Politecnico di Torino

Machine Learning for Networking

01DSMBG

Project Report
Patterns in comments from Reddit social network

professors:

Prof. Luca Vassio
Dr. Giordano Paoletti

Authors:

Sadegh Jamishi - S314215
Ali Makki - S306604
Olivier Jasson - S321275

Academic Year 2023/2024

Table of Contents

Introduction

Project Inspiration: Everything we express (either verbally or in written) carries huge amounts of information. The topic we choose, our tone, our selection of words, everything adds some type of information that can be interpreted and value extracted from it. In theory, we can understand and even predict human behaviour using that information.

Natural Language Processing or NLP is a field of Artificial Intelligence that gives the machines the ability to read, understand and derive meaning from human languages.

Data generated from conversations, declarations or even tweets are examples of unstructured data. Unstructured data doesn't fit neatly into the traditional row and column structure of relational databases, and represent the vast majority of data available in the actual world. It is messy and hard to manipulate. Nevertheless, thanks to the advances in disciplines like machine learning a big revolution is going on regarding this topic. Nowadays it is no longer about trying to interpret a text or speech based on its keywords (the old fashioned mechanical way), but about understanding the meaning behind those words (the cognitive way). This way it is possible to detect figures of speech like irony, or even perform sentiment analysis.
Before proceed with an introduction of our project, we want to shortly explain some of the most frequently used algorithms in our project , or even used at any NLP project:

- **Bag of Words:** It is a commonly used model that allows you to count all words in a piece of text. Basically it creates an occurrence matrix for the sentence or document, disregarding grammar and word order.
- **Term Frequency — Inverse Document Frequency (TFIDF):** It improves the bag of words algorithm by using the weights. Since Bag of Words may reflect several downsides like the absence of semantic meaning and context, and the facts that stop words (like "the" or "a") add noise to the analysis and some words are not weighted accordingly, so through TFIDF frequent terms in the text are "rewarded", but they also get "punished" if those terms are frequent in other texts we include in the algorithm too
- **Tokenization:** This is the process of segmenting running text into sentences and words. In essence, it's the task of cutting a text into pieces called tokens, and at the same time throwing away certain characters, such as punctuation
- **Stop Words Removal:** It Includes getting rid of common language articles, pronouns and prepositions such as "and", "the" or "to" in English. In this process some very common words that appear to provide little or no value to the NLP objective are filtered and excluded from the text to be processed.
- **Stemming:** It refers to the process of slicing the end or the beginning of words with the intention of removing affixes (lexical additions to the root of the word).
- **Lemmatization:** It Has the objective of reducing a word to its base form and grouping together different forms of the same word. For example, verbs in past tense are changed into present (e.g. "went" is changed to "go") and synonyms are unified (e.g. "best" is changed to "good"), hence standardizing words with similar meaning to their root.

[1]

ML4N - Group Project 2

Reddit is an entertainment, social networking, and news website where registered community members can submit content, such as text posts or direct links, making it essentially an online bulletin board system. Registered users can then vote submissions up or down to organize the posts and determine their position on the site's pages. Content entries are organized by areas of interest called "subreddits". The subreddit topics include news, gaming, movies, music, books, fitness, food, and photosharing, among many others.
When items (links or text posts) are submitted to a subreddit, users can vote for or against them (upvote/downvote). Each subreddit has a front page that shows newer submissions that have been rated highly. Users can also post comments about the submission and respond back and forth in a conversation-tree of comments; the comments themselves can also be upvoted and downvoted. The front page of the site itself shows a combination of the highest-rated posts out of all the subreddits a user is subscribed to. The project uses machine learning techniques to look for trends in Reddit comments. It involves studying a dataset of 20,000 users' 1.4 million comments, with a particular emphasis on gender prediction and content-based categorization. Four elements comprise the project: preprocessing and data exploration; supervised learning for gender classification; unsupervised learning for comment clustering; and a deep learning-based portion. The goal of using techniques like text vectorization, PCA, and fine-tuning of pre-trained models is better grasp Reddit user behaviors and features.

[1] https://towardsdatascience.com/your-guide-to-natural-language-processing-nlp-48ea2511f6e1

1 Section 1 – Data Exploration and Pre-processing

1.1 Exploratory Data Analysis (EDA)

As the first step toward the project, we started by exploring the provided supervised dataset containing 5000 rows as the data and 4 columns: author, subreddits, submission date, and body of the comments respectively. Here are all of the findings by us:

- Number of unique elements and the number of missed (NaN) values within each of the features is as shown in Table 3.

Feature	Number of Unique Values	Number of Missed Values
Author	5000	0
Subreddit	3468	0
Created_utc	278142	0
Body	289608	0

Table 1: Number of unique values and missed values for each feature

The number of total samples in the dataset is 296042.

- Distribution of Number of characters per Comments

It can be seen that the majority of comments are short, with most having fewer than 500 characters.

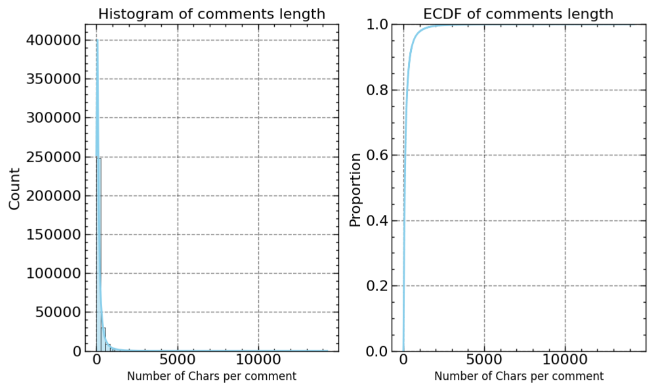


Figure 1: Distribution of the Number of Characters per Comments

The left plot, the histogram, shows that most comments are relatively short. There's a sharp peak at the lower end, indicating that the majority of comments have a small number of characters. Moreover, the distribution is skewed to the right, meaning that there are a few comments with a very large number of characters. The right plot, ECDF, shows that a large proportion of comments (close to 100%) have lengths less than or equal to a few thousand characters. As seen from the plot, it rises sharply at the beginning, indicating that most comments are very short and then flattens out quickly, showing that comments with lengths beyond a few thousand characters are very uncommon.

- Distribution of the Number of Comments per Author

There is a significant variation in the number of comments per author. Some authors are highly active, while others have very few comments. This distributions, which have been visualized by using histograms and ECDF plots, shows that a small number of authors are responsible for a large portion of comments.

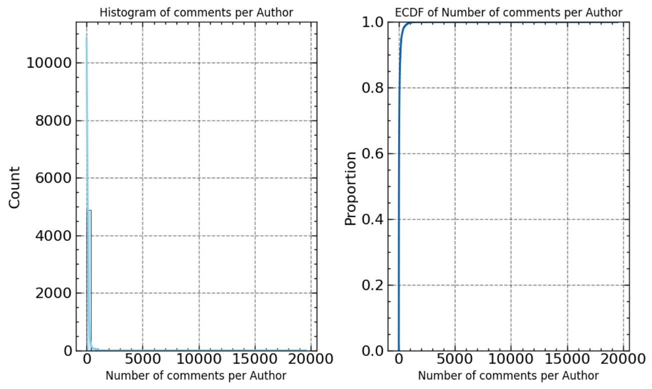


Figure 2: Distribution of the Number of Comments per Author

The left plot, histogram, has a sharp peak at the lower end, indicating that most authors have made a relatively small number of comments.The distribution is skewed to right side, with a long tail extending towards higher numbers of comments meaning that a few authors have made a very large number of comments. The right plot, ECDF, shows that a large proportion of authors have made only a few comments. As seen from the plot, it rises sharply at the beginning, indicating that authors with a very high number of comments are very rare and then flattens out quickly, showing that authors with number of submitted comments beyond a few thousand are very uncommon.

- Gender Balance of the Train Dataset

The dataset is imbalanced with 3,651 males and 1,349 females. This shows a bias toward the more represented class which is the male.

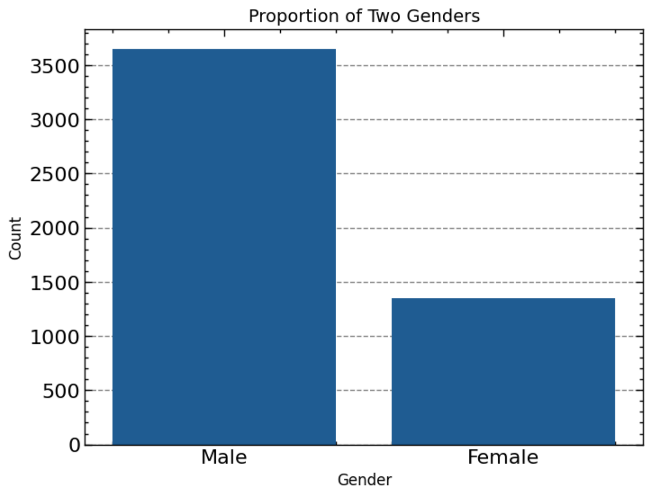


Figure 3: Distribution of the two gender within the supervised dataset

- Language Diversity in Comments

Using the `googletrans` and `langdetect` libraries, the first 100 of comments have been sampled and analyzed for language diversity. Results from both libraries showed that while the majority of comments are in English, the `langdetect` library found a small proportion in other languages like Swedish, Afrikaans, Tagalog, and Indonesian, which makes it an unreliable library for detection.

Library	Detected languages
`googletrans`	'en', 'mi'
`langdetect`	'en', 'pt', 'et', 'ro', 'so', 'sq', 'id', 'cy', 'tl', 'da', 'nl'

Table 2: Detected languages for the same comments by two libraries

1.2 Data Cleaning and Text Standardization

From this part, we started to delve into words and manipulate them. We mainly implemented two things:

1. **Uniform format for the text:** We implemented normalization in order to be sure that all text is processed in lowercase, helping in reducing complexity and variability in the data. We also used the `contractions` library to expand contractions to their full forms in such a way that provide consistency in textual analysis.

2. **Data cleaning:** Using the `nltk` library, we removed the stop words to focus on more meaningful words in the text. Moreover, we got rid of distracting elements like email addresses, URLs, excess whitespaces, and punctuations. Initially, we handled these tasks using the `regex` library, but it was too difficult to handle all these cases manually, so we used the `nltk` and `contractions` libraries.

```
clean_data['body'] = train_data['body'].apply(lambda x: re.sub('^a-zA-Z0-9 -', '', x))
```

1.3 Text Vectorization

Here is a summary of the steps we did during this section:

1. **Grouping Comments by Author:** We aggregated all comments per author to consolidate the textual data and ensure that each author's comments were combined into a single text string.

2. **Creating Subreddit Features:** A sparse matrix was created for subreddit features where each column represented a subreddit. The resulting matrix shows whether an author had posted in a particular subreddit.

3. **Vectorization of Text Data:** We converted the aggregated text data into numerical representations by using two main techniques:

   (a) **Bag of Words (BoW):** We first transformed the text into a Bag of Words model, which counts the number of times each word appears in a document.

   (b) **TF-IDF Transformation:** Then we applied TF-IDF weighting to the Bag of Words counts. This method enhances the feature space by reducing the weight of more common words and giving higher importance to words that are more unique to specific documents.

4. **Combining Feature Sets:** The final step was to horizontally stack the subreddit features (X_subreddits) and the TF-IDF features (X_body) into a single feature matrix (X). The resulting matrix is a sparse matrix with 5000 rows and 102442 columns and will be used then for subsequent machine learning tasks.

1.4 Reduce the Complexity of the Dataset

Here, we focused on reducing the complexity of our dataset by applying PCA to the high-dimensional feature matrix generated from our text and subreddit matrices. However, in order to do this, especially with sparse matrices, we had to pay careful attention to two things:

1. Normally, PCA requires centering data by subtracting the mean of each feature. However, doing such operation on a sparse matrix will probably convert it to a dense matrix, resulting in an increase in memory usage and therefore the computational time.

2. Since standard PCA is not a good choice to deal with the sparse feature matrix, we used an alternative method called TruncatedSVD. It performs linear dimensionality reduction by using truncated singular value decomposition and is also compatible with the sparse matrices.

Therefore, here's how we proceed:

1. We Started TruncatedSVD with 1500 components(1000 PCs mentioned at the project file was too small to reach the 95% of variance of data.)

2. For the initial value of 1500 components, we reached to 91.27% of the explained variance ratio

3. We gradually increased the number of the SVD components and checked the resulting variance ratio in order to satisfy our threshold value.

4. Having 2300 components we reached to 95.31% of the explained variance ratio representing preserve most of the variance of the data.

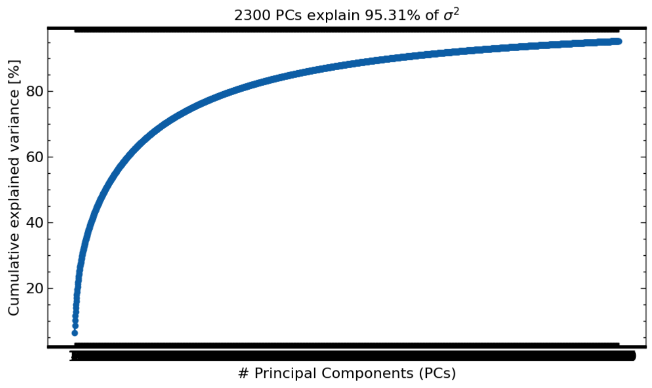


Figure 4: 2300 PCs explain 95.31% of σ^2

5. We also found out that from these 2300 PCs, 93 of them have the cumulative explained variance value greater than 95, indicating that there are 93 principal components that together explain more than 95% of the total variance in the dataset.

2 Section 2 – Supervised Learning – Classification

In the second part of our project, we have built a general framework that allows for the selection and evaluation of multiple classifiers. The steps we took include:

- **Model Selection:** The `model_selector` function is used for fetching different classifiers based on the given string class.
- **Metric Evaluation:** The `evaluate_metrics` function fits the model, predicts outcomes, and calculates the required performance metrics.
- **Results Presentation:** Using the `print_metrics` and `plot_confusion_matrix` functions, we present detailed results and visualizations for each model's performance.
- **Data Splitting:** We are splitting both the original and SVD-reduced datasets and running evaluations on each in order to conduct a thorough examination of how dimensionality reduction impacts model performance. This is crucial for understanding the trade-offs associated with such preprocessing steps.

2.1 Splitting the Dataset

The goal of this part was to split the dataset and corresponding labels of the data into a train and test set. We have separated, independently, the original dataset and the resulting reduced dataset of the SVD output, into two parts each as described below:

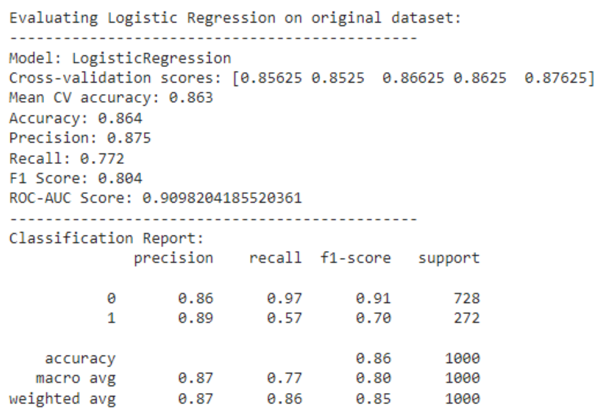
```
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)
X_svd_train, X_svd_test, y_train, y_test = train_test_split(X_svd, y, test_size=0.2, random_state=42)
```

We have separated, independently, the original dataset and the resulting reduced dataset from the SVD output into two segments. From now on, we apply the rest of the tasks inside this section to both the original and the reduced dataset in order to compare the results.

2.2 Machine Learning Methods

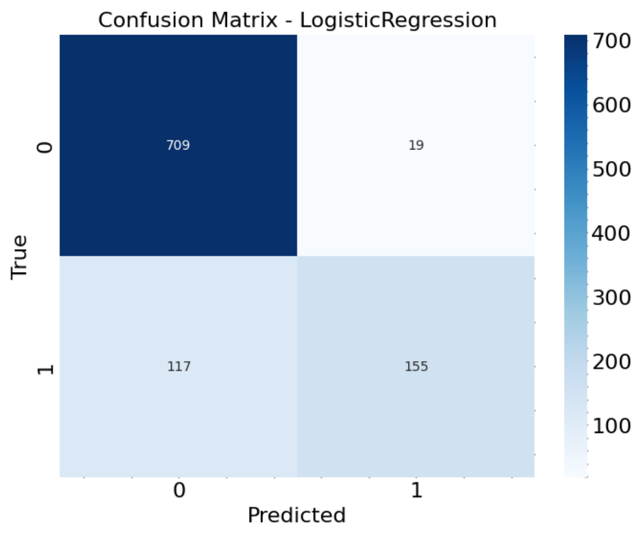
In this part, we executed various classifiers on two versions of our dataset: the original feature matrix and the SVD-reduced feature matrix. Here is a full description of the results obtained in this part:

1. **Logistic Regression:** We ran the classifier on the two datasets separately, starting with the original dataset:



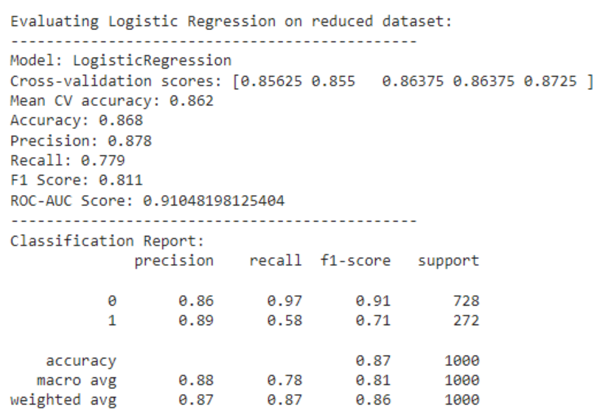
```
Evaluating Logistic Regression on original dataset:
---------------------------------------------
Model: LogisticRegression
Cross-validation scores: [0.85625 0.8525  0.86625 0.8625  0.87625]
Mean CV accuracy: 0.863
Accuracy: 0.864
Precision: 0.875
Recall: 0.772
F1 Score: 0.804
ROC-AUC Score: 0.9098204185520361
---------------------------------------------
Classification Report:
              precision    recall  f1-score   support

           0       0.86      0.97      0.91       728
           1       0.89      0.57      0.70       272

    accuracy                           0.86      1000
   macro avg       0.87      0.77      0.80      1000
weighted avg       0.87      0.86      0.85      1000
```

(a) Logistic Regression Results on Original Dataset



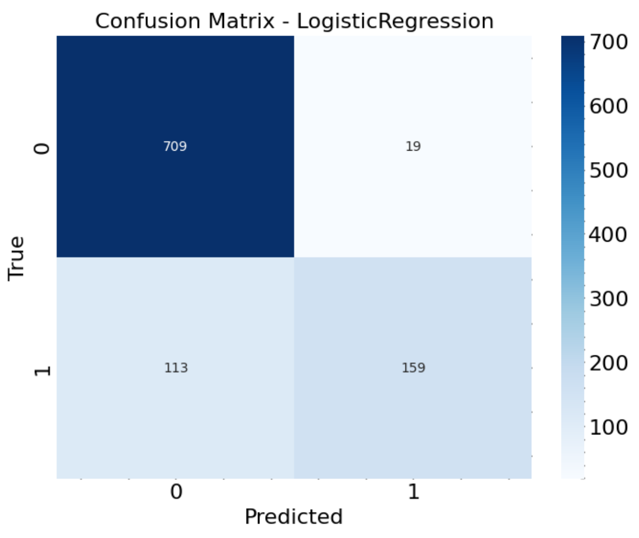
(b) Confusion Matrix of Logistic Regression on Original Dataset

And then on the reduced dataset:



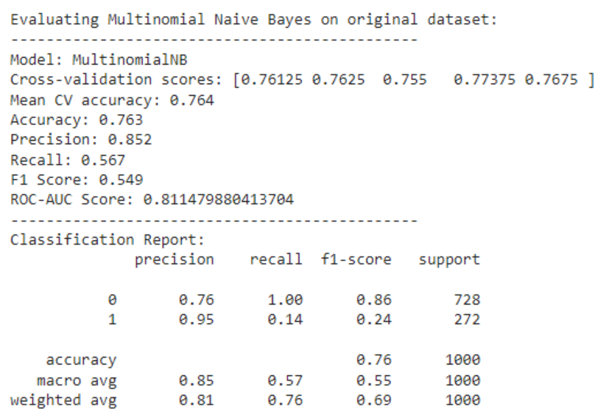
```
Evaluating Logistic Regression on reduced dataset:
----------------------------------------------
Model: LogisticRegression
Cross-validation scores: [0.85625 0.855   0.86375 0.86375 0.8725 ]
Mean CV accuracy: 0.862
Accuracy: 0.868
Precision: 0.878
Recall: 0.779
F1 Score: 0.811
ROC-AUC Score: 0.91048198125404
----------------------------------------------
Classification Report:
              precision    recall  f1-score   support

           0       0.86      0.97      0.91       728
           1       0.89      0.58      0.71       272

    accuracy                           0.87      1000
   macro avg       0.88      0.78      0.81      1000
weighted avg       0.87      0.87      0.86      1000
```

(a) Logistic Regression Results on Reduced Dataset



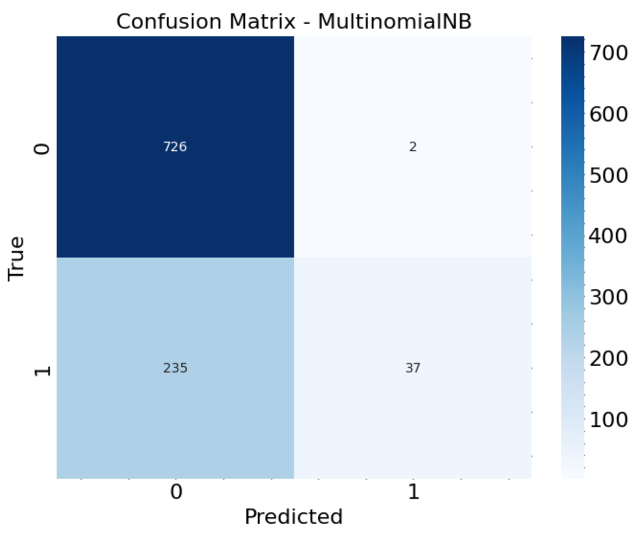
(b) Confusion Matrix of Logistic Regression on Reduced Dataset

2. **Multinomial Naive Bayes:** We skipped the Multinomial Naive Bayes classifier for the reduced dataset because this classifier is not suitable for data transformed by SVD, as the result may contain negative values. We start with the original dataset:



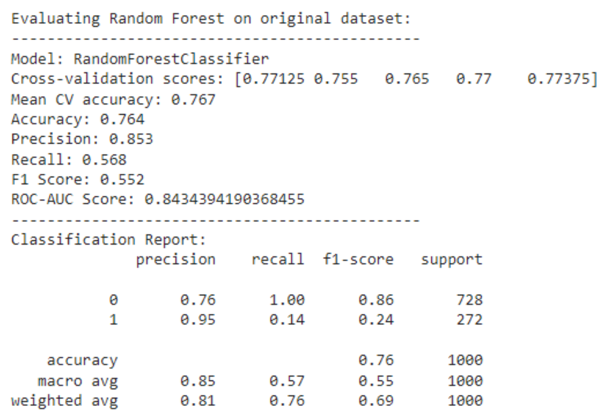
```
Evaluating Multinomial Naive Bayes on original dataset:
----------------------------------------------
Model: MultinomialNB
Cross-validation scores: [0.76125 0.7625  0.755   0.77375 0.7675 ]
Mean CV accuracy: 0.764
Accuracy: 0.763
Precision: 0.852
Recall: 0.567
F1 Score: 0.549
ROC-AUC Score: 0.811479880413704
----------------------------------------------
Classification Report:
              precision    recall  f1-score   support

           0       0.76      1.00      0.86       728
           1       0.95      0.14      0.24       272

    accuracy                           0.76      1000
   macro avg       0.85      0.57      0.55      1000
weighted avg       0.81      0.76      0.69      1000
```

(a) Multinomial Naive Bayes Results on Original Dataset



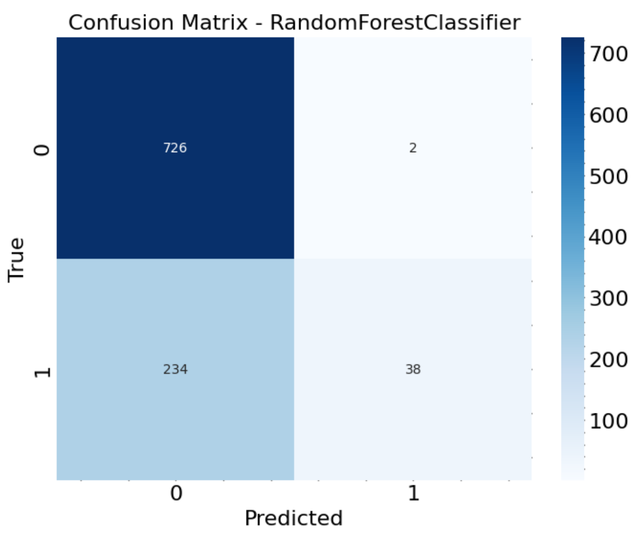
(b) Confusion Matrix of Multinomial Naive Bayes on Original Dataset

3. **Random Forest:** We start with the original dataset:



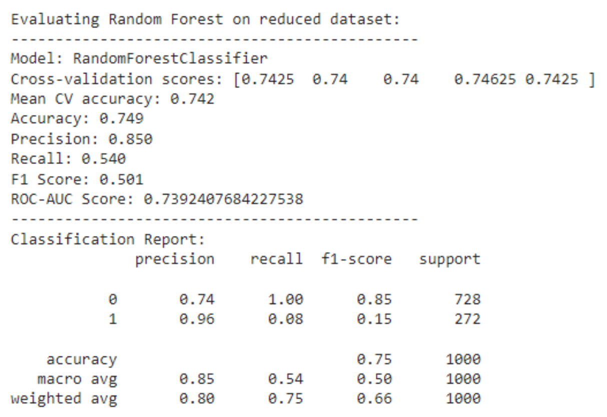
```
Evaluating Random Forest on original dataset:
----------------------------------------------
Model: RandomForestClassifier
Cross-validation scores: [0.77125 0.755   0.765   0.77    0.77375]
Mean CV accuracy: 0.767
Accuracy: 0.764
Precision: 0.853
Recall: 0.568
F1 Score: 0.552
ROC-AUC Score: 0.8434394190368455
----------------------------------------------
Classification Report:
              precision    recall  f1-score   support

           0       0.76      1.00      0.86       728
           1       0.95      0.14      0.24       272

    accuracy                           0.76      1000
   macro avg       0.85      0.57      0.55      1000
weighted avg       0.81      0.76      0.69      1000
```

(a) Random Forest Results on Original Dataset



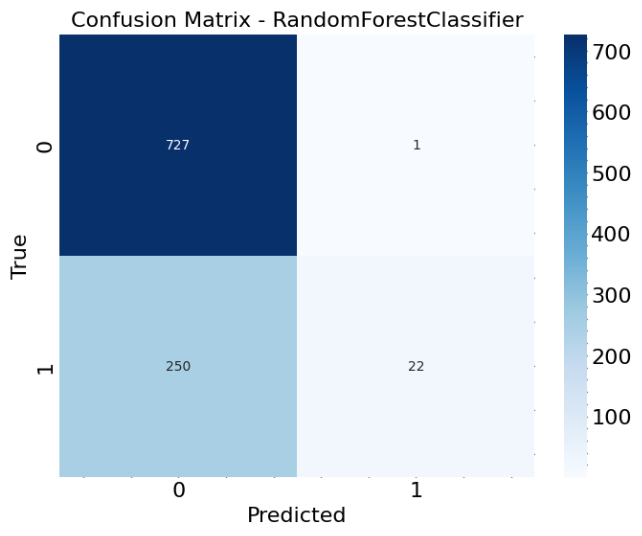
(b) Confusion Matrix of Random Forest on Original Dataset

And then on the reduced dataset:



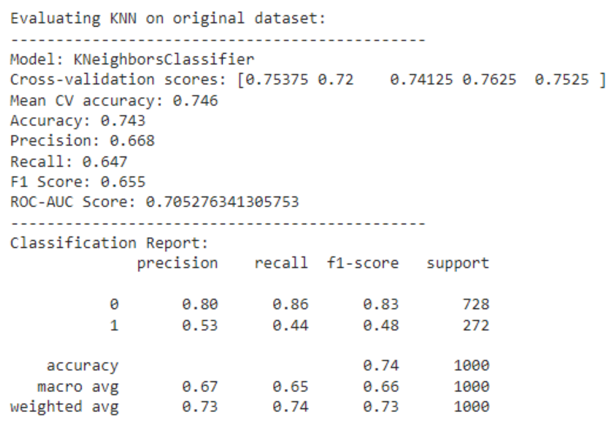
```
Evaluating Random Forest on reduced dataset:
----------------------------------------------
Model: RandomForestClassifier
Cross-validation scores: [0.7425  0.74    0.74    0.74625 0.7425 ]
Mean CV accuracy: 0.742
Accuracy: 0.749
Precision: 0.850
Recall: 0.540
F1 Score: 0.501
ROC-AUC Score: 0.7392407684227538
----------------------------------------------
Classification Report:
              precision    recall  f1-score   support

           0       0.74      1.00      0.85       728
           1       0.96      0.08      0.15       272

    accuracy                           0.75      1000
   macro avg       0.85      0.54      0.50      1000
weighted avg       0.80      0.75      0.66      1000
```

(a) Random Forest Results on Reduced Dataset



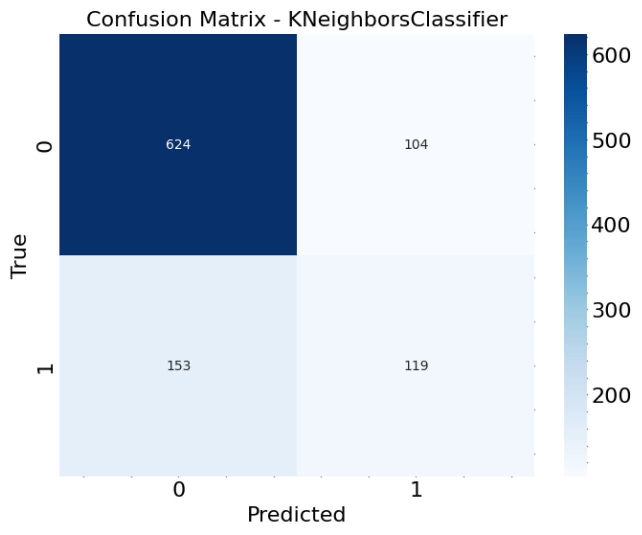
(b) Confusion Matrix of Random Forest on Reduced Dataset

4. **K-Nearest Neighbors:** We ran the classifier on the two datasets separately, starting with the original dataset:



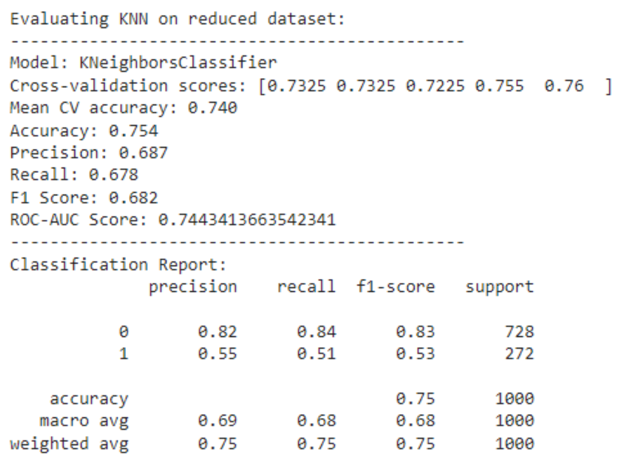
```
Evaluating KNN on original dataset:
----------------------------------------------
Model: KNeighborsClassifier
Cross-validation scores: [0.75375 0.72    0.74125 0.7625  0.7525 ]
Mean CV accuracy: 0.746
Accuracy: 0.743
Precision: 0.668
Recall: 0.647
F1 Score: 0.655
ROC-AUC Score: 0.705276341305753
----------------------------------------------
Classification Report:
              precision    recall  f1-score   support

           0       0.80      0.86      0.83       728
           1       0.53      0.44      0.48       272

    accuracy                           0.74      1000
   macro avg       0.67      0.65      0.66      1000
weighted avg       0.73      0.74      0.73      1000
```

(a) KNN Results on Original Dataset



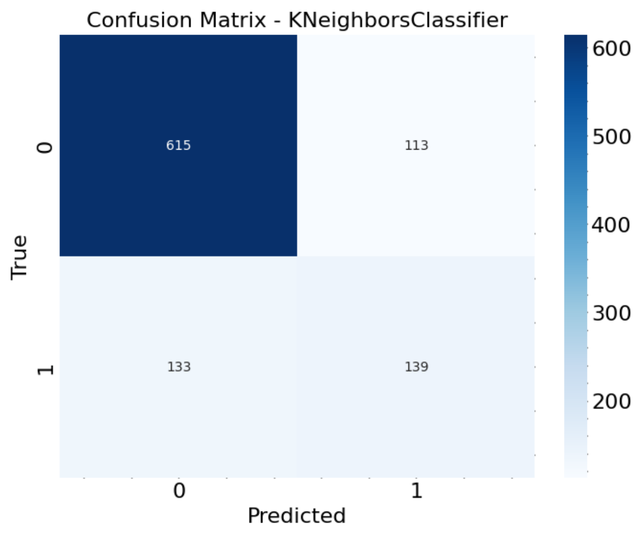
(b) Confusion Matrix of KNN on Original Dataset

And then on the reduced dataset:



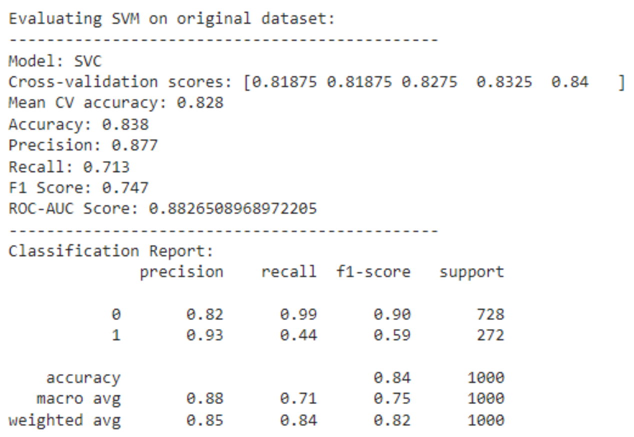
```
Evaluating KNN on reduced dataset:
----------------------------------------------
Model: KNeighborsClassifier
Cross-validation scores: [0.7325 0.7325 0.7225 0.755  0.76  ]
Mean CV accuracy: 0.740
Accuracy: 0.754
Precision: 0.687
Recall: 0.678
F1 Score: 0.682
ROC-AUC Score: 0.7443413663542341
----------------------------------------------
Classification Report:
              precision    recall  f1-score   support

           0       0.82      0.84      0.83       728
           1       0.55      0.51      0.53       272

    accuracy                           0.75      1000
   macro avg       0.69      0.68      0.68      1000
weighted avg       0.75      0.75      0.75      1000
```

(a) KNN Results on Reduced Dataset



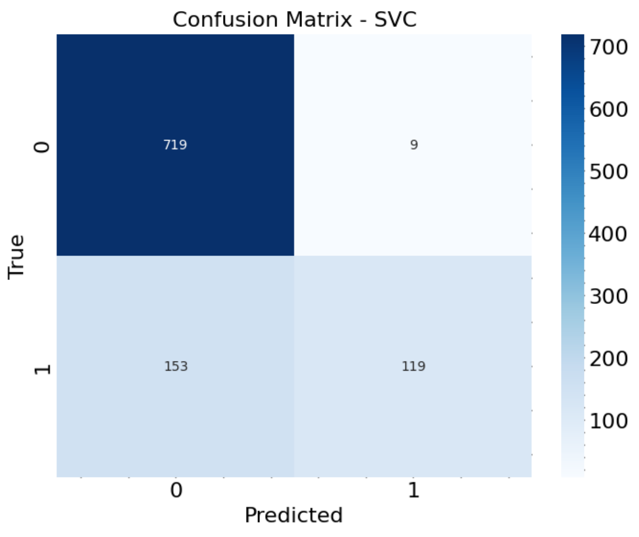
(b) Confusion Matrix of KNN on Reduced Dataset

5. **SVM:** We ran the classifier on the two datasets separately, starting with the original dataset:



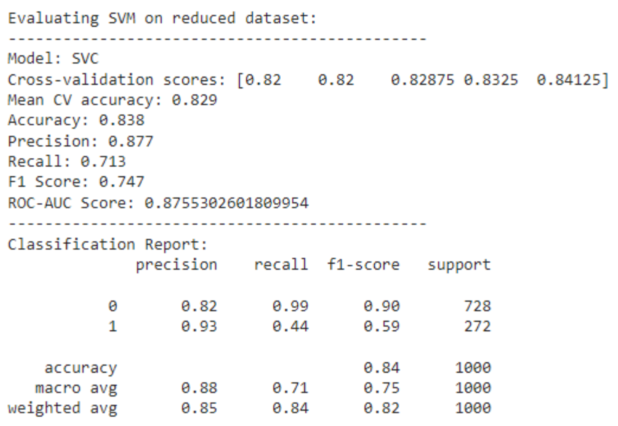
```
Evaluating SVM on original dataset:
----------------------------------------------
Model: SVC
Cross-validation scores: [0.81875 0.81875 0.8275  0.8325  0.84   ]
Mean CV accuracy: 0.828
Accuracy: 0.838
Precision: 0.877
Recall: 0.713
F1 Score: 0.747
ROC-AUC Score: 0.8826508968972205
----------------------------------------------
Classification Report:
              precision    recall  f1-score   support

           0       0.82      0.99      0.90       728
           1       0.93      0.44      0.59       272

    accuracy                           0.84      1000
   macro avg       0.88      0.71      0.75      1000
weighted avg       0.85      0.84      0.82      1000
```

(a) SVM Results on Original Dataset



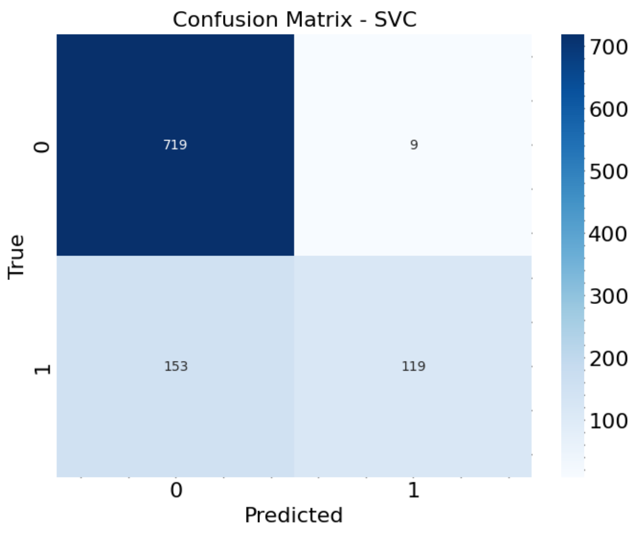
(b) Confusion Matrix of SVM on Original Dataset

And then on the reduced dataset:



```
Evaluating SVM on reduced dataset:
----------------------------------------------
Model: SVC
Cross-validation scores: [0.82    0.82    0.82875 0.8325  0.84125]
Mean CV accuracy: 0.829
Accuracy: 0.838
Precision: 0.877
Recall: 0.713
F1 Score: 0.747
ROC-AUC Score: 0.8755302601809954
----------------------------------------------
Classification Report:
              precision    recall  f1-score   support

           0       0.82      0.99      0.90       728
           1       0.93      0.44      0.59       272

    accuracy                           0.84      1000
   macro avg       0.88      0.71      0.75      1000
weighted avg       0.85      0.84      0.82      1000
```

(a) SVM Results on Reduced Dataset



(b) Confusion Matrix of SVM on Reduced Dataset

2.3 Results

1. **Logistic Regression:**
   (a) **Original Dataset:** It obtained a ROC-AUC of 0.910 and 0.864 of accuracy. The model's performance has a good precision and recall balance.
   (b) **Reduced Dataset (SVD):** The resulting ROC-AUC is 0.910 (Slightly better) and the accuracy is similar to the original dataset. These results shows that Logistic Regression preserves its optimal performance even with fewer dimensions(It justifies that the most important features are preserved in the reduction process by SVD.)
2. **Multinomial Naive Bayes**
   (a) **Original Dataset Only:** It obtained a ROC-AUC of 0.811 with a lower accuracy of 0.763. This model shows a significant precision-recall trade-off, particularly in correspondence of the minority class, i.e. the female class.
   (b) **Reduced Dataset:** Not applicable, because this model is unable to work with negative values which have been generated from SVD transformations.
3. **Random Forest**
   (a) **Original Dataset:** It obtained a ROC-AUC of 0.843 and 0.764 of accuracy. It shows a high precision but low recall for the female class, showing that there might be overfitting to the male class or failing to generalize well for the female class.

   (b) **Reduced Dataset (SVD):** it resulted in a lower ROC-AUC of 0.739 and also reduced accuracy of 0.749. Since the reduction in performance is much higher than the Logistic Regression, we can infer that Random Forest may be more sensitive to the loss of features through the applied dimensionality reduction done by SVD.

4. **K-Nearest Neighbors (KNN)**
   (a) **Original Dataset:** It obtained a lower ROC-AUC value of 0.705 and 0.743 of accuracy. It has a weaker performance with respect to the other models, mainly due to its sensitivity to the high-dimensional feature space and probably due to the existence of noise in the data.
   (b) **Reduced Dataset (SVD):** However for reduced dataset, KNN obtained a ROC-AUC value of 0.744 which is slightly improved and 0.754 of accuracy. As we expected this is possibly because the dimensionality reduction which helps in order to get rid of the curse of dimensionality and moreover removing the noisy data.

5. **SVM**
   (a) **Performance Consistency:** SVM shows an approximately good consistency on its performance across the both datasets, while still a small variation in the ROC-AUC value can be observed. It shows that SVM is robust to the dimensionality reduction and preserve its optimal performance, we will perform hypertuning over the reduced datasets consisting of a smaller amount of features.

6. **Final Conclusions**
   (a) **Dimensionality Reduction Impact:** Logistic Regression seems least affected by the reduction in feature space, maintaining its performance, which indicates its robustness and effectiveness in handling sparse high-dimensional data even when reduced. In contrast, models like Random Forest and KNN show varied performance changes, suggesting they might rely more on the nuances in the full dataset that could be lost during reduction.
   (b) **Model Suitability:** Logistic Regression stands out as particularly well-suited for this dataset, given its performance consistency across different data representations. It could be considered the best model for further tuning and deployment based on these results.
   (c) **Further Analysis:** For models showing a significant drop in performance on the reduced dataset, further investigation into feature importance and selection prior to dimensionality reduction could help in preserving critical information.
   (d) **Indicators of Overfitting or Underfitting:**
      - **Overfitting:** Typically indicated by a high variance in cross-validation scores or a significant difference between training performance and test performance. In this case, the consistent performance across training (cross-validation) and testing along with relatively stable cross-validation scores suggests that overfitting is not a significant concern.
      - **Underfitting:** This occurs when a model is too simple to capture the underlying pattern. SVM's moderate to high performance metrics suggest it is capturing the underlying patterns adequately. However, the low recall could be a sign that the model is not complex enough or not correctly parameterized to capture all relevant signals, especially for the minority class.

2.4 Tuning of Hyperparameters through Cross-validation

Beginning of Hyperparameter Tuning: For the given models in the previous part, we selected these set of parameters to tune the models in order to obtain the best possible results:

```
hyperparameters = {
'Logistic Regression': {'C': [0.001, 0.01, 0.1, 1, 10, 100]},
'Multinomial Naive Bayes': {'alpha': [0.1, 0.5, 1.0, 2.0, 5.0]},
'Decision Tree': {'max_depth': [None, 10, 20, 30, 40, 50]},
'Random Forest': {'n_estimators': [50, 100, 200, 300]},
'KNN': {'n_neighbors': [3, 5, 10]},
'SVM': {'C': [0.1, 1, 10], 'kernel': ['linear', 'rbf'],'probability': [True]}
```

We start with the SVM to see how scaling the dataset will affect the throughput of this algorithm. Since the datasets are sparse matrices and in order to avoid densing the dataset by using StandardScaler, we use MinMaxScaler to preserve the characteristics of the data. As expected, SVM has poor performance with respect to the previous part even with applying data scaling over it. Consequently, our approach of ignoring the applying data scaling algorithm on the sparse datasets seems correct and reasonable.

1. **Logistic Regression**
    - **Best Hyperparameters:** 'C': 1
    - **Accuracy:** 0.864
    - **ROC-AUC Score:** 0.910
    - **Observations:** This model showed a robust performance besides a good balance between precision and recall. The high ROC-AUC value shows strong predictive capabilities. Since the model performed consistently across different metrics, so we can say that there is no signs of overfitting or underfitting.
2. **Multinomial Naive Bayes**
    - **Best Hyperparameters:** 'alpha': 0.1
    - **Accuracy:** 0.863
    - **ROC-AUC Score:** 0.915
    - **Observations:** It obtained a slightly higher ROC-AUC than the Logistic Regression. It also showed a good balance between precision and recall. The alpha value suggests a slight regularization that helps in managing overfitting.
3. **Decision Tree**
    - **Best Hyperparameters:** 'max depth':10
    - **Accuracy:** 0.781
    - **ROC-AUC Score:** 0.684
    - **Observations:** It showed underfitting with a limited max depth, which prevents the model from capturing more complex patterns in the data. The lower ROC-AUC and accuracy scores compared to other models is a good evidence to prove our claim.
4. **Random Forest**
    - **Best Hyperparameters:** 'n estimators': 50
    - **Accuracy:** 0.769
    - **ROC-AUC Score:** 0.840
    - **Observations:** Although the ROC-AUC score is acceptable, the poor recall in the female class showing that there might be overfitting, meaning that while the model works well in the male class, it is not able to generalize well in the female class.
5. **KNN**
    - **Best Hyperparameters:** 'n neighbors': 10
    - **Accuracy:** 0.774
    - **ROC-AUC Score:** 0.767
    - **Observations:** While KNN performed regularly, its recall value was low and it shows that it had trouble to manage the variations in male and female classes and identify more complex relationships in the data.
6. **SVM (on reduced dataset)**
    - **Best Hyperparameters:** 'C': 1, 'kernel': 'linear', 'probability': True
    - **Accuracy:** 0.85
    - **ROC-AUC Score:** 0.890
    - **Observations:** SVM totally showed a good performance with balanced precision and recall. The linear kernel worked well and suggesting that there may be some linear separability in the data. Moreover as we expected tuning its parameters on the reduced dataset did not considerably decrease its usefulness.
7. **Final Conclusions**
    (a) Hyperparameter tuning enhanced the model performances, specially for Logistic Regression and Naive Bayes, which showed high ROC-AUC scores.
    (b) Overfitting was mainly a concern with the Random Forest model. Underfitting was observed with the Decision Tree due to the depth limitations.
    (c) For the female class, most models need some extra works to improve the recall while they do not sacrificing the precision.
    (d) Feature engineering and also combining the model predictions (ensemble methods) could result in better overall performance and robustness.(The next part of this section is related to this task)

2.5 Exploration of Different Features

at this part, we checked the effectiveness of different feature engineering strategies, particularly focusing on how textual features are transformed and combined with subreddit information.

1. **Combining TF-IDF with Subreddit Features**
   - **Current Approach:** We have already combined subreddit features (extracted at section 1-3 using one-hat encoding) with text features transformed via TF-IDF(again at section 1-3). This combination is what we were using in all of our models up to now.
   - **Possible Experiment:** We can consider adjusting the effect of text features versus subreddit features by scaling the TF-IDF features before combining them with subreddit features. We implemented it by using MaxAbsScaler which is suitable for handling sparse data.
2. **Evaluate the Impact of TF-IDF**
   - **Current Approach:** We were transforming raw counts into TF-IDF, as it reduces the impact of frequently occurring words which might be less informative.
   - **Possible Experiment:** We can evaluate the performance impact of using raw count vectors versus TF-IDF. Then at one version of our experiment we use directly CountVectorizer without transforming to the TF-IDF, and on another version we only use TfidfTransformer.
3. **Data Standardization**
   - **Possible Experiment:** MaxAbsScaler can be used for standardization. So, we scale each feature by its maximum absolute value without shifting/centering data in order to preserve the zero entries(despite the common approach at standardscaler). Then we can compare the effect of Standardization on the baseline model.
4. **Using more Complex Methods:**
   Up to now, we mainly tried to use all of the available feature sets that we had already obtained during the first section. However, it is also possible to use a class rebalancing method used in machine learning to increase the number of samples in a dataset's minority class, called SMOTE. Although we did not follow the SMOTE method, we still obtained an increase in our performance by simply using our own approach. SO, we just mentioned that we can also use SMOTE as an alternative. of samples in a dataset's minority class

For these different feature sets, we test the effect of each of them on our best model which obtained through the hyper tuning part, i.e. LogisticRegression(C=1). So, let's analyze the results, both when stacking (combining) features and when evaluating them individually.

1. **Combined Feature Sets (Using hstack)**
   - **ROC-AUC with Counts:** 0.856
   - **ROC-AUC with TF-IDF:** 0.910
   - **ROC-AUC with TF-IDF Scaled:** 0.908

   Observations:
   - **TF-IDF outperforms Counts:** The TF-IDF transformation, which weights terms based on their frequency, provides a more discriminative feature set compared to raw counts. This is evident from the higher ROC-AUC score and suggests that TF-IDF helps in emphasizing words that are more relevant for classification (by reducing the weight of common words across documents).
   - **Scaling has minimal impact:**when applying MaxAbsScaler to the TF-IDF features, the slight decrease in ROC-AUC score shows that scaling does not significantly changes the performance for this model and dataset. Therefore we can say that the Logistic Regression model doesn't benefit much from this additional scaling step.
2. **Individual Feature Sets**
   - **ROC-AUC with Subreddits:** 0.895
   - **ROC-AUC with Body Counts:** 0.818
   - **ROC-AUC with Body TF-IDF:** 0.864
   - **ROC-AUC with Body TF-IDF Scaled:** 0.855

Observations:

- **Subreddit features are highly predictive:** The subreddit features alone provide a very high ROC-AUC, almost equal to the performance of combined TF-IDF features. This shows that the subreddit context itself is a strong predictor of the target variable, likely capturing significant behavioral patterns.
- **Text features alone are less effective:** When evaluated without the context provided by subreddit features, text features (both counts and TF-IDF) perform notably worse. This underscores the importance of context when combining text data with categorical data like subreddits.

3. **Inferences**

- **Features Integration:** The improvement in ROC-AUC when combining features (especially in the TF-IDF case) shows that the text features and subreddit features complement each other. The features based on subreddits capture one aspect of the data. Additional textual features bring another layer of information into the mix, thus enhancing the model's capability.
- **Importance of Contextual Information:** The strong performance of subreddit features alone might indicate that users involvement in some specific subreddits are highly indicative of the behaviors or attributes being modeled.
- **Utility of TF-IDF:** The TF-IDF transformation is useful in improving the importance of text input for the machine learning models, due to its ability to reduce noise which caused by common but less informative words.

3 Section 3 – Unsupervised Learning – Clustering

In this unsupervised learning task, the objective is to cluster Reddit comments based on the words used within them. The aim is to identify patterns and group comments with similar words together. The dataset '*data_unsupervised.csv*' is the main dataset used in this section. Before proceeding with the tasks of this section, we performed a brief EDA over this dataset:

3.1 Data Preprocessing for Clustering

1. Loading the dataset. We also performed some extra EDA to extract some useful information from the dataset, and the results are clear and concrete.

Feature	Number of Unique Values	Number of Missed Values
Author	15000	0
Subreddit	3970	0
Created_utc	715561	0
Body	828126	2

Table 3: Number of unique values and missed values for each feature

2. Performing data cleaning and text standardization on the body column, the same as we did in Section 1 using the nltk and contractions libraries.

3. Executing text vectorization using Bag of Words (BoW) and TF-IDF.
    - We have transformed the text data into a numerical format that is suitable for clustering algorithms by converting the corpus into a TF-IDF weighted sparse matrix. This matrix encodes the importance of words in the comments relative to the entire dataset, which is ideal for detecting patterns and grouping similar comments.
    - **The resulting sparse matrix:**
        - Has dimensions indicating that there are 1,107,946 comments.
        - Contains 179,353 unique words after removing stop words and applying max document frequency filtering.
        - Has about 12,166,273 non-zero elements, which suggests the data is quite sparse as expected.

Once all of these steps completed, then we can trust that the preprocessing part has prepared our dataset for unsupervised learning tasks. We decided to use K-means and MiniBatch K-means algorithms in this section.

3.2 K-Means

3.2.1 Determining the Number of Clusters

In order to find the optimal number of clusters for K-means, we used the elbow method. The elbow point seems to occur at k=4, suggesting that increasing the number of clusters beyond 4 will result in a diminishing returns in terms of within-cluster variance reduction. From now we proceed with k=4 obtained from the results of elbow method for K-means, however k=3 might be also a good choice.

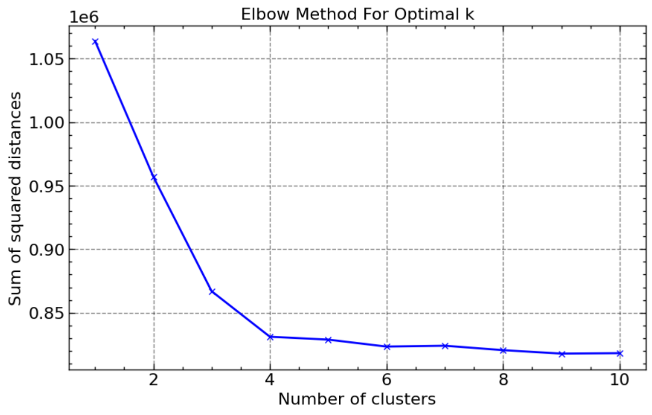
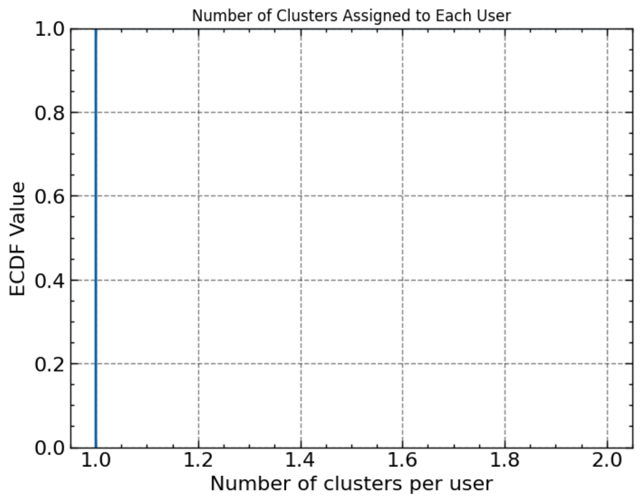


Figure 14: Elbow curve for K-Means algorithm

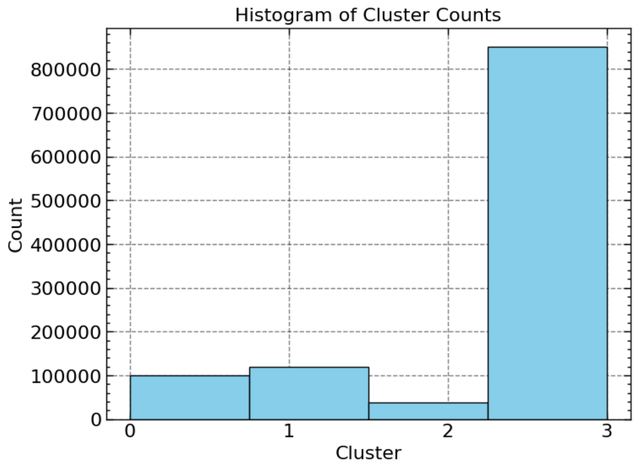
3.2.2 Analysis of the Distribution of Users

Cluster Assignments to Users: According to the ECDF plot and cluster count histogram, most users are grouped in one cluster,i.e. Cluster 3, . This may be because Cluster 3 identifies a pattern of thought or language that appears frequently in a wide variety of remarks. Nevertheless, considering their lesser proportions, the other three groups indicate more specific or different kinds of remarks. First we show the numerical outputs and then visualize it through the ECDF curve:

```
clusters_per_author.value_counts()
cluster
1    14995
2        5
Name: count, dtype: int64
```



(a) Number of Clusters Assigned to Each User at K-means



(b) Histogram of Cluster Counts at K-means

3.2.3 Main Components of the PCA Visualization

After the dimensionality of the TF-IDF features is reduced, the PCA scatter plot displays the distribution of the comments among the first two principle components. The separability of clusters in a 2d space can be observed from this image. It seems that most data points are concentrated in a particular area of the graph. For example, Cluster that 3 that has been shown in red, is highly dense and appears to be tightly packed near the origin and that comments within this cluster might have similar or closely related textual content. However, Clusters 1, 2, and 0 show more spread on the plot, meanwhile they remain somewhat close to the origin. From this spread in the plot, we can notify that there is a bit more diversity within these clusters compared to Cluster 3.
The aforementioned 2d plot, has been obtained without applying any standardization algorithm to our data-set(since it is sparse). (we also try to Scale the data as a possible solution for better and more distinguishable representation, but the output plot did not change very significantly.)

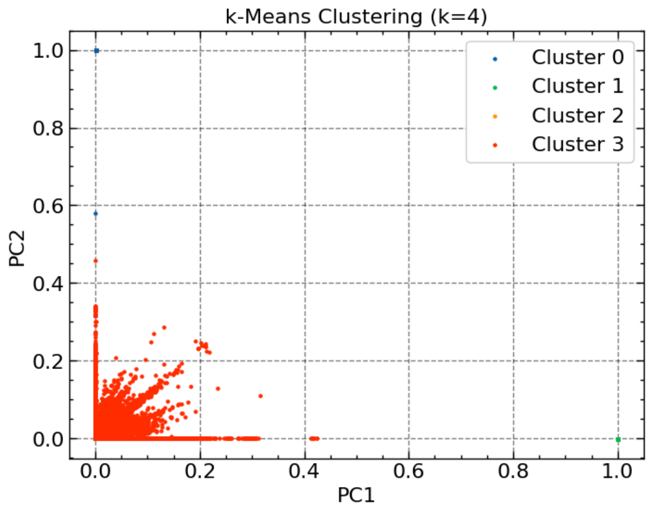


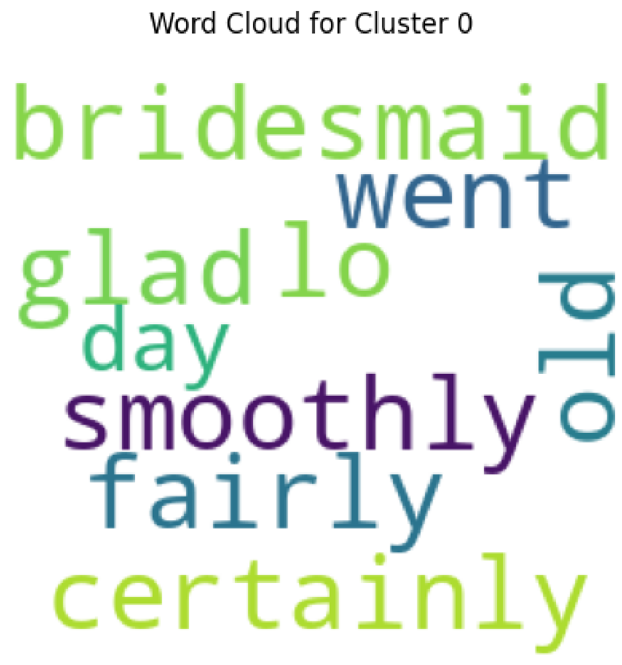
Figure 16: K-Means Clustering (k=4) with 2D representation

3.2.4 Cluster Analysis

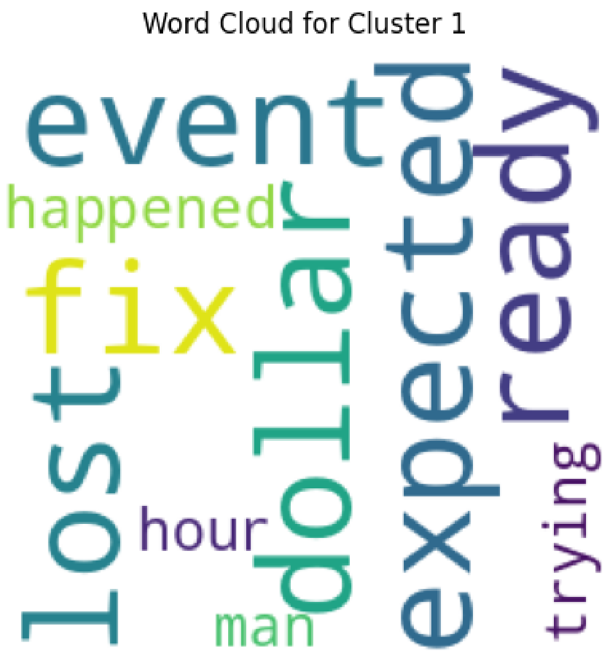
Cluster Characterization: First by obtaining the list of top 10 words in each cluster, we obtain a knowledge of the most significant words within each cluster by their corresponding weights. We notified that the presence of words like "bridesmaid", "smoothly", "dollar", and "expected" show a thematic distinctions between the clusters. Yet we found that there are also nonsensical entries like "дщ" and "gazan", which might be noise, outliers, or artifacts from the clustering process.

```
Cluster 0 top words:
bridesmaid: 34010.4871558859
smoothly: 34002.50480821532
lo: 34001.52137038842
fairly: 33744.85380010924
certainly: 33644.61881292434
glad: 33621.78058216417
went: 33019.77446112449
old: 32849.79462805808
day: 31715.39211050673
gazan: 0.0

Cluster 1 top words:
expected: 36360.330890799254
dollar: 36329.8888618669
fix: 36280.3271394992
ready: 36268.87769762303
event: 36265.90088258924
lost: 36073.34015949899
happened: 35946.67594931406
hour: 35723.03288390035
trying: 35236.43862535388
man: 35055.67558074195

Cluster 2 top words:
yank: 37241.89837444261
going: 0.4392304188703921
дщ: 0.0
gazebo: 0.0
gazidis: 0.0
gazi: 0.0
gazette: 0.0
gazer: 0.0
gazelle: 0.0
gazela: 0.0

Cluster 3 top words:
like: 15655.899010599367
http: 14338.546410793386
think: 9964.558472677134
know: 9725.258743926175
people: 9638.029228747579
good: 8805.10133598556
really: 8740.163381975533
time: 8578.805713020021
make: 7694.824549597467
thing: 7528.047985118332
```

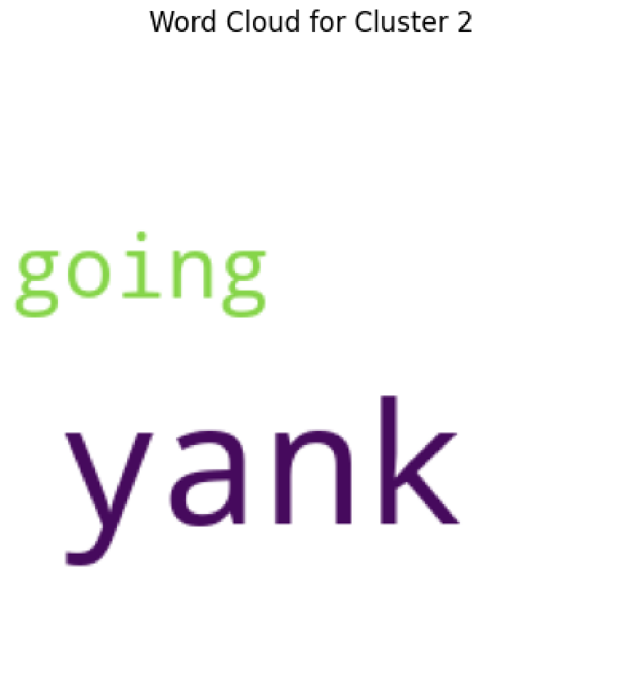
Word Cloud Visualization: Here we tried to visualize the 10 top words within each cluster that obtained through the previous part. In order to do that we used the word clouds library which visualize the frequency and importance of the words in each cluster and provides a more intuitive understanding of the topics or themes inside each cluster.



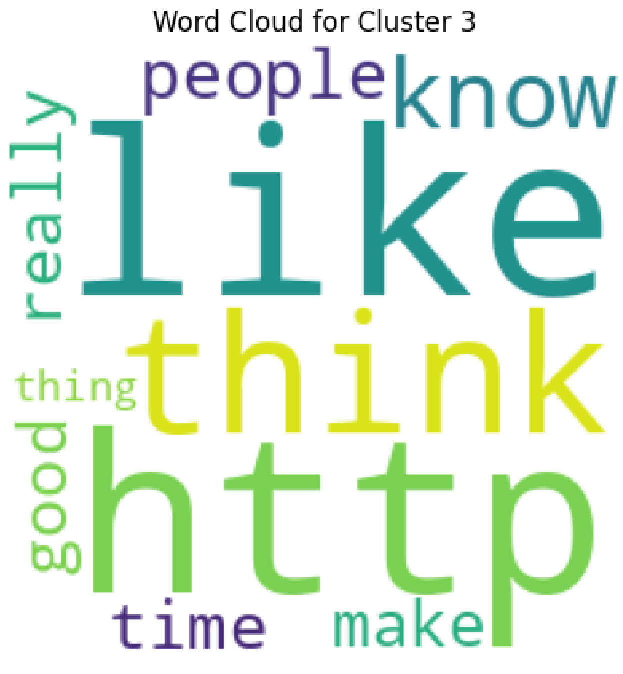
(a) Word Cloud for Cluster 0 at K-means



(b) Word Cloud for Cluster 1 at K-means



(c) Word Cloud for Cluster 2 at K-means



(d) Word Cloud for Cluster 3 at K-means

Inferences:

1. **Thematic Trends:** Specific topic patterns seem to be reflected in clusters, with Cluster 0 having words related to perhaps personal experiences or events, Cluster 1 having words that may relate to financial discussions, and Cluster 3 containing more general discussion words like "like," "think," and "know."
2. **Cluster Purity:** The mixed nature of the top words, especially in Clusters 2 and 3, may suggest that the clusters are not entirely pure or that the method for identifying top words needs refinement.

For a more sophisticated view, we can consider looking at a wider range of words instead of just looking at the top 10 words within the clusters.

3.2.5 Distribution of Subreddits in the Clusters

Finally, from the heatmap and the crosstabulation output provided for the following part, here are our observations for the subreddit distribution across the clusters:

1. **Dominant Clusters:** The heatmap implies that there are clusters that have a significant concentration of

comments from specific subreddits. For example, "AskReddit" seems to dominate one cluster and shows that a general discussion cluster where a variety of topics are covered.

```
    subreddit
AskReddit         135811
projectcar        118677
beyondthebump     100626
AskMen             26088
AskWomen           22691
                    ...
HiTMAN                 1
CompetitiveEDH         1
lifepluswomen          1
beercirclejerk         1
acecombat              1
Name: count, Length: 3970, dtype: int64
```

2. **Cluster-Subreddit Alignment:** Certain subreddits are exclusively found in particular clusters. This is an indication that the clustering algorithm is effectively grouping comments based on subreddit-specific language or topics.

```
cluster       0  1  2    3

subreddit
1022          0  0  0   24
1200isplenty  0  0  0   25
1911          0  0  0   17
195           0  0  0   21
2007scape     0  0  0  102
...          .. .. ..  ...
yugioh        0  0  0  142
zelda         0  0  0  195
zen           0  0  0   26
zombies       0  0  0   27
zyzz          0  0  0   20

[3970 rows x 4 columns]
```

3. **Sparse Clusters:** Some clusters have a very sparse contribution across different subreddits. It represents that these clusters are picking up on less common language usage or more unique discussion threads.

4. **Top Subreddit Distribution:** Based on comment counts, top subreddits show a skewed distribution across the clusters. This reflect the varying popularity and activity levels of different subreddits in such a way that more active ones contributing to more comments on the clusters.

5. **Inter-cluster Subreddit Distribution:** As it can be seen from the plots, some subreddits like "AskReddit" seem to be isolated to a single cluster,while others are spread across multiple clusters. So it can be infer that the language or topics in these subreddits are more varied and do not correspond to a single thematic cluster.

6. **Bias and Overlap:** The clustering method and the subsequent assignment of subreddits to clusters can be influenced by the overall size and activity of the subreddit.In other word, Larger subreddits might overwhelm smaller ones, and similarly themed subreddits might end up in the same cluster due to shared vocabulary, even if their topics are different.

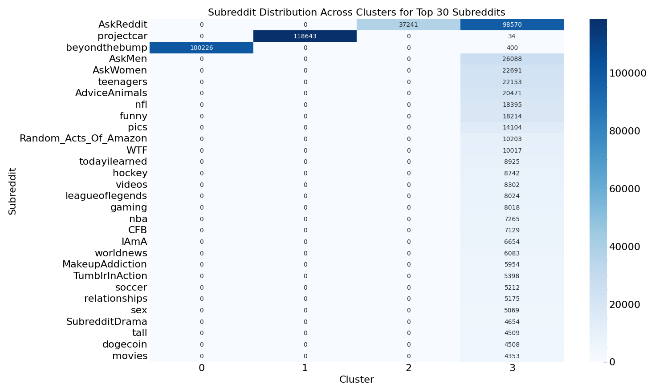


Figure 18: Subreddit Distribution Across Clusters for Top 30 Subreddits at K-means

3.3 MiniBatch K-means

3.3.1 Determining the Number of Clusters

MiniBatch K-means seems to have a different distribution in the clusters compared to the standard K-means. While K-means has a very large Cluster3, MiniBatch K-means appears to have a more balanced distribution among clusters 0, 1, and 2.

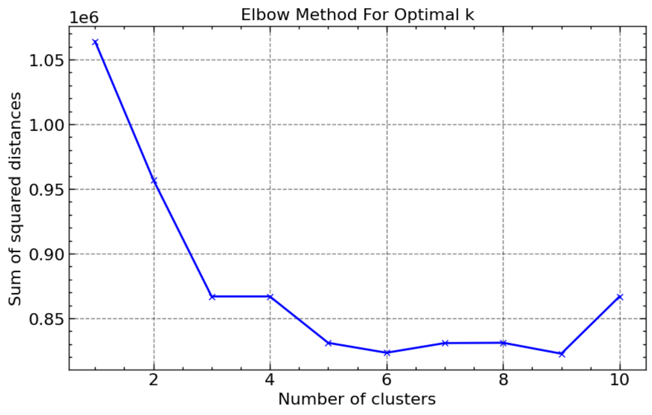
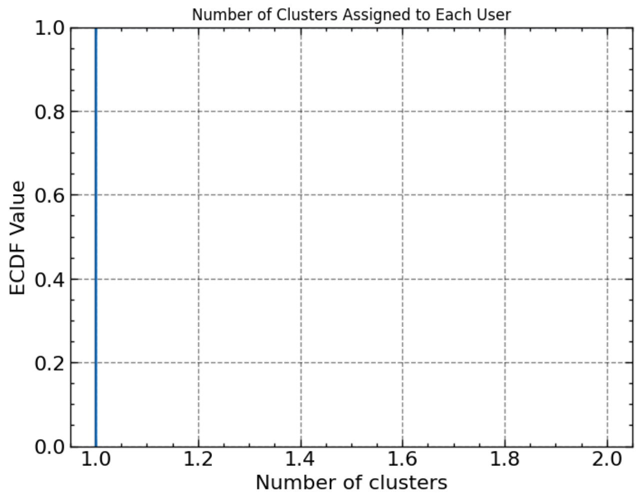


Figure 19: Elbow curve for MiniBatch K-means algorithm

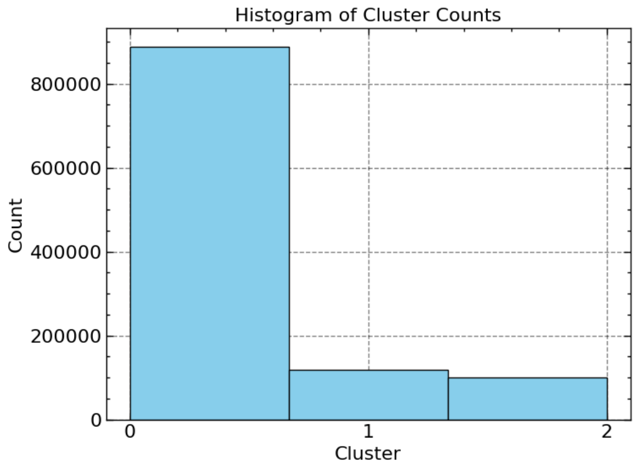
3.3.2 Analysis of the Distribution of Users

According to the ECDF plot and cluster count histogram, most users are grouped in one cluster,i.e. Cluster 0, . This may be because Cluster 0 identifies a pattern of thought or language that appears frequently in a wide variety of remarks. Nevertheless, the other three two clusters indicate more specific or different kinds of remarks. First we show the numerical outputs and then visualize it through the ECDF curve:

```
clusters_per_author.value_counts()
cluster
1    14997
2        3
Name: count, dtype: int64
```



(a) Number of Clusters Assigned to Each User at MiniBatch K-means



(b) Histogram of Cluster Counts at MiniBatch K-means

3.3.3 Main Components of the PCA Visualization

Again as for K-means, here we have also the same justification for our output 2d plot. It seems that most data points are concentrated in a particular area of the graph. For example, Cluster 0 that has been shown in blue, is highly dense and appears to be tightly packed near the origin and that comments within this cluster might have similar or closely related textual content. However, the other two Clusters, cluster 1 and 2, show more spread on the plot, while they remain somewhat close to the origin. From this spread in the plot, we can notify that there is a bit more diversity within these clusters compared to Cluster 3.
The aforementioned 2d plot, has been obtained without applying any standardization algorithm to our data-set(since it is sparse). (we also try to Scale the data as a possible solution for better and more distinguishable representation, but the output plot did not change very significantly.)

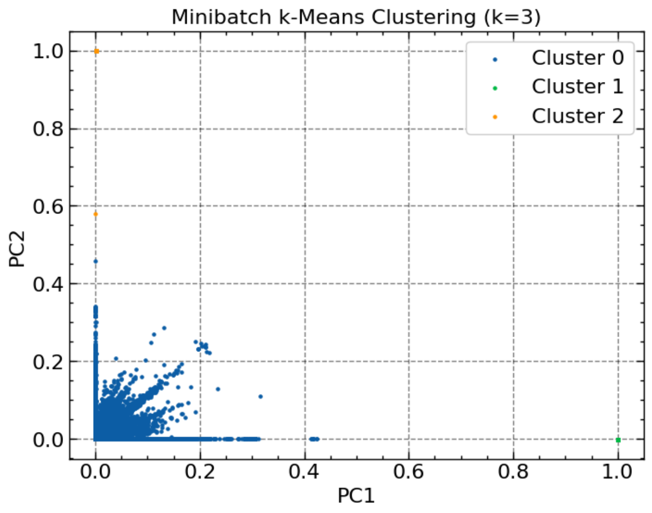


Figure 21: MiniBatch K-means Clustering (k=3) with 2D representation

3.3.4 Cluster Analysis

First by obtaining the list of top 10 words in each cluster, we obtain a knowledge of the most significant words within each cluster by their corresponding weights. Then,the word clouds and top words per cluster give insights into the thematic contents of each cluster. From the results, we have that:

- Cluster 0 for MiniBatch K-means has words associated with general discussion ('like', 'think', 'know'), which are indicative of clusters which capture conversational or common discourse.

```
Cluster 0 top words:
yank: 37255.79267283046
like: 15655.899010599367
http: 14338.546410793386
```

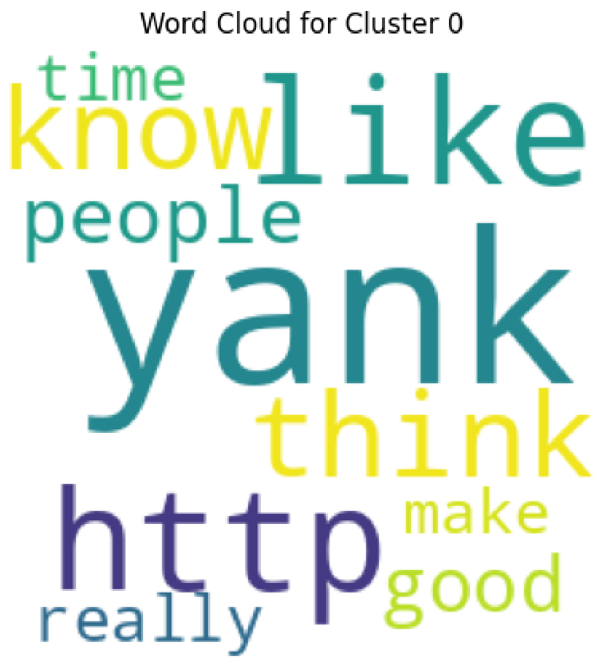
```
think: 9964.558472677134
know: 9725.258743926175
people: 9638.029228747579
good: 8805.10133598556
really: 8740.163381975533
time: 8578.805713020021
make: 7694.824549597467
```

- Cluster 1 for MiniBatch K-means has words like 'expected', 'dollar', 'fix', which are extracted from discussions on finance, expectations, or problem-solving.

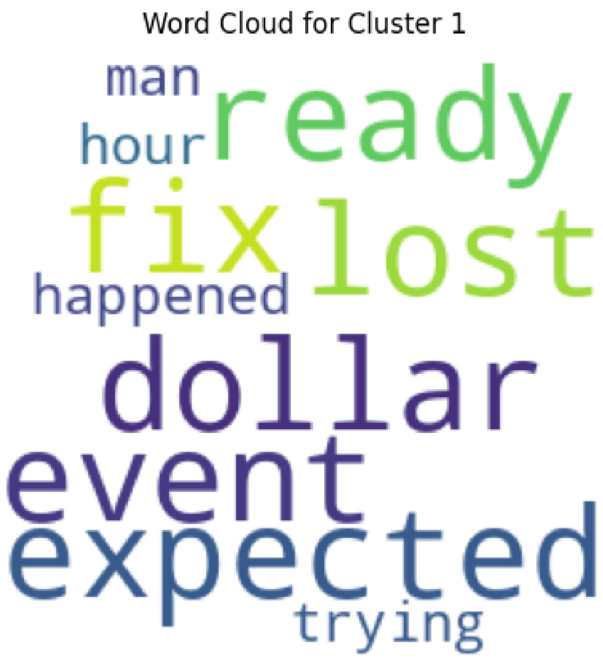
  ```
  Cluster 1 top words:
  expected: 36360.330890799254
  dollar: 36329.8888618669
  fix: 36280.3271394992
  ready: 36268.87769762303
  event: 36265.90088258924
  lost: 36073.34015949899
  happened: 35946.67594931406
  hour: 35723.03288390035
  trying: 35236.43862535388
  man: 35055.67558074195
  ```

- Cluster 2 for MiniBatch K-means contains words like 'bridesmaid', 'smoothly', 'fairly' focuses on discussions about events such as wedding, party or any other related activities.

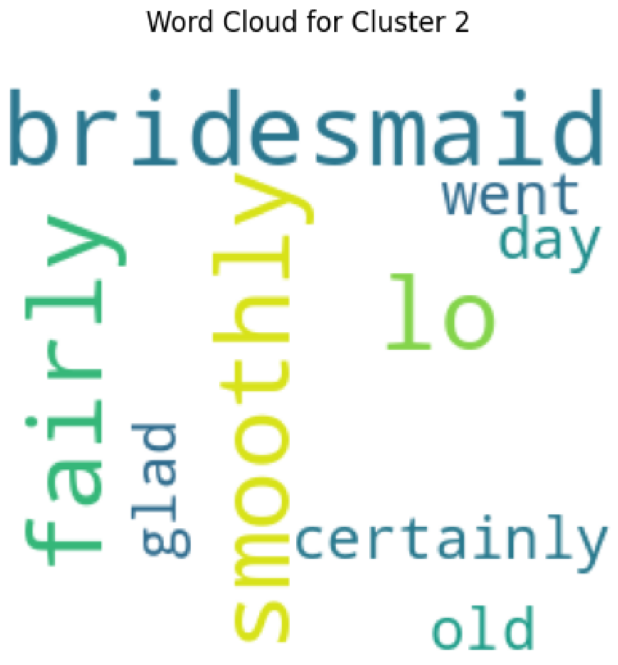
  ```
  Cluster 2 top words:
  bridesmaid: 34010.4871558859
  smoothly: 34002.50480821532
  lo: 34001.52137038842
  fairly: 33744.85380010924
  certainly: 33644.61881292434
  glad: 33621.78058216417
  went: 33019.77446112449
  old: 32849.79462805808
  day: 31715.39211050673
  gazan: 0.0
  ```



(a) Word Cloud for Cluster 0 at MiniBatch K-means



(b) Word Cloud for Cluster 1 at MiniBatch K-means



(c) Word Cloud for Cluster 2 at MiniBatch K-means

3.3.5 Distribution of Subreddits in the Clusters

Finally, from the heatmap and the crosstabulation output provided for the following part, here are our observations for the subreddit distribution across the clusters:

1. **Dominant Clusters:** The heatmap implies that there are clusters that have a significant concentration of comments from specific subreddits. For example, "AskReddit" seems to dominate one cluster and shows that a general discussion cluster where a variety of topics are covered.

```
        subreddit
AskReddit          135811
projectcar         118677
beyondthebump      100626
AskMen              26088
AskWomen            22691
                     ...
HiTMAN                  1
CompetitiveEDH          1
```

```
lifepluswomen          1
beercirclejerk         1
acecombat              1
Name: count, Length: 3970, dtype: int64
```

2. **Cluster-Subreddit Alignment:** Certain subreddits are exclusively found in particular clusters. This is an indication that the clustering algorithm is effectively grouping comments based on subreddit-specific language or topics.

```
cluster          0  1  2

subreddit
1022            24  0  0
1200isplenty    25  0  0
1911            17  0  0
195             21  0  0
2007scape      102  0  0
...            ... .. ..
yugioh         142  0  0
zelda          195  0  0
zen             26  0  0
zombies         27  0  0
zyzz            20  0  0

[3970 rows x 3 columns]
```

The heatmap visualization provide a sight at how subreddits are distributed among the clusters. We mainly define a pure cluster as a cluster dominated by a single subreddit, where discussions from that subreddit are distinct enough to be grouped separately. In contrast, a more spread-out distribution shows that the clustering isn't aligning as clearly with subreddit divisions.

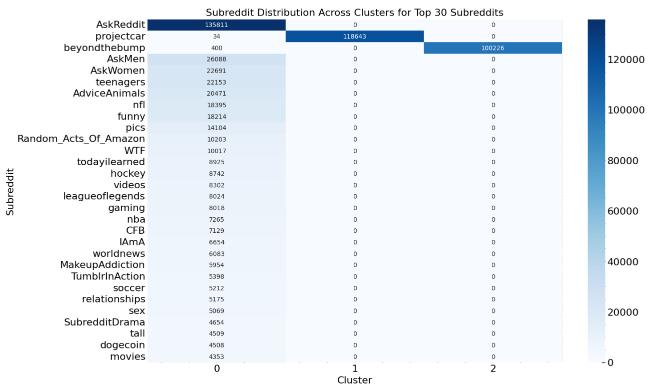


Figure 23: Subreddit Distribution Across Clusters for Top 30 Subreddits at MiniBatch K-means

3.4 Comparing K-means to MiniBatch K-means

- MiniBatch K-means tends to be faster and more scalable to large datasets due to the incremental update to cluster centers.
- The resulting clusters from MiniBatch K-means can sometimes be less consistent due to the randomness introduced by the mini-batches. So if the dataset has a complex or subtle cluster structure, this will be a disadvantage.
- As seen in our results, the final cluster quality from MiniBatch K-means slightly differ from the standard K-means. Differences in clusters size and and their composition can be appeared due to the different approaches for convergence that is used by these algorithms.

In other words, Both algorithms have their use cases and can be selected based on the specific needs of the dataset and computational resources. If we're dealing with very large datasets and need a faster and more scalable solution, MiniBatch K-means might be preferable. However, if we're looking for more precise cluster assignments and have the enough computational resources to handle this task, standard K-means might be the better choice.

Final Comment: At the end of Section 3 of our Jupyter Notebook, there is a subsection called "REST (unsuccessful trials over other unsupervised algorithms)". During this section, we tried to test other unsupervised clustering algorithms such as DBSCAN and GMM, but we were confronted with some challenges. These issues mainly came from the fact that, since the dimension of our feature matrix was enormous, it was too costly from a computational resource point of view to implement any algorithms. So we tried to take a subsample of the data to find the optimal configuration of the clustering algorithms (fine tuning of the hyperparameters), but the resulting silhouette score for the optimal cases was also unsatisfactory. We tried with different percentages of samples, but the best silhoute score that we obtained was around 0.4 for 5% of the main dataset, which was again not satisfying. So, we decided not to perform subsampling, and despite the supervised task, we examined too many classifiers,only limited our options for clustering algorithms to K-means and minibatch K-means, as the project had asked from us (just 2 algorithms).

4 Section 4 – Language Models exploration

4.1 Language Model Preparation

We chose BERT to encode the comments. For this purpose, we used the Hugging Face library to load the pretrained BERT tokenizer and BERT base model. We opted for DistilBERT, a smaller and faster version of BERT that maintains approximately 97% of BERT's performance.

To encode the comments:

1. Firstly, we tokenized the text using the BERT tokenizer. We added BERT's special tokens to the text (`[CLS]` at the beginning and `[SEP]` at the end of each input text).
2. We set a maximum input length of 512 tokens. Texts longer than 512 tokens were truncated, and padding was applied to texts shorter than 512 tokens.
3. Next, we created a corresponding vector of token IDs, where each cell corresponds to the index of that token in BERT's vocabulary.
4. The token IDs vector was then fed into the BERT model.
5. The model returned the results of all its hidden layers as well as an attention mask. The attention mask indicates which tokens are in the input text and which correspond to the `PAD` token.
6. To compute the text embedding, we took the last hidden layer and computed the mean pooling. It's important to note that the attention mask vector was used to skip the `PAD` tokens during mean pooling. It's worth mentioning that we also considered using the embedding of the first token (`[CLS]`), but found that mean pooling provided better performance.
7. Finally, an embedding of size 768 was created for each data row.

4.2 Fraction of times posted in subreddits

1. We started by creating a matrix with a shape of `len(unique_authors)` = 5000 and `len(unique_subreddits)` = 3468. For each (user, subreddit) pair, we counted the number of comments the user posted in that subreddit.
2. Next, we normalized the matrix by dividing each row by the total number of comments posted by that user. This ensured that each entry in the matrix represented the fraction of comments the user made in each subreddit.
3. To reduce the cardinality of the subreddit matrix, we employed Singular Value Decomposition (SVD). We aimed to retain 95% of the variance in the data. Upon analysis, we found that the dimension with a cumulative variance ratio greater than 0.95 was 806. Therefore, we used SVD to reduce the cardinality to 806 dimensions.

4.3 Concatenation

1. Tokenizing and Embedding Comments

To embed user comments, we use the `embed_text` function, which leverages the DistilBert model. The process involves:

- Tokenizing each comment using the DistilBert tokenizer.
- Passing the tokenized text through the DistilBert model.
- Applying mean pooling to the token embeddings to obtain a single vector representation for each comment.

2. Handling Memory Limitations

Due to memory constraints, comments are processed in chunks of 1000. For each chunk:

- The embeddings are computed and stored as compressed files.
- These embeddings are then appended to a list for further processing.

3. Concatenating Embeddings

Once all chunks are processed, the saved embeddings are:

- Loaded from the compressed files.
- Concatenated into a single array, referred to as `embeddings_all`.

4. Combining with Subreddit Matrix

To create the final feature matrix, `input_matrix`:

- The reduced subreddit matrix (obtained from step 2) is concatenated with the DistilBert embeddings.
- This concatenation is done along the columns, integrating both textual features from the comments and structural features from subreddit activity.

4.4 Dense Layer Addition

For the classification task, we experimented with a Multilayer Perceptron (MLP) that included one or two Dense layers. The architecture culminates in a final Dense layer of size 1 with a sigmoid activation function. This design choice is crucial for several reasons:

1. **Binary Classification Output:**
   - The final Dense layer has a size of 1 because we are performing binary classification. Each output value corresponds to the probability that a given input belongs to the positive class.
   - The sigmoid activation function is used because it maps the output to a range between 0 and 1, effectively representing a probability.
2. **Sigmoid Activation Function:**
   - The sigmoid function is defined as $\sigma(x) = \frac{1}{1+e^{-x}}$, which squashes the input values to the (0, 1) interval.
   - This makes it suitable for binary classification tasks, where the output can be interpreted as a probability. A threshold (typically 0.5) is then applied to classify the input into one of the two classes.
3. **Regularization and Dropout:**
   - In addition to the final Dense layer, the model includes regularization techniques such as L2 regularization and Dropout layers. These techniques help prevent overfitting by penalizing large weights and randomly setting a fraction of input units to zero during training, respectively.
4. **Model Architecture and Training:**
   - The model is defined using the `Sequential` API from Keras, which allows for easy stacking of layers. The architecture includes:

```
model = Sequential([
    Dense(128, activation='relu', input_shape=(input_matrix.shape[1],)),
    Dropout(0.5),
    Dense(64, activation='relu', kernel_regularizer=regularizers.l2(0.01)),
    Dropout(0.3),
    Dense(1, activation='sigmoid', bias_initializer=keras.initializers.Constant(initia
])
```

   - The input layer is a Dense layer with 128 units and ReLU activation.
   - Dropout layers with dropout rates of 0.5 and 0.3 follow each Dense layer to mitigate overfitting.
   - A Dense layer with 64 units and L2 regularization follows, providing an additional layer of feature extraction.
   - The final Dense layer with 1 unit and sigmoid activation produces the probability output for the binary classification.
5. **Training Process:**

- The model is compiled with the Adam optimizer, binary cross-entropy loss function, and AUC as a metric:

```
model.compile(optimizer=Adam(5e-5), loss='binary_crossentropy', metrics=[keras.metrics
```

- During training, class weights are computed and used to handle class imbalance, ensuring the model does not become biased towards the majority class.
- Early stopping is employed to monitor the validation AUC and stop training if performance does not improve for a set number of epochs, restoring the best weights observed during training.

4.5 Fine-tuning the Last Layer of the Network

Fine-tune the last layer of the network on the supervised training set for N epochs : To fine-tune the model, we utilized `keras_tuner` to search among different hyperparameters:

- Learning rate: {1e-3, 4e-4, 5e-5, 6e-6}
- Batch size: {4, 8, 16}
- Hidden size of the first Dense layer: {64, 128, 256, 512}
- Hidden size of the second Dense layer: {32, 64}

Additionally, we incorporated dropout with a rate of 0.5 to the first dense layer and a dropout rate of 0.3 to the second dense layer. For generalization purposes, we applied L2 normalization to the second dense layer.

To address the issue of imbalanced data, we assigned class weights {0: 0.68, 1: 1.85} to the model.

We split the training set into training and validation sets with a ratio of 0.1 and selected the model with the best AUC on the validation set.

4.6 Early Stopping Criteria and Learning Curves

The point at which to stop training varies for different model configurations. As a general guideline, we halt training when the validation loss ceases to decrease or starts to increase while the training loss continues to decrease. This indicates that the model is beginning to overfit the training data.

For the configuration with:

- Batch size: 8
- Learning rate: 5e-5
- Hidden size of the first dense layer: 128
- Dropout rate for the first layer: 0.5
- Hidden size of the second dense layer: 64
- Dropout rate for the second layer: 0.3

The following learning curve is observed. As depicted, from epoch $\sim$ 400 onwards, the validation loss ceases to decrease while the training loss continues to decrease. At this point, the model begins to overfit the training data. Therefore, we should stop training to prevent overfitting.

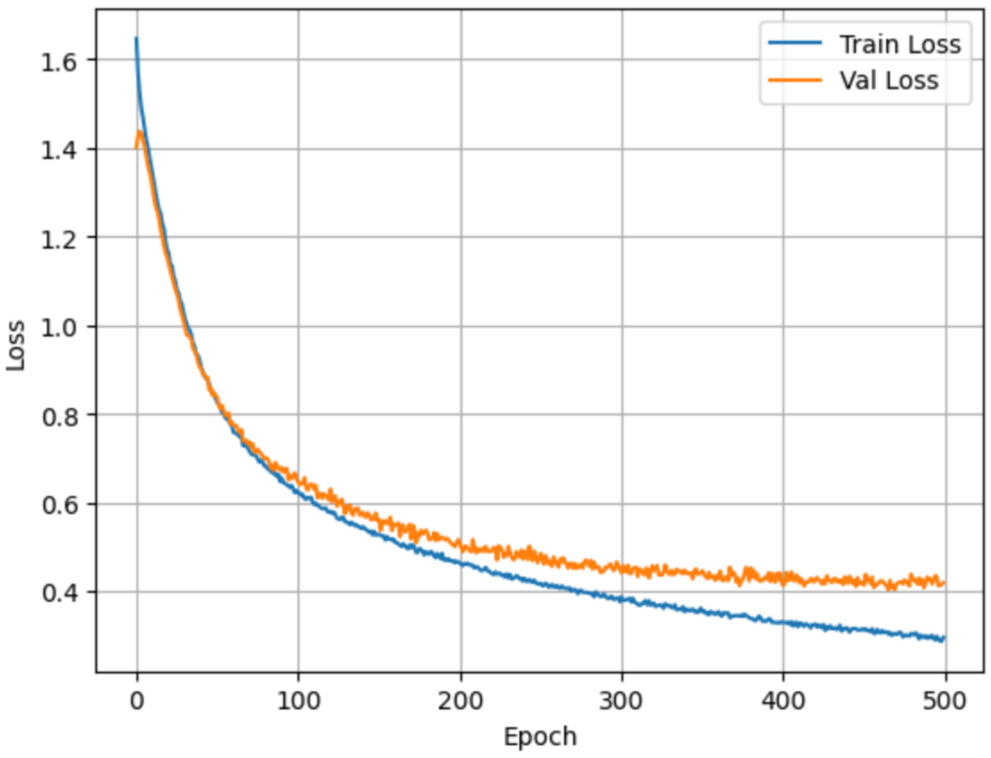


Figure 24: Learning curves for training and validation sets

Model Performance and Results Analysis

Following the model training, we achieved a performance of 87.5% AUC score on the test set. Additionally, we trained a logistic regression model on the same training data and attained an AUC of 84.5% on the test set. This experiment indicates that the input data may not contain as much informative content as what we derived using the TF-IDF vectorization.

Given the constraint of 512 tokens for BERT models, any text exceeding this limit must be truncated, leading to information loss. Our analysis revealed that out of 5000 data points, 3468 had more than 500 tokens. This limitation may be one of the primary reasons why BERT embeddings do not outperform TF-IDF vectors. While TF-IDF vectors encompass all tokens in the input text, BERT embeddings only include the first 512 tokens.

A potential solution to address this limitation is to utilize Longformer (`allenai/longformer-base-4096`), which is designed to handle long texts. However, we did not explore this option due to resource constraints, particularly RAM limitations.

Another approach involves splitting the long text into chunks of 512 tokens, feeding each chunk to BERT, and then combining the embeddings of these chunks, either by taking the mean or concatenating them. This method could potentially preserve more information from longer texts.