Indiana University Bloomington

Yelp Dataset Challenge

Report

Authors:
Suyash Santosh Poredi
Kaustubh Dattatraya Bhalerao
Raja Rajeshwari Premkumar

Faculty:
Xiaozhong Liu

A final report submitted in the fulfillment of the requirement of course ILS - Z534 Information Retrieval

In

School of Informatics, Computing and Engineering

Fall 2019

Abstract

Yelp Dataset Challenge

by
Kaustubh Dattatraya Bhalerao, Raja Rajeshwari Premkumar, Suyash Santosh Poredi

Yelp Dataset Challenge helps to analyze the data provided by Yelp and find out different patterns that will be useful to solve variety of problems. In this project we have 2 tasks. Recommend businesses to users using both information retrieval and collaborative filtering. Perform Sentiment Analysis on the reviews dataset to predict if the customer likes, dislikes or is neutral in the review.

Chapter 1

Introduction

Task 1: Recommend businesses to users. Data was divided into two sets training and test data. After pre-processing the data, for approach 1 we have used techniques like Bags of Words, POS tagging, query expansion, Lucene Analyzers and Similarities to give Top K recommendation to a user. To test the accuracy of the retrieved results, we have used Precision and Recall and F1 Score as evaluation metrics for random users. Approach 2 was implemented using Collaborative Filtering technique, where we used memory based model, namely, User-Based and Item-Based Collaborative filtering. To recommend the business to the active user we have used K nearest Neighbor algorithm and Pearson Correlation Similarity and Apache Mahout machine learning library. We have computed the evaluation of both the approaches considering 5 random users and testing the performance against the evaluation metrics mentioned above.

Task 2

Sentiment Analysis:

It involves the mining of text to identify the emotional tone behind it. This helps in extracting subjective information in the social media streams, thus helping the business understand the social sentiment of their product, service or brand when monitoring online conversations.

Our analysis classifies a text into one of the below 3 types of tones. Hence, it is a multi-class classification problem:

-1: Negative sentiment
0: Neutral sentiment
1: Positive sentiment

Keywords : Information Retrieval, Collaborative Filtering (User-Based, Item-Based), Sentiment Analysis

Chapter 2

Task 1

Recommend Business to User

Data Pre-processing :

- As the data is to huge and sparse we are considering only Restaurants Businesses for Phoenix City.
- Picked the Restaurants which have reviews more than 50
- For Training and Testing the Reviews data was divided on the basis of Date of the review. (Firstly we divided the data into 60-40 % respectively. But we got better results when data was split according to the dates.)

Approach 1 :

Information Retrieval:

Information retrieval addresses the information needs of users by delivering relevant pieces of information but requires users to convey their information needs explicitly. We are using bags of words technique to extract and store features for our IR Algorithm. A bag-of-words is a representation of text that describes the occurrence of words within a document.

Proposed Algorithm:

- We have used user's reviews to understand his preferences of restaurants.
- A Unique Query will be generated for each user to retrieve the best recommendations.
- To Generate this query we are extracting nouns and adjectives from the reviews the user has given.
- Only top 100 words are chosen using TFIDF scores of extracted words for expansion of the query.
- This Generated Query is used on Training Data to Recommend Business to User.
- Lucene is used for indexing and retrieve the information for recommendation.

Experiment Design:

- From Yelp Dataset we are first extracting the relevant information i.e reviews and business data into MongoDB.
- After that we are generating the Training and Testing Data on User Reviews as we are using it to understand the preferences of the users.
- Further we index the business data using Lucene. Here we are creating document for each business. For every Business we are storing its reviews. We are considering Nouns, Adjectives etc. which have high importance as compared to other parts of speech. To extract POS Stanford NLP library is used.
- After indexing, we are generating Unique query for each user considering all the reviews he has given. Again, we are only considering Nouns and Adjectives to form query words for each user.

- We have removed stop-words for handing the noisy words in the query.
- For query expansion, we are calculating TFIDF scores of each query words to extract TOP 100 words. Query expansion will improve the retrieval performance for understanding the users preference better.
- For calculating TFIDF scores for each word, we are using Lucene indexes which were generated on business data. By querying with the single word on that indexes we can find the TFIDF scores of each word and sort them accordingly.

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

$tf_{i,j}$ = number of occurrences of i in j
df_i = number of documents containing i
N = total number of documents

- TFIDF Formula :
- Standard Analyzer and LMJelinekMercerSimilarity is used to Recommend Top 'K' businesses to User.

Evaluation Metrics:

- We have considered random 5 users to whom business were recommended for evaluation.
- Below metrices were used to evaluate the recommendation.

Precision: True Positive (No of Business Recommended i.e. user verified)

True Positive + False Positive (Total no of Recommendation)

Recall: True Positive (No of Business Recommended i.e. user verified)

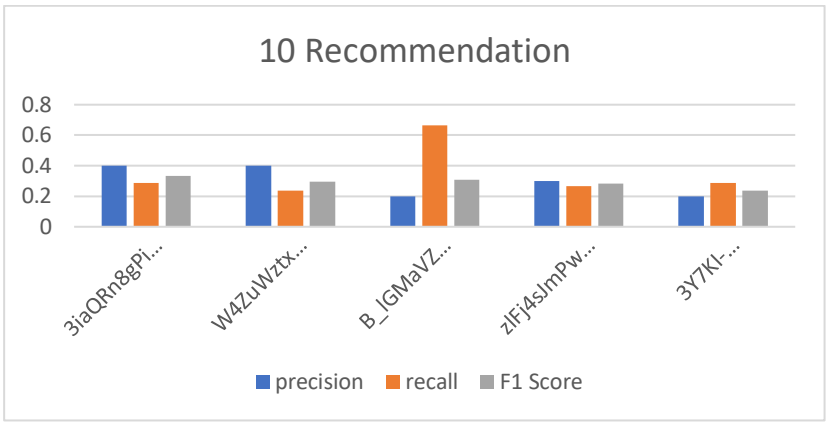
True Positive + False Negative (Actual Restaurants in Ground truth)

F1 Score: (Precision * Recall)
2*---
(Precision + Recall)

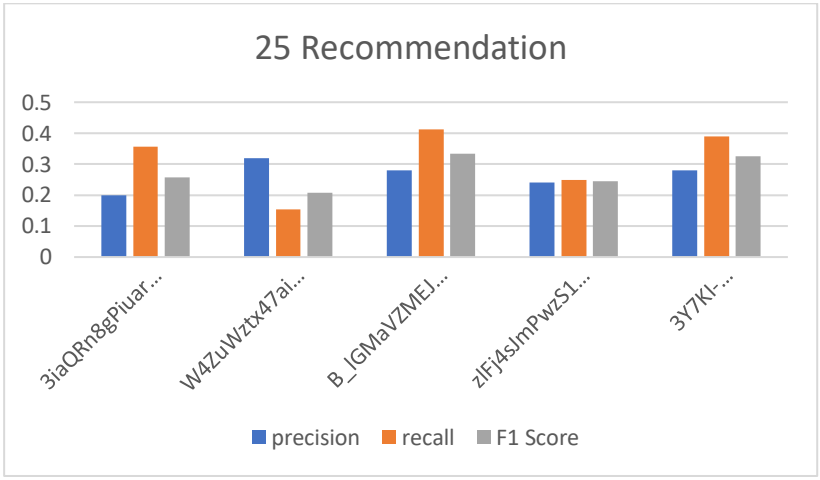
Evaluation Results :

- X Axis : 5 Random Users
- Y Axis : Numeric Values for Metrics
-

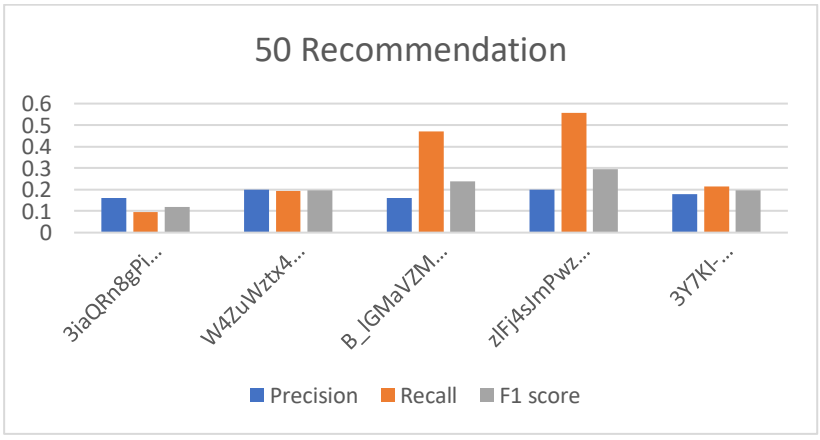
Top 10 Recommendation:



Top 25 Recommendation:



Top 50 Recommendation:



- Precision recall values when query considering Noun, Noun + Adjectives and Adjectives only were also calculated.
- The best values got were with the query formed using Noun + Adjectives.

User : 3iaQRn8gPiuarpYmv9dOJA	Top 10 (Precision)	Top 10 (Recall)	Top 25 (Precision)	Top 25 (Recall)	Top 50 (Precision)	Top 50(Recall)
Query (noun + adj)	0.4	0.285714286	0.2	0.357142857	0.16	0.091
Query (noun)	0.24	0.16	0.1342	0.2124	0.128342	0.145
Query (Adjectives)	0.157	0.1163	0.0935	0.15279	0.078	0.0684

Approach 2: Memory based Collaborative Filtering

In order to recommend business to users we have used memory based collaborative filtering approach. They are of two types namely, User based and Item based collaborative filtering. The prior one(User-based) has a focus on generating an user-item matrix and suggesting the business by evaluating the other users which are much more similar to the active user (the user to whom we want to recommend the business). While the later(Item-based), generates an item-item matrix and recommends business to the active user based on similar items. We have used rating, as a measure of evaluation.

2.3.1 **Experiment Design**:

- To implement collaborative filtering we have used non-hadoop based recommender engine inside Apache machine learning library: Mahout taste. It takes input the user's taste(preference for item) and returns the estimated preference for other items.
- The algorithm takes in user Id, the business Id and rating as an input, so we calculated the matrix for all the users and associated business(item) matrix in the form of csv, "userID, businessID, preferences(rating)". Thus, rating now denotes the strength of interaction.
- The recommender of the library takes input the matrix and uses an interface called DataModel, to handle the interaction between the data. It is thus used in evaluation given training data matrix.
- The next step would now be, to find the similar users, in order to do that we need to compare their interactions(rating). There are several methods to do so, we implemented some like, Tanimoto Coefficient similarity, Log Likelihood similarity, Euclidean Distance similarity, and Pearson Correlation similarity. After comparing various results, we found Pearson Correlation similarity as a best measure to compute their similarities, as they gave better evaluation result.
- User Based Collaborating Filtering Design:
    - The next thing, we had to do is to define which similar users do we want to leverage for the active user(recommender), In order to do so, we have used Nearest Neighbor algorithm, we had set a neighborhood scale of 25, selecting 25 most similar people.

This is implemented via ThresholdUserNeighboorhood Class. We passed in Pearson-correlation similarity as a parameter to the constructor as a means of similarity measure.

- Now it's time to design the recommender, as we have all the pieces needed to do it, we have used GenericUserBasedRecommender which has a method named recommender which evaluates for the active user based on how many neighbours and a 'rescorer' function to recalculate the scores and return the list of recommended items ordered from most strongly recommended to the least for best 25 neighbours.
- Depending on how many business we have to recommend we can ask the recommender for recommendations. For this task, we have generated top 10, top 25 and top 50 recommendation of business to the user.

- Item Based Collaborative Filtering:
  - The steps are same as of User-based collaborative filtering. We have used the same similarity measure, Pearson Correlation Similarity.
  - The recommender model for the item-based collaborative filtering changes a bit, here we have used ItembasedRecommender class, we don't use neighborhood definition for item-based recommender.
  - The recommender then evaluates based on matrix and the similarity measure and returns the top 10, top 25 and top 50 recommendations for the active user.

Similarity Evaluation Formulae:

- Pearson Correlation Similarity:
  - It's the measure of linear co-variance between two user's rating.
  - Positive Corelation: +1, Little Cor-relation -0 and Negative Co-relation:- 1

- Formulae:
  - $PC(X,Y) = \frac{\text{Sum XY}}{\text{Sqrt (Sum X^2 + Sum Y^2)}}$

X and Y represents the user.

Sum XY represents sum of the product of X's and Y's (ratings) for all items for which both X and Y express an equal rating.

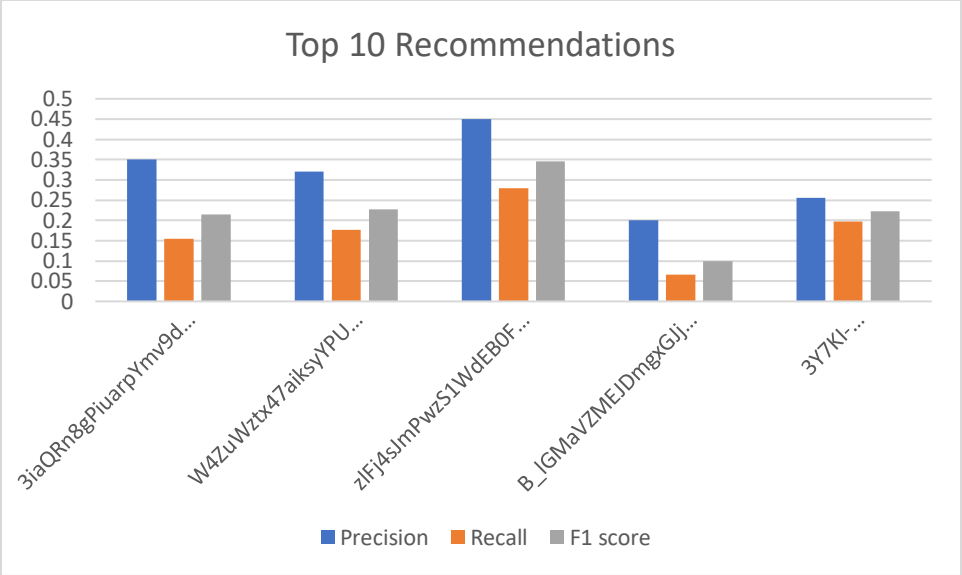
X^2 sum of all squares of all X's ratings.

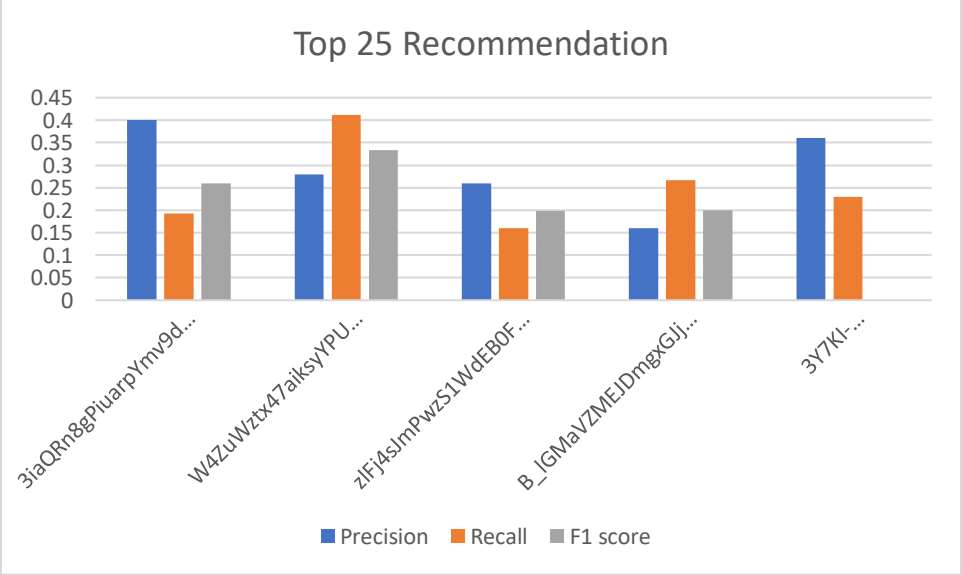
Y^2 sum of all squares of all Y's ratings.

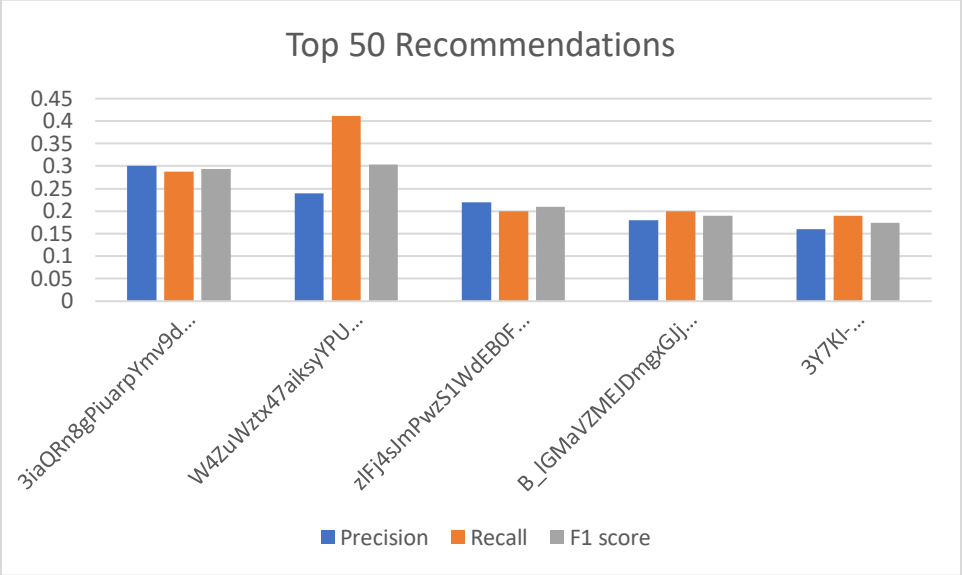
Based on this we get to choose, the most similar users which have given equal preference.

Evaluation Metrics:

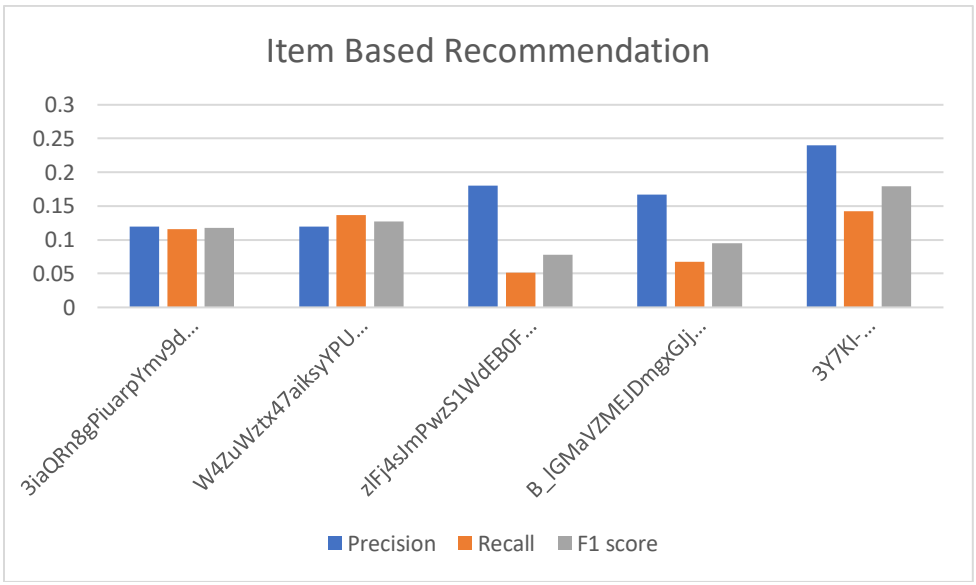
- **User Based Collaborative Filtering Evaluation**:
  - For the evaluation of this approach we have used the same business users and the same evaluation metrics like Precision, Recall and F1 score.
  - The following are the results for:- User Based Collaborative Filtering Approach.
  - On X axis, I have placed the 4 users and on Y axis are the numeric values:-







- **Item Based Collaborative Filtering Evaluation:**



Conclusion and Future Work

As you can see from the evaluation result, we are getting better precision for approach 2 i.e Collaborative filtering whereas we are getting better recall values for approach 1 i.e Information retrieval as we are considering reviews for giving recommendation.

We were trying to implement SVD(Singular Vector Decomposition) model for collaborative filtering, we were not able to generate eigendecomposition of a normal matrix that is needed as an input to the algorithm. Because of this problem, we skipped the implementation

Chapter 3

Task 2

Sentiment Analysis:

We consider the data from the year 2006. It has approx. 6000 data points. Below are the steps we follow to preprocess and vectorize the data and to build a model.

Preprocessing

1. We work with only the reviews dataset.
2. We consider the columns "text" and "stars" only.
3. The column "stars" has ratings ranging from 1 to 5.
4. Convert ratings less than 3 to -1, rating 3 to 0 and ratings above 3 to label 1.
5. The distribution of the ylabel across the data is not balanced. See below:

1	40445
0	11964
-1	9144

6. Split the data to train and test
7. Perform undersampling on the train set. Test set is left unbalanced.
8. Preprocess the data:
    1. Remove non alphabetic characters (except hyphen and space)
    2. Lower case the text and tokenize the words.
    3. Remove insignificant words such as stop words and words of length 1

Vectorization

We have applied two types of vectorization technique here:
1. Frequency based embedding:
2. Semantic Meaning based embeddings:

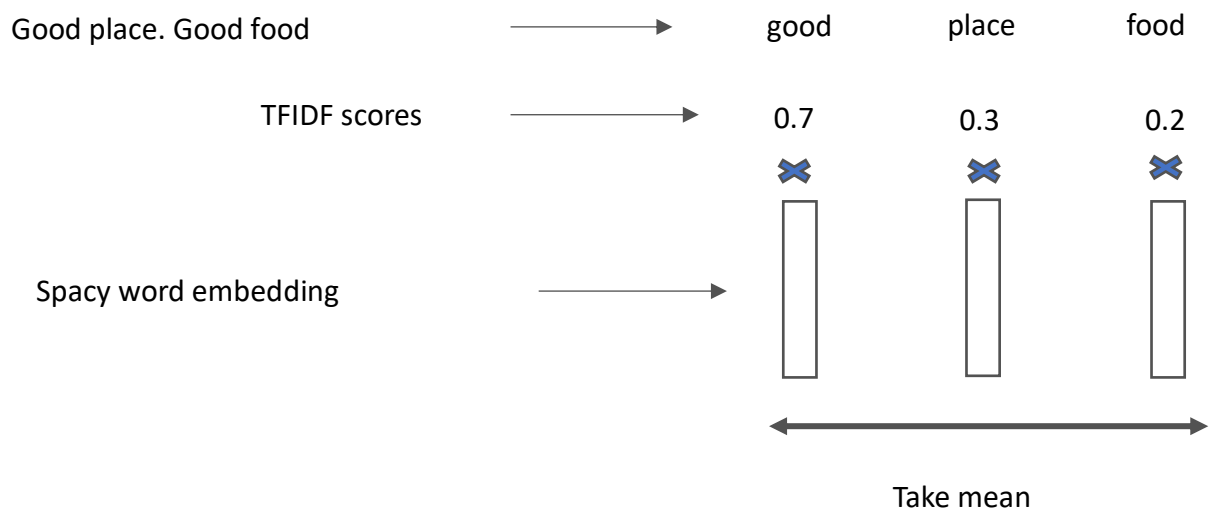
1. Frequency based embedding:
We use TFIDF count for each word in a text as embedding. TFIDF helps to give more weightage to the words that have less document frequency. The maximum document frequency is set to 0.8 and the minimum document frequency is set to integer 5 i.e. 5 documents. Below is the TFIDF calculation followed:

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

$tf_{i,j}$ = number of occurrences of i in j
df_i = number of documents containing i
N = total number of documents

2. Semantic Meaning based embedding:
Here we use the spacy glove embedding from the en_core_web_lg which is the largest English model of Spacy. It has 300 dimensions. Below is the steps followed for this approach:

1. For a list of words in a text review we do the following for each word:
    a. Find the tfidf score for that word in the document
    b. Multiply the tfidf score with the glove embedding of the word
2. Take the sum of the above vector across all the words vectors i.e. for each and every dimension in the spacy embedding, the score is summed up across all the words in the text.
3. Take the mean of the above summed up vector.



MODELS IMPLEMENTED:

Below are the steps followed for model building:

1. We have implemented 5 models for each one of the vectorization techniques.
   The file SearchProjectTFIDF.ipynb has the results for the frequency based embedding.
   The file SearchProjectEMBED.ipynb has the results for the semantic embedding.

2. Below are the 5 models implemented using SKLEARN:
   - Random Forest: Bootstrapping and bagging decision trees
   - Decision Tree: A flowchart model where the prediction is made based on yes and no conditions.
   - AdaBoost: Ensemble of weak learners. It involves boosting stumps.
   - XGBoost: Gradient Boosting variant that follows level wise tree growth.
   - LightGBM: Gradient Boosting variant that follows leaf wise tree growth.

3. We perform Random Search with different hyperparameters values for the models. This is similar to Grid Search but it randomly selects the hyperparameters from the entire sample space.
4. We use the F1-Score as metric to obtain the best hyperparameter value. We also display below the metrics – Precision, Recall, F1_score and accuracy for the different models.
5. Below is how the F1_Score calculated for this multi-class problem:
   a. Calculate Precision/Recall for class 1. We do this by considering other classes as negatives and 1 as positive. Similarly, we do the same for other classes (i.e. 0 and -1) as well.
   b. Calculate the mean of Precision/Recall across all the classes.
   c. Calculate F1-Score using the mean Precision/Recall.

Results:

BASE MODELS:

The models here are not tuned using the Search CV technique:

RANDOM FOREST:

FREQUENCY BASED

```
       PRECISION    RECALL  F1_SCORE  ACCURACY
Train   0.982255  0.982196  0.982169  0.982196
Test    0.360634  0.376519  0.305145  0.365000
Train confusion marix
[[672   0   2]
 [  8 660   6]
 [  7  13 654]]
Test confusion matrix
[[10  6  4]
 [16  9 10]
 [43 48 54]]
```

SEMANTIC MEANING BASED

```
       PRECISION    RECALL  F1_SCORE  ACCURACY
Train   0.991619  0.991592  0.991592  0.991592
Test    0.413828  0.504516  0.390594  0.440000
Train confusion marix
[[672   2   0]
 [  4 668   2]
 [  3   6 665]]
Test confusion matrix
[[15  2  3]
 [11 12 12]
 [33 51 61]]
```

DECISION TREE:

FREQUENCY BASED

```
      PRECISION    RECALL  F1_SCORE  ACCURACY
Train   1.000000  1.000000  1.000000     1.000
Test    0.367714  0.372824  0.335606     0.435
Train confusion marix
[[674   0   0]
 [  0 674   0]
 [  0   0 674]]
Test confusion matrix
[[ 7 10  3]
 [11 10 14]
 [33 42 70]]
```

SEMANTIC MEANING BASED

```
      PRECISION    RECALL  F1_SCORE  ACCURACY
Train   1.000000  1.000000  1.000000       1.0
Test    0.376595  0.435961  0.351096       0.4
Train confusion marix
[[674   0   0]
 [  0 674   0]
 [  0   0 674]]
Test confusion matrix
[[10  6  4]
 [ 4 15 16]
 [40 50 55]]
```

XGBOOST Classifier:

FREQUENCY BASED

SEMANTIC MEANING BASED

```
       PRECISION    RECALL  F1_SCORE  ACCURACY
Train   0.779743  0.777943  0.777422  0.777943
Test    0.439818  0.513218  0.420600  0.490000
Train confusion marix
[[548  57  69]
 [ 97 482  95]
 [ 75  56 543]]
Test confusion matrix
[[13  5  2]
 [11 14 10]
 [29 45 71]]
```

```
       PRECISION    RECALL  F1_SCORE  ACCURACY
Train   0.900584  0.900593  0.900587  0.900593
Test    0.447009  0.529228  0.438965  0.525000
Train confusion marix
[[611  33  30]
 [ 38 601  35]
 [ 27  38 609]]
Test confusion matrix
[[14  4  2]
 [12 12 11]
 [27 39 79]]
```

ADABOOST Classifier:

FREQUENCY BASED

```
       PRECISION    RECALL  F1_SCORE  ACCURACY
Train   0.621913  0.590504  0.594803  0.590504
Test    0.425756  0.444335  0.385664  0.465000
Train confusion marix
[[357 258  59]
 [ 99 447 128]
 [ 41 243 390]]
Test confusion matrix
[[ 8 11  1]
 [11 16  8]
 [22 54 69]]
```

SEMANTIC MEANING BASED

```
       PRECISION    RECALL  F1_SCORE  ACCURACY
Train   0.586069  0.587043  0.586540  0.587043
Test    0.407598  0.470608  0.390586  0.460000
Train confusion marix
[[468 165  41]
 [166 293 215]
 [ 42 206 426]]
Test confusion matrix
[[12  6  2]
 [ 8 12 15]
 [31 46 68]]
```

LIGHTGBM Classifier:

FREQUENCY BASED

```
       PRECISION    RECALL  F1_SCORE  ACCURACY
Train   0.976416  0.976261  0.976247  0.976261
Test    0.442930  0.526929  0.443187  0.520000
Train confusion marix
[[666   5   3]
 [ 15 648  11]
 [  9   5 660]]
Test confusion matrix
[[14  3  3]
 [ 9 12 14]
 [23 44 78]]
```

SEMANTIC MEANING BASED

```
       PRECISION    RECALL  F1_SCORE  ACCURACY
Train   1.000000  1.000000  1.000000      1.00
Test    0.434693  0.498851  0.417047      0.49
Train confusion marix
[[674   0   0]
 [  0 674   0]
 [  0   0 674]]
Test confusion matrix
[[12  5  3]
 [11 14 10]
 [26 47 72]]
```

BETTER MODELS:

Here the models are tuned using the Random Search CV technique discussed before:

ADABOOST:

FREQUENCY BASED

```
       PRECISION    RECALL  F1_SCORE  ACCURACY
Train   0.610052  0.563304  0.563699  0.563304
Test    0.423341  0.437192  0.385391  0.465000
Train confusion marix
[[282 308  84]
 [ 72 456 146]
 [ 25 248 401]]
Test confusion matrix
[[ 7 11  2]
 [ 9 17  9]
 [17 59 69]]
```

SEMANTIC MEANING BASED

```
       PRECISION    RECALL  F1_SCORE  ACCURACY
Train   0.621838  0.617211  0.619184  0.617211
Test    0.439007  0.506650  0.412871  0.460000
Train confusion marix
[[464 155  55]
 [139 355 180]
 [ 37 208 429]]
Test confusion matrix
[[13  6  1]
 [ 9 15 11]
 [24 57 64]]
```

LIGHTGBM Classifier:

FREQUENCY BASED

```
       PRECISION    RECALL  F1_SCORE  ACCURACY
Train   0.484467  0.483680  0.476394   0.48368
Test    0.408319  0.462233  0.411229   0.52000
Train confusion marix
[[325 134 215]
 [193 229 252]
 [138 112 424]]
Test confusion matrix
[[10  2  8]
 [ 7 11 17]
 [27 35 83]]
```

SEMANTIC MEANING BASED

```
       PRECISION    RECALL  F1_SCORE  ACCURACY
Train   0.857439  0.855589  0.855449  0.855589
Test    0.438463  0.522003  0.436089  0.525000
Train confusion marix
[[578  43  53]
 [ 47 547  80]
 [ 37  32 605]]
Test confusion matrix
[[14  3  3]
 [11 11 13]
 [26 39 80]]
```

DECISION TREE:

FREQUENCY BASED

SEMANTIC MEANING BASED

```
       PRECISION    RECALL  F1_SCORE  ACCURACY
Train   0.558606  0.526706  0.530222  0.526706
Test    0.381432  0.383005  0.341238  0.410000
Train confusion marix
[[308 263 103]
 [108 401 165]
 [ 36 282 356]]
Test confusion matrix
[[ 7 10  3]
 [ 9 13 13]
 [19 64 62]]
```

```
       PRECISION    RECALL  F1_SCORE  ACCURACY
Train   0.463729  0.456973  0.410806  0.456973
Test    0.368285  0.422824  0.373721  0.560000
Train confusion marix
[[326  42 306]
 [184  75 415]
 [ 94  57 523]]
Test confusion matrix
[[10  0 10]
 [10  3 22]
 [27 19 99]]
```

RANDOM FOREST:

FREQUENCY BASED

```
       PRECISION    RECALL  F1_SCORE  ACCURACY
Train   0.729480  0.729476  0.728242  0.729476
Test    0.437409  0.526847  0.427589  0.520000
Train confusion marix
[[517  82  75]
 [104 431 139]
 [ 56  91 527]]
Test confusion matrix
[[15  1  4]
 [15 10 10]
 [30 36 79]]
```

SEMANTIC MEANING BASED

```
       PRECISION    RECALL  F1_SCORE  ACCURACY
Train   0.562957  0.560336  0.542262  0.560336
Test    0.405998  0.489080  0.385394  0.485000
Train confusion marix
[[498  71 105]
 [225 203 246]
 [163  79 432]]
Test confusion matrix
[[15  3  2]
 [13  7 15]
 [44 26 75]]
```

XGBOOST CLASSIFIER:

FREQUENCY BASED

```
       PRECISION   RECALL  F1_SCORE  ACCURACY
Train   0.760157  0.75816  0.757281   0.75816
Test    0.415798  0.48399  0.404908   0.50500
Train confusion marix
[[529  67  78]
 [103 460 111]
 [ 74  56 544]]
Test confusion matrix
[[13  5  2]
 [12  9 14]
 [31 35 79]]
```

SEMANTIC MEANING BASED

```
       PRECISION    RECALL  F1_SCORE  ACCURACY
Train   0.908116  0.908012  0.907896  0.908012
Test    0.450351  0.541379  0.446503  0.520000
Train confusion marix
[[603  40  31]
 [ 36 601  37]
 [ 24  18 632]]
Test confusion matrix
[[14  3  3]
 [ 9 14 12]
 [28 41 76]]
```

Observations and Conclusion:

1. In almost all cases including the base models, the semantic meaning-based embedding gives better performance. This is because we are leveraging two features here i.e. the

Count TFIDF and the TFIDF weighed embedding. The embedding feature adds additional information to the count feature- hence the improvement.

2. This improvement can be better observed if we had used more data or if the tfidf minimum document frequency was even lower than the value set i.e. 5. This is because more words add more meaning to the semantic based embedding. Change in min_df did not affect the Frequency based model as much as it did the Semantic based model.
3. The best model is XGBOOST with 0.446 f1_score and 0.52 accuracy. Please note that the value is too low due to small data size. The results on the full dataset gave 0.65 f1_score with base model itself. Results highlighted in the ppt.