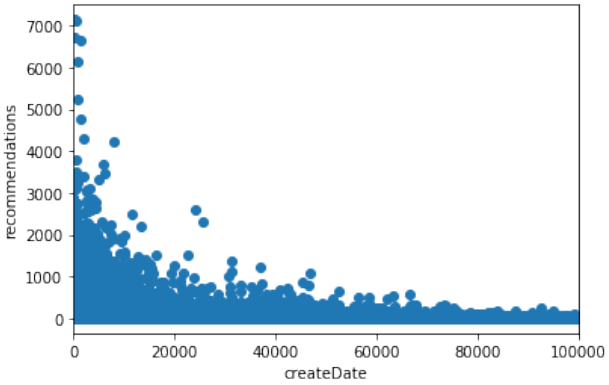
   $W_m \in [-1, 1]$



2. Sentiment score: We use a Latent Sentiment Analysis model from NLTK[1] to evaluate each comment's sentiment score, which ranges from -1 to 1.

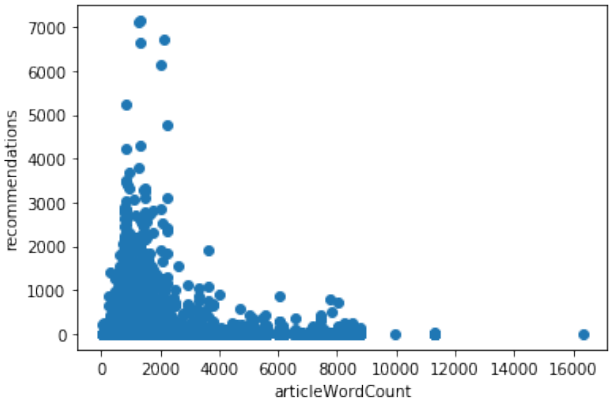


3. Time (create Date):
   As per the work by Ambroselli et al. (2018)[6], the early comments presumably receive more attention than the others and with their tone and sentiment, influence the comment volume to a larger extent. Thus, intuitively, the earlier the comments are posted, the more audience it shall receive and higher shall be the recommendations. So the create date feature was used to represent how recent the comment is. This was provided in the format of seconds. It was then modified such that the value is equal to the difference between the publishing date of the article and the comment in seconds. On performing the analysis, the following was the distribution of recommendations against the create date feature.



Therefore, this is just like we expect, the highly recommended comments are generally the ones which were posted earlier.

4. Article word count
   The article dataset contained the corresponding url to each of the article ID. Thus, for each of the comments in the comment dataset, the corresponding article urls for the respective article IDs were used to scrape the article content. Beautiful Soup API was used for scraping and to accelerate the process, multi-core processing was used.
   The number of words in the article was also taken as a feature. This may not seem directly relevant, but the following was the :

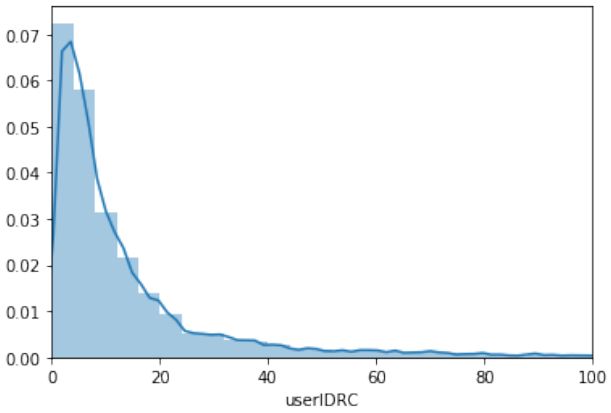


   It shows that comments on the longer articles were lesser recommended probably because the articles weren't read by many and similarly for the very short articles, the recommendations were fewer. But it peaked for the articles with word count in the range of 500-2000 and thus, this could be a good feature to have.

5. Reply count
   This feature was considered as the popular comments would probably have more replies. There is a possibility that a comment is a controversial one and thus, may have a large number of replies and that can possibly attract readers of different polarities to start an argument and thus, may lead to higher upvotes. Also, it is possible that the comment was a question and thus, may not have large number of recommendations but may have high reply count.

6. Average recommendations received on a comment by userID ('userIDRC' in dataFrame)
   The comments data-set included the userID of each of the users who made the comments. This was used to generate this feature. This feature was to capture the popularity of the user. It was calculated by adding all the recommendations a user had on his/her comments divided by the total number of comments by that user. If a particular user in general had higher average number of recommendations, he would tend to have more followers and the recommendations can

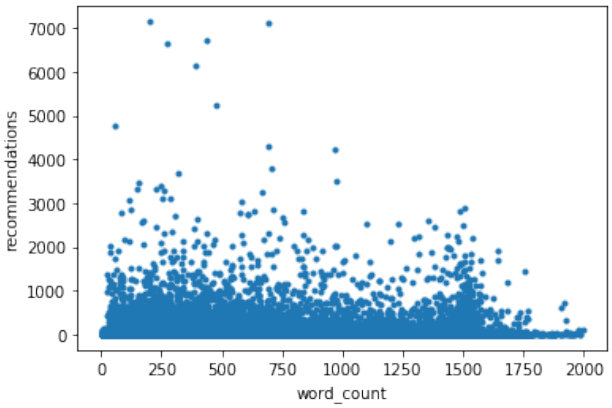
be assumed to be higher for the popular users' comments. The following was the distribution of this feature:



7. Editor's Selection
   This feature is considered as the editor's selection would probably be also popular among the readers and thus, may have higher recommendation count. Aashita Kesawani âs [4] work predicted if a comment shall be an editor's selection or not. Her exploratory analysis of the feature reveals that the feature has a very high class imbalance with the ratio of comment being an editor's selection was 1:20, but the ones which were picked by the editor had a higher number of recommendations on an average. The pie chart is as shown below:

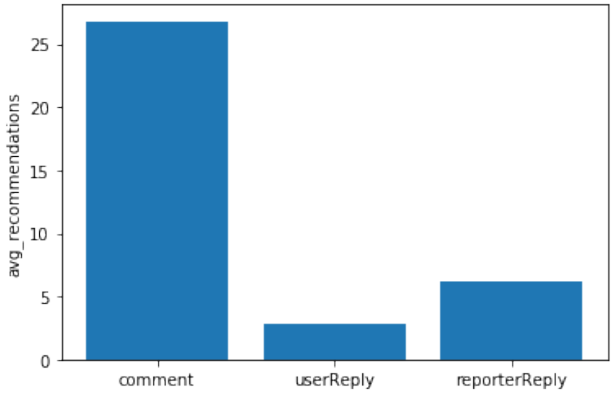
8. Author's popularity ('ratio_comment_article' in dataFrame)
   The number of readers can strongly affect the number of recommendations over the comments on a given article. Also, we may assume that articles by popular author may have larger audience. This feature was used to capture the popularity of the author by the average number of comments the articles by a given author received. Thus, more popular author may have higher number of comments and thus, might increase the number of recommendations on a given comment owing to a larger audience.

9. Comment Word count
   This is the number of important words in the comment. This was obtained by counting the number of words in a comment after removing the stop words. The plot of recommendations vs the comment word count is shown below:

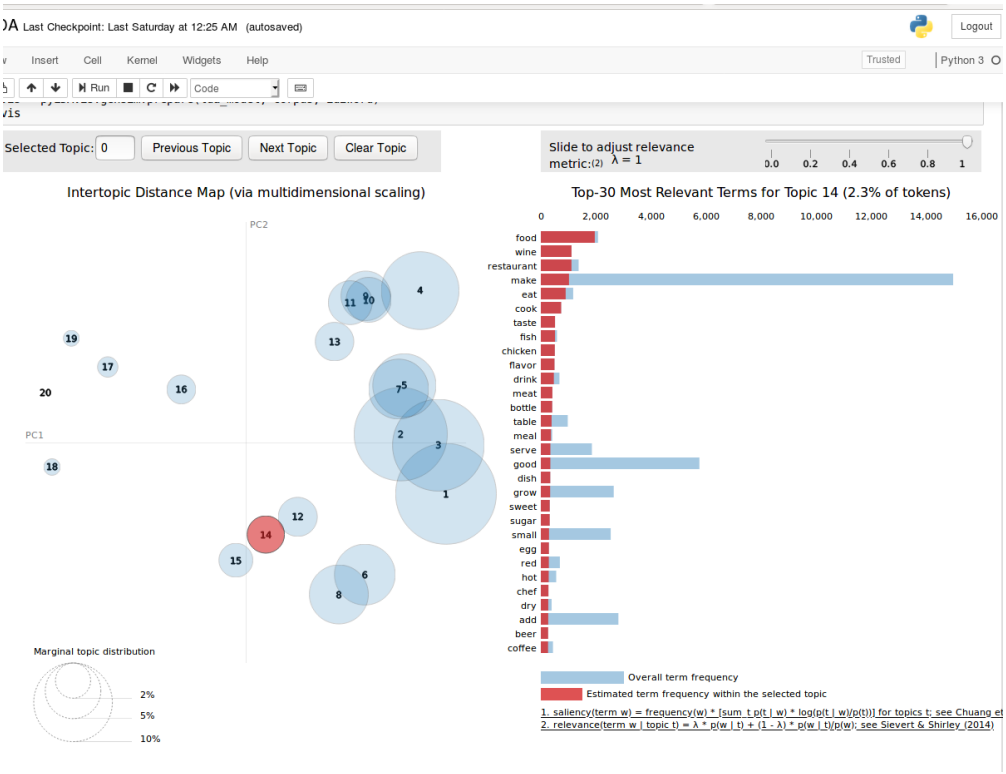


Thus, it shows that the comments which are very short or very long may not be recommended as much as those in the middle range. Although, the distribution seems to have a large standard deviation.

10. Type of the comment (comment, user Reply, reporter Reply)
This represents if a comment is a comment or a reply by a user or a reply by the reporter. The type of the comment may help us determine the recommendations as we may not expect a user Reply to be as highly recommended as the main comment or even a reply by the reporter. This feature was one-hot encoded as it is a categorical variable.
The bar graph below shows the average number of recommendations received for each type:



11. Topic wise similarity of the comments to the article
There were a total of 9000+ articles. The content of each of them was scrapped using multi-core processing and Beautiful soup API as explained above. This was used to generate a corpus. The articles were merged, the stop words were removed, bigrams and trigrams were found and the data was lammentized to create a corpus. This was then passed to gensim's LDA model. Topic modelling was done over this entire set of articles to get 20 different topics (decided by checking the coherence for different number of topics) with their respective distributions of the words. Using this model, the topic distribution over the comment was found. The product of the topic score for each comment with the respective article for each of the topic was found and thus, 20 dimensional feature was generated for all the comments. The product was taken so as to capture the contextual similarity between the comments and the articles. Thus, more relevant comments may have similar topic scores as that of the articles and thus, higher product values. The higher the product, the more is the similarity. The visualization of the trained LDA model using pyLDAvis can be seen below:



The bubbles represent the distribution of topics in the latent space and the bar graph on the right shows the distribution of the word in the 14th topic (highlighted in red). The interactive visualization can be found in the final notebook.

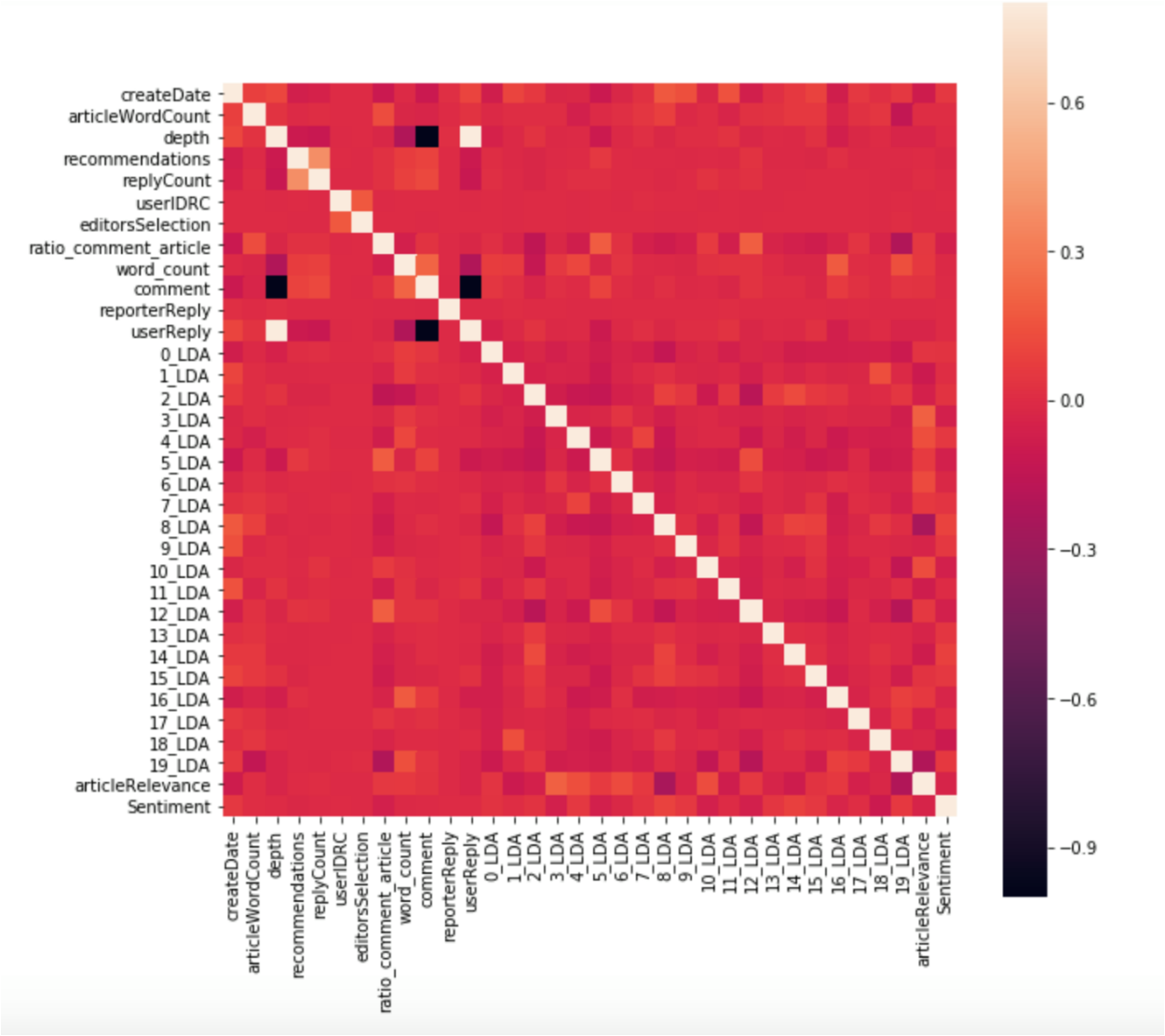
12. Term frequency-inverse document frequency
    The TF-IDF feature here captures how important a word is to the comment among all the comments in the corpus of all the comments. The importance or the TF-IDF weight increases proportionally to the number of times the word appears in the comment relative to how often it is generally used. It captures specificity of words in the comments which can be a good feature to look at, to determine the recommendations a comment would receive. The dimensions of the TF-IDF features were reduced to 30 using PCA.

Thus, the final feature matrix $\mathbf{X}$ contains 63 columns which shall be used to predict the target $\mathbf{y}$ which is $log(recommendations + 1)$ using the methods describted in the next section.

5 Methods

1. **Baseline:**
   Jordan Segall et. al [8] had used median score of all the posts within the data set to predict the post's score as the baseline algorithm in their work on predicting the popularity of the REDDIT posts. It turns out that for our case as well, fitting a simple linear regression model worked only slightly better than using the baseline algorithm as used above. Thus, beating the baseline algorithm which predicts the comment's recommendation as the median of the recommendations in the data set shall be a challenging and appropriate baseline to have.

2. **Polynomial Regression**
   On performing exploratory data analysis on the features, most of them were not correlated directly with the target variable. The figure below shows the correlation matrix of the data set:



Thus, performing Linear regression would not have been suitable for this problem. To capture the possible non-linearity, we planned to use a second order polynomial regression (regression using transformed basis of X).
We used the scikit-learn's polynomial features to create the transformed features and then use the linear regression module for polynomial regression. The linear regression on the polynomial features

was also performed with lasso penalty (for feature selection) and with elastic net (best of both worlds of ridge and lasso).

3. **KNN Regression on selected features**
   Since the correlation between our target Y, recommendations, and our features are likely highly non-linear, KNN Regression emerged as a potential candidate. However, with engineered and raw features totaling 63, KNN regression would suffer from the curse of dimensionality. We therefore performed feature selection to reduce the number of features from 63 to 9. We then used Skicit-learn's KNeighborsRegressor and experimented with the parameter k while setting the weights to the neighbors' distance. We have found that setting the weights to be the neighbor's distance worked slightly better than simply using uniform weights.

4. **Neural Network**
   We used a simple neural network comprising 3 layers. The first 2 layers have 64-dimensional outputs, each followed by a ReLU activation. The Last layer is a linear mapping of 64-dimensional hidden vectors to a scalar. The loss function the network optimizes is the mean absolute error loss. Adam optimizer with a batch size of 256 was chosen over stochastic gradient descent because with SGD, our loss got was stuck in a local minimum. We used Keras's sequential network model to implement the network.

5. **Decision Tree Regression on selected features**
   We used scikit learn's Decision Tree Regressor, hoping that the splitting procedure of a decision tree would give us another perspective on the importance of each feature.

6. **Ensemble Method(2 stage prediction):**
   The data set contains a lot of comments with lower recommendations recommendations.A large number of comments have zero recommendations, with mean recommendation being 17 and median 3. The descriptive statistics of the recommendations are shown below:

```
count    243832.000000
mean         17.625172
std          93.219419
min           0.000000
25%           1.000000
50%           3.000000
75%           9.000000
max        7938.000000
Name: recommendations,
```

   The value counts for the recommendations in the data set is as shown below:

```
df1.recommendations.value_counts()
```

```
0.0     26356
1.0     20534
2.0     15705
3.0     11955
4.0      9389
5.0      7483
6.0      6130
7.0      5066
8.0      4375
9.0      3645
10.0     3153
11.0     2687
12.0     2408
13.0     2141
14.0     1826
15.0     1686
16.0     1463
17.0     1275
18.0     1239
19.0     1042
20.0     1013
21.0      939
22.0      810
23.0      780
24.0      647
26.0      600
25.0      597
27.0      545
28.0      508
29.0      486
```

Thus, we used a classifier as a top level filter to deal with the skew. The classifier classified if the comment will receive more than 3 (median of the recommendations) recommendations or not. Following this, we shall used 2 different regression models to deal with each of the categories.

7. **XGBoost**
   XGBoost is one of the fastest implementations of gradient boosted trees. It was used as regressor for our problem. The model is trained by using simple decision trees and boosting by fitting the trees on the errors. Boosting algorithms are prone to over-fitting but XGBoost implements a few regularization tricks, this speed up is by far the most useful feature of the library, allowing many hyperparameter settings to be investigated quickly. To further reduce overfitting, we had used subsampling and early stopping. We have used Python's XGBoost API for implementation.

8. **AutoML (autosklearn regressor)**
   As encouraged in class, we tried using autoML for getting the optimized ensemble. We had used autosklearn's regressor for this purpose. The training time was very high so we had to subsample the training data to reduce the training time. We had tried implementing k-means clustering to get the most distinct set of 10000 datapoints but the time was too long. Thus, finally a random sample of 10000 points was used to train the autosklearn model and the results were surprisingly much better than a lot of other models tried, which can be seen in the results section.

9. Other models tried:
   LSTM models using keras, Linear regression with Ridge penalty, logistic regression. They aren't explained in detail because they didn't perform as well due to the reasons further explained in the Experiments and Results section.

   The implementation and results of these models can be found in the 'Modelling.ipynb' notebook

6 Experiments and Results

Performance Metric: The performance metric chosen was the mean absolute error on the prediction of log (recommendations+1). As seen from the dataset, the number of recommendations the comments receive is highly skewed towards the right as most of the comments do not receive any recommendations, using RMSE may not be a good choice as it shall largely weight the errors made for the higher number of recommendations and may potentially bias the results towards higher number of upvotes. Thus, we planned to work with mean absolute error as the performance metric.

Note: The term test error in the report represents the error on the data set used for testing and comparing among the models and the held out test data was used only for reporting the final errors by the tuned models at the end of this section in the table.

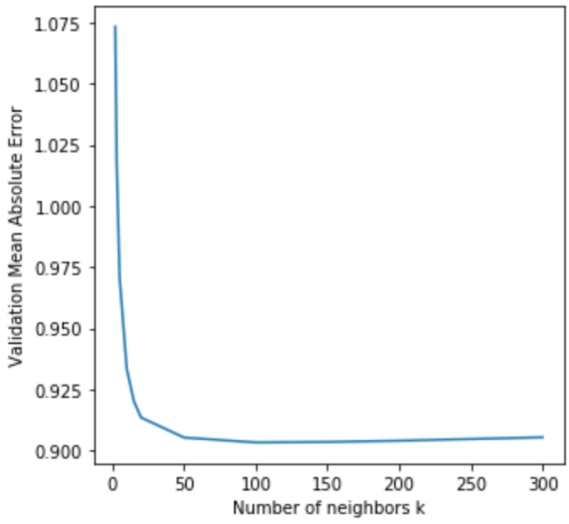
1. **Baseline**:
   The baseline algorithm of using the median value $log(recommendations + 1)$ as the prediction gave the mean absolute error of **1.034** on the training data, where as the error was **1.042** on the test data.

2. **Polynomial Regression**
   Polynomial Regression was performed by generating polynomial features using sklearn's PolynomialFeatures with degree 2. Degree was set to 2 because the number of features scale exponentially with higher degrees. Linear Regression models were then fit on the polynomial feature set. The results for regression with no regularization, lasso penalty and with elastic net are tabulated below:

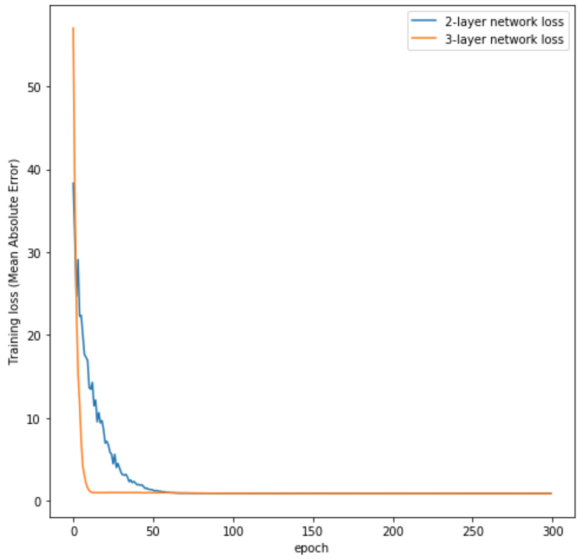
	No regularization	Lasso Penalty	Elastic Net
Train MAE	0.8634	0.9132	0.9098
Validation MAE	0.9131	0.9441	0.9424

   Thus, the result was the best in case of no regularization. The training error increases on adding penalty to the regression, as expected, as it adds to the bias. The bias allows us to prevent overfitting, but in this case, the model was already too simple to probably fit the highly non-linear nature of data and thus, regularization only degraded the performance of the model.

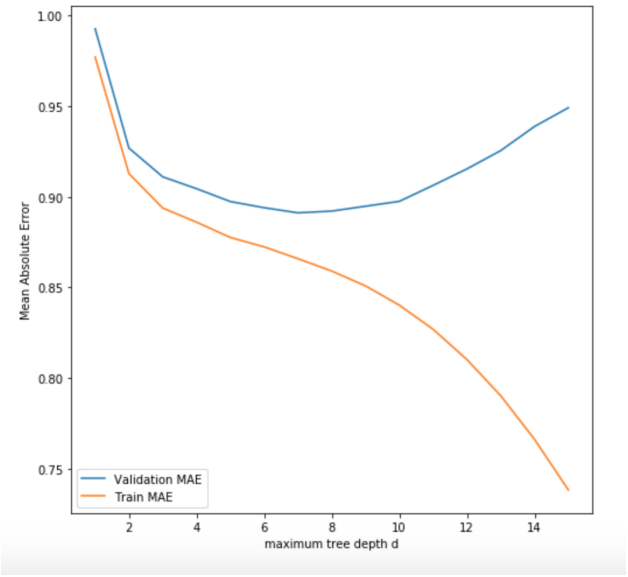
3. **KNN Regression**
   KNN: KNN regression on standardized 9 features selected from the original 63 features. The feature-selection process can be viewed in the jupyter notebook. The regression model with $k = 100$ gives an optimal validation MAE of 0.903, which is slightly lower than the base line and that using polynomial regression.



4. **Neural Network**
We trained 2 simple neural networks also on the selected 9 features. Below is a plot of the network training losses over 300 epochs. We experimented on 2 simple networks; one with 3 hidden layers and the other with 2 hidden layers. As apparent in the plot below, the 2 networks converges to 0.9. The 3-layer network, however, converges relatively faster than the 2-layer one. After training, we feed both networks with validation data and obtain the following losses: 2-layer network has a validation MAE of 0.957 while the 3-layer network has a validation MAE of 0.9157. The negligible difference between the 2 validation errors and the similar point of convergence during training suggest that simple neural networks such as these are limited in capturing the correlations between the target y and the features.

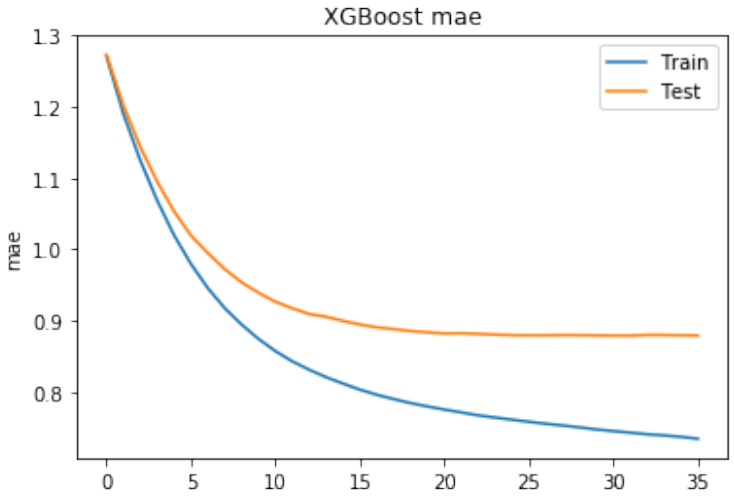


5. **Decision Tree Regression on selected features**
According to the plot below, the optimal tree depth = 7, for which the validation MAE is 0.891. Using $feature\_importances$ attribute of the decision tree regressor, the following are the 9 selected features ordered by increasing importance: 'Sentiment', 'userIDRC', 'comment', 'depth', 'wordCount','articleWordCount', 'createDate', 'userReply', and 'replyCount'.



6. **Ensemble**
   The Ensemble model with 2 stages was tried. The first stage was a binary classifier which predicted if a comment shall receive more than 3 recommendations (since 3 was the median). Random Forest Classifier module of sklearn was used for the purpose and 3-fold cross validation was performed for optimal hyper parameter settings of max-leaf nodes, max-features and max-depth. The number of estimators was fixed to **100**. The accuracy of the first stage classifier was **0.673**.
   For the second stage, sklearn's random forest regression models were used. 3-fold Cross validation was performed for this case as well for getting the optimal hyper parameter settings of max-leaf nodes, max-features and max-depth. The number of estimators was again fixed to **100**. The mean absolute error for regression on the points with classified recommendations greater than 3 was **0.981** and that on the points with classified recommendations to be lesser than 3 was **0.743**. Thus, the model actually performed better on predicting the target in lower range.
   The combined MAE was found to be **0.896**.

7. **XGBoost**
   XGBoost is a boosting algorithm and thus, it is very easy to overfit. Thus, strict regularization was used while training this model. 'eta' which is the step size shrinkage used in update to prevents overfitting. After each boosting step, we can directly get the weights of new features, and eta shrinks the feature weights to make the boosting process more conservative[2]. This was set to 0.4. Also, early stopping was set to 5. Subsampling ratio was set to 0.5 to subsample the training set used to fit during each iteration, which further helps to prevent overfitting. The maximum depth was set to 10 as boosting methods can easily overfit the data even when simple base classifiers are used. The training and testing error curves for this model with respect to epochs/iterations is shown below:



   Thus, the MAE on the train set was **0.7423**, and that on the test set **0.8787**. This was the best result on the test error we had got so far.

8. **Auto sklearn**
   Further more, we also went ahead to try AutoML implementation using sklearn's Autosklearn regressor. The algorithm took a very long time to run and thus, we had to prune our training data to get results in feasible time. Thus, we took 5000 randomly sampled data points to train the autosklearn regressor. The training time was too long but the wait was worth the results. Surprisingly, the train mae was **0.6635** which may seem like it was overfit. But the test error was only **0.8892** which was the second best result from all our models we had tried. Even with lesser number of training examples, the model was able to get splendid results. Thus, it indeed would be a very helpful tool for the future when the time constraints would not be a deciding factor.

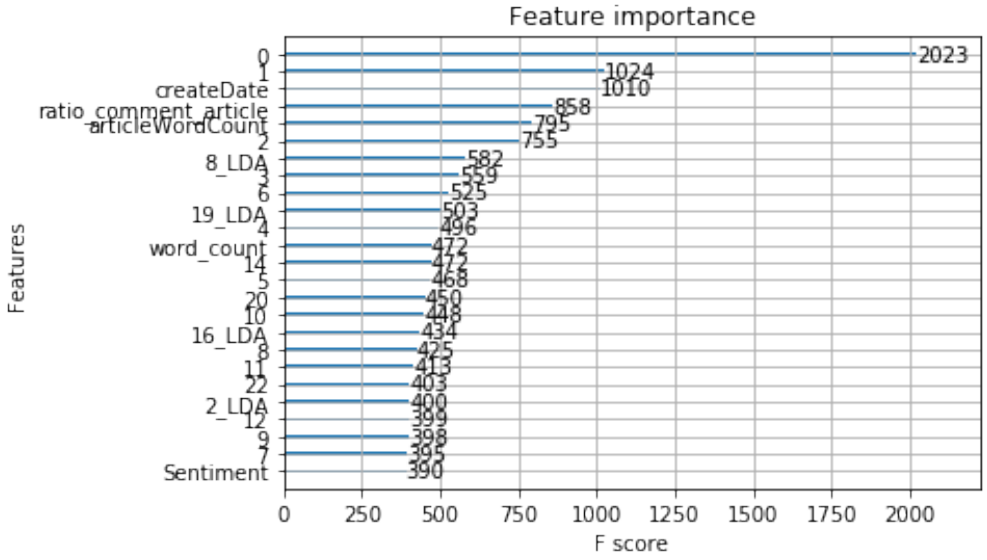
The final summary of all the train and test errors is tabulated below:

MAE	Baseline	Polynomial Reg	KNN	NNet	TreeReg	Ensemble	XGBoost	Autosklearn
Train set	1.034	0.8634	-	0.9	0.865	0.836	**0.7423**	0.6635
Test test	1.042	0.9131	0.903	0.9157	0.891	0.894	**0.8787**	0.8892
Heldout set	1.033	0.8986	0.8966	0.8969	0.884	0.8826	**0.8641**	0.8793

Finally, the best model i.e. XGBoost was run on the held out testing set. The mean absolute error was found to be **0.8641**.

7 Conclusion and Discussion

The feature importance resulting from the best model i.e. XGBoost is as shown below:



Conclusions based on feature importance:
Although the feature importance doesn't tell us about how the features explicitly affect the target prediction, but we can fairly conclude the trends based on the importance and the pattern observed for the feature relative to the target variable from the respective plots.

1. From the feature importance map, we realize that the top 25 features included a lot of TFIDF features (the features named with numeric values), the dimensions of which were reduced to 30 using PCA. Although it may be difficult to understand the importance in the transformed PCA space, we see that the components included are mostly the PCA components corresponding to the higher singular values of the SVD (components 0-9 which are the top 10 principal components appear as important features). Thus, we can conclude that certain set of words have their tf-idf captured in the top principal components govern the dynamics of our problem.

2. Next, createDate feature was important. Create date specified the time interval between when the comment was posted and when the article was published. Thus, analyzing the feature importance and weights, we conclude that the later the comment was made, the lesser were the number of recommendations it received as expected as also seen from the plot of recommendations vs createDate in the problem formulation section.

3. The ratio comment article feature captured the popularity of author as explained before in the problem formulation section. Thus, the popularity of the author of the article (i.e. the average number of comments an author receives on his articles) indeed plays a role in determining the recommendations a comment would receive. Our assumption of comment on an article by a popular author would have a chance of receiving higher number of recommendations owing to a larger audience can thus be proved.

4. The article word count also shows up as an important feature. Thus, it may not affect the recommendations on the comment directly as seen on the plot of recommendations vs the article word count, but it does affect the target in a non-linear fashion which most probably is a case that comments on too short or too long of an article may not be as liked due to lesser audience as assumed before.

5. comment word count: The word_count feature is the length of the comment without the stop words. This is important as expected since too long of a comment may not be read by a lot of them and too short of a comment as assumed.

6. 8_LDA: This feature on looking up at the LDA distribution for topic 8 represents the group of words largely related to Law and Administration. Thus, probably the comments related to this topic were popular among the readers.

7. 19_LDA: This feature was related to the topic largely encompassing the words related to Crimes and Safety and thus, the comments related to this topic were also popular.

8. 16_LDA: This feature was related to the topic with the major words related to Food and thus, since we know it is an important feature, it may have positively or negatively, but certainly affected the prediction on recommendations.

9. Sentiment: Sentiment of the comment also turned out to be an important feature to consider. We may conclude that the neutral comment were probably received well on average as seen from the sentiment vs recommendations plot.

10. Features that didn't matter: The type of comment (comment/user reply/ reporter reply) didn't play a major role. The editors' selection didn't show up. The obvious reason for their absence is that they had a high imbalance in their distributions. The editors' selection had less than 6% of the values as positive. And, average number of recommendations received on a comment for a given userID didn't receive good weightage probably because it the recommendations didn't depend as much on, from whom the comment was coming, as expected. The reply count didn't matter probably because of the sparsity of the feature as well as the possibility of a comment just being a question expecting an answer as a reply.

Conclusions on the performance of the models:

1. Polynomial Regression: The polynomial regression didn't perform as well most probably because it was only restricted to degree 2. Thus, 2^{nd} degree terms could not probably represent the actual non-linear relations between the feature space and the target variable well enough.

2. KNN Regression model, with its optimal validation MAE of 0.903 which is lower than the baseline of 1.042, is very limited in its capacity to predict the target y. We speculate that this limitation is due to the quality of the features of the training samples. Even after an initial feature selection of 9 features from the original total of 63 features, the 9 features correlations with the targets are still not strong enough.

3. Like the KNN Regression model, the resulting MAEs from our neural network models suggest weak correlations between the majority of our selected features and the target. Even if these correlations were highly non-linear, we had hoped that neural networks would outperform regression models, but they unfortunately do not, most probably due to our inability to find the optimal architecture.

4. Unlike KNN regression model and neural networks, however, decision tree regression model has given noticeable MAEs both on the training and validation sets when compared to the baseline. Such lower MAEs further highlights the weak correlations between the majority of the selected features and the target. In other words, the reason for the decision tree regressor obtaining a lower MAEs than both KNN regressor and neural network, may be that it will attempt to makes use of the most informative features in branching rather than considering or weighting all of them.

5. Ensemble (classification followed by regression): This method suffered primarily because the accuracy of the first stage of classification was 0.69 on the training data and 0.67 on the validation set, which weren't as good and thus, added to the error in the following stage.

6. XGBoost: Fitting a simple decision tree performed really well and thus, the idea of using ensemble of trees with boosting spiked up. As expected, the XGBoost tree boosting algorithm worked the best for us evidently. This was because it was able to capture the non-linearities of the feature space better than most and boosting further helped improve the performance of the ensemble of trees.

7. Auto-sklearn: The performance of this model was surprisingly very good in spite of the smaller training dataset. Thus due to the reliable results, this shall surely be on the list of trials for our future projects.

8. The other models which were tried that didn't help us as well were Logistic Regression and LSTMs. The main reason why LSTMs weren't performing well was because of failure to find the optimal architecture and hyper-parameters. Not a lot of time was spent on LSTM models as the search space for parameter tuning was huge and probably wouldn't have contributed to the learning objectives of the project. Also, another reason to not having spent time on LSTMs was that they would not have helped us draw conclusions directly due to their black-box nature.
   Also, the embeddings of the comments using doc2vec API were initially added to the feature space but the trade off between the improvement on the accuracy and the added computation time, and also the incapability to draw conclusions from the embedding features led us to not consider them at a later stage of evaluation.

Future scope:
The comment dynamics are very complex to capture. It would indeed need a lot more work to improve the accuracy further. Perhaps, more features to capture the contextual information like different embeddings for comments can be tried. Also, efforts can be made to improvise the LSTM model, to utilize their potential in the field of Language processing better. The auto-sklearn implementation can be extended to more training points and the training points can be wisely selected by selecting distinct points (using methods like k-means clustering and taking the centroids or points around them) from the set.

Acknowledgments

We would like to thank Prof. Lyle Ungar for his valuable guidance and providing this opportunity to work on a project. It has indeed helped us learn a lot by implementing what we had learnt in the class, on a real-world problem. We would also like to thank the Teaching Assistants for their support during our struggles.

References

[1] https:/https://www.nltk.org/api/nltk.sentiment.html.

[2] XGBoost parameters documentation https://xgboost.readthedocs.io/en/latest/parameter.html.

[3] New York Times Comments Dataset.

[4] Predicting the selection of comments on nyt articles as editor's picks. https://www.kaggle.com/aashita/predicting-nyt-s-pick.

[5] Ng A. Y. Blei, D. M. and M. J. Jordan. *Latent Dirichlet allocation.* Journal of Machine Learning Research, 2003.

[6] Ralf Krestel Carl Ambroselli, Julian Risch and An dreas Loos. *Prediction for the newsroom:Which articles will get the most comments?* Pro-ceedings of the 16th Annual Conference of the NorthAmerican Chapter of the Association for Computa-tional Linguistics (NAACL), 2018.

[7] Wouter Weerkamp Manos Tsagkias and Maarten de Rijke. *Predicting the volume of comments on online news stories*. Proceedings of the International Conference on Information and Knowledge Management (CIKM). ACM, New York, NY, USA, 2009.

[8] Jordan Segall and Alex Zamoshchin. *PREDICTING REDDIT POST POPULARITY*.