MIDTERM REPORT - CS 506 Computational Tools for Data Science

1. **Dataset -**
   - The dataset used was that of movie reviews and the summarized version of reviews for various movies, along with the rating of the movie given by the reviewer.
   - From the basic EDA that was already done:
     - train.csv shape is (139753, 9)
       - Had columns such as Id, Product Id, Helpfulness Numerator and Denominator, Summary, Text and the Score of the review.
     - test.csv shape is (17470, 2)
       - Has only the IDs and the score which are all NaN values.
     - The dataset had the following distribution of each rating associated with the reviews:
       - Score
       - 5.0 65313
       - 4.0 27818
       - 3.0 14482
       - 1.0 7361
       - 2.0 7309
     - Clearly, we can notice that this **dataset is skewed**. (towards 5 star rated reviews)

2. **Data Cleaning and Preprocessing:**
   - **Removed any nan values** in the Text and Summary columns of the train and test dataframes.
   - Processed the text and summary columns as follows:
     - Tokenize the text, convert to lowercase, **remove stopwords and lemmatize** the words.
     - Additionally remove digits from the text and summaries.

3. **Feature Extraction, Engineering and Model Building (various approaches):**
   - The main idea was to convert the text and summaries into embeddings/feature vectors that can then be feature engineered to then build a model.
   - I tried various approaches for this process.
   - Here are some of the approaches that I adopted:
     - My **first approach** was to use the **Counter function** from the collection package
       - In this approach I initialized two instances of the Counter function for text and summary.
       - For text I chose the top 200 most common words and for summary I chose the top 100.
       - I then concatenated these two feature vectors into one, resulting in a 300 dimensional feature vector of the counts of various words from the vocabulary of the text and summary.
       - I then built a XGBClassifier on this feature vector and got an RMSE of about 1.25.
       - I then used PCA on the feature vector reducing the feature space to 75 and then performed the same. The RMSE did not change much, in fact got worse.
       - I performed this with various different most common values for both the text and summary counters, but this approach hit a dead end.

In [9]:
```
word_counts_sum = Counter()
for sentence in trainX['Summary']:
    words = sentence.split()
    word_counts_sum.update(words)
```

In [10]:
```
word_counts_text = Counter()
for sentence in trainX['Text']:
    words = sentence.split()
    word_counts_text.update(words)
```

In [12]:
```
top_words_summary = word_counts_sum.most_common(100)
```

In [14]:
```
top_words_text = word_counts_text.most_common(200)
```

- **Second approach** was using **Glove embeddings** instead of Counters.
    - I loaded the glove embeddings and then created embeddings of the text and summaries of the movie reviews.
    - Upon doing this, I **concatenated the two embeddings** which resulted in a final feature vector of dimension 600.
    - After the train test splitting of the stacked feature vectors, I then trained a **XGBClassifier** as well as an Random Forest Classifer (which took forever to train) and then evaluated the RMSE scores for the two of them.
    - My best model with this pipeline gave the following results:
        - Accuracy on testing set = 0.6013411293290265
        - RMSE on testing set = 1.0756631215207266
    - The RFClassifier performed similarly to this and I even tried changing the hyperparameters, but was not able to get a good working model.

- For the **3rd approach**, I tried building multiple binary classifiers:
    - I converted both text and summary to glove embeddings.
    - The new approach I adopted was to create 5 binary classifiers that would help in classifying the final test set.
    - I created a dataset for each class, i.e, score 1, score 2, …, score 5.
    - Each of these datasets had the scores of only one class and 0 for every other class. For example:
    -

Summary	Text	Score	Helpfulness	ReviewLength	Text_Embeddings	Summary_Embeddings
unexplained anime review	anxious see uncut version kite called finally ...	0	0.500000	234	[-0.12357161, 0.08572641, -0.13271326, -0.0624...	[-0.36291003, -0.06073999, -0.194, -0.12302334...
great	movie okay great	0	0.000000	4	[-0.077345334, 0.013289015, -0.12594034, -0.16...	[-0.093846, 0.58296, -0.019271, -0.070072, 0.1...
technical problem dvd	like dinosaur collector edition dvd one wo pla...	1	0.083333	26	[0.0117825, 0.030195307, -0.07427306, -0.07504...	[-0.2628233, 0.07638634, 0.024054006, -0.17563...

    - After building the 5 new datasets, I concatenated the text and summary embeddings for each of the 5 datasets.
    - Using undersampling, and SMOTE, I tried to balance the 5 datasets in terms of the 0 class and the corresponding score class.
    - I then created individual train test splits on this data for each dataset and then built Binary classifiers using RF and XGBClassifier.
    - Each of the individual models had low RMSE scores and high accuracy, however when I predicted using each of them for the final test set and combined the scores, I got a very high RMSE.
    - Clearly I was doing something terribly wrong. Either I was overfitting heavily, or my models weren't good enough for a multi class problem.
    - Another thing I noticed was that the sampling technique wasn't good enough for text data. SMOTE doesn't work well for high dimensional data like text embeddings.

- For the **4th approach (data augmentation)**:
    - I augmented the summaries using **nlpaug** package. I ran it overnight to create new data samples for the classes 1, 2, 3 and 4.

    - I then appended the new augmented summaries with the original dataset's summaries, then followed the same procedure as the 2nd approach (glove embeddings only for summaries -> train test split ->model building)
    - This approach also was futile as it did not produce a low enough RMSE value on the test set.

- **5th and final approach** which was successful! (TFIDF+Regression)
    - In this approach, I first split my dataset into train and test.
    - I then created **TFIDF Vectorizer** instances for both the text and summary.
    - I fit the instances on the train split for both and summary and transformed the test split using the same instances.
    - I then stacked the train and test data individually. (for both text and summary)
    - **Since Classifiers did not work well for any of the procedures, I tried my luck with Regressors** (I know this sounds very counterintuitive to the problem statement)
    - I tried several regressors, but my best model was a **Ridge** Regressor.
    - I then performed a **grid search CV** using the ridge regressor and got the best working model yet.
    - **Best Hyperparameters: {'alpha': 10.0, 'solver': 'auto'}**
    - My best model gave a**n RMSE on testing set of 0.84376**
    - It did well on the test set when I submitted my scores to the Kaggle Competition as well! **RMSE - 0.84954**

4. Conclusion:

- I began the experiment with a simple approach of using Counter functions, then delved into some more complex word embedding representations like GLOVE, and TFIDF (with uni, bi and tri grams).
- I experimented with various different feature engineering techniques.
    - I applied PCA for high dimensional data (did not work well on the test sets).
    - I tried text augmenting using nlpaug to create new data points for the imbalance data classes.
    - I tried SMOTE and Sampling techniques for balancing the dataset.
    - I even tried creating separate binary classifiers to then build a final model.
    - Another approach I tried was ensembling. (Built two weak classifiers and then tried building a meta model that learned on the errors from the weak classifiers.)
    - After all classifier based approaches failed, I resorted to regressors.
    - I figured out that my best working model was obtained using a **Grid Search and Cross validation on the Ridge Regressor model.**

5. Future Scope:

- Few techniques I feel could help in boosting the model performances:
    - Performing Latent Dirichlet Allocation (LDA) on the text and using it as another feature vector.
    - Trying grid search with other regressors to obtain a more accurate model.
    - Using Glove embeddings for regressors and performing grid search.
    - Bag Of Words combined with Glove Embeddings could be another approach for the feature engineering step.
    - Ensembling regressors and building a final Meta regression model.
    - Ensembling regressors and building a final Classifier (with all the weak regressors as the predictors)
    - Therefore, there are surely plenty of different approaches out there that could help in boosting the model's performance