

EECS 4412 Project

Michael Angelo Masangkay
219005867
York University
Toronto, Canada
mixhael@my.yorku.ca

Jason Lau
218835066
York University
Toronto, Canada
jason245@my.yorku.ca

ACM Reference Format:

Michael Angelo Masangkay, 219005867 and Jason Lau, 218835066. 2025. EECS 4412 Project. In . ACM, New York, NY, USA, 3 pages. <https://doi.org/10.1145/nnnnnnn.nnnnnnn>

1 Introduction

In this assignment we explored text classification with two different implementations, Bag of words and text embeddings. This report will talk about both implementation in detail, and the various decisions we chose to improve our accuracy or make it more efficient.

2 Text classification with Bag Of Words

2.1 Preprocessing

When preprocessing our data for the BOW method, we decided to set the text to lowercase, tokenize the words, and remove punctuation and stopwords. This was done to eliminate irrelevant information and noise that could negatively impact accuracy, allowing the model to focus more on the key content of the text. We used and compare 3 different classifiers which include Random Forest, XGBoost, and AdaBoost. When obtaining the results without preprocessing on these classifiers, their F1-score was low at approximately 50-61 percent.

	BOW Accuracy	BOW Precision	BOW Recall	BOW F1
Random Forest	0.793167	0.742856	0.542315	0.536016
XGBoost	0.807917	0.707919	0.594457	0.608429
AdaBoost	0.754417	0.630625	0.510205	0.513671

Figure 1: Classifier Comparison

2.2 Reasoning and Discussion

After comparing the results for each classifier, we decided to proceed with XGBoost, as it performed the best. Following the preprocessing of the text and the application of SMOTE, we observed a significant improvement in precision, recall, and F1-score, as indicated by the weighted average. Without rebalancing using SMOTE, the BOW method struggled with class imbalance, particularly for

Permission to make digital or hard copies of all or part of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. Copyrights for components of this work owned by others than the author(s) must be honored. Abstracting with credit is permitted. To copy otherwise, or republish, to post on servers or to redistribute to lists, requires prior specific permission and/or a fee. Request permissions from permissions@acm.org.

Conference'17, July 2017, Washington, DC, USA

© 2025 Copyright held by the owner/author(s). Publication rights licensed to ACM.
ACM ISBN 978-x-xxxx-xxxx-x/YY/MM
<https://doi.org/10.1145/nnnnnnn.nnnnnnn>

the neutral class. However, after applying SMOTE, the performance for the neutral class improved, even though the precision, recall, and F1-score values for this class remained relatively low.

	precision	recall	f1-score	support
negative	0.79	0.67	0.72	2784
neutral	0.35	0.35	0.35	1359
positive	0.84	0.89	0.86	7857
accuracy			0.78	12000
macro avg	0.66	0.64	0.65	12000
weighted avg	0.77	0.78	0.77	12000

Figure 2: Results With Preprocessing

Next, we implemented feature selection and tested various values of k (the number of features to select). After testing multiple values ranging from 500 to 3000, we observed no significant change in the weighted average for precision, recall, and F1-score. The largest change was a 1 percent increase in recall when k was set to 1000. Additionally, we experimented with different values for the maximum number of features in the vectorizer. Both increasing and decreasing the original value of 5000 resulted in worse performance. We then increased the number of estimators in XGBoost from 100 to 200, which led to a 2-3 percent improvement in the weighted average for precision, recall, and F1-score. Further increasing the number of estimators to 500 resulted in an additional 2 percent improvement. With preprocessing and the implementation of these features, training the model took around 5 to 6 minutes, but when increasing the number of estimators to 500, training time increased to approximately 10 minutes. After training the classifier with these features, it predicted the classes as positive, negative, and neutral, with counts of 40,965, 13,126, and 5,909, respectively.

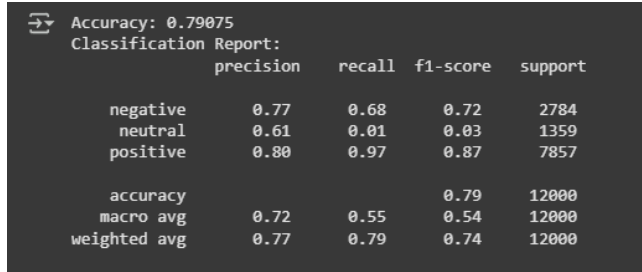
	precision	recall	f1-score	support
negative	0.81	0.78	0.80	2784
neutral	0.43	0.36	0.40	1359
positive	0.88	0.91	0.90	7857
accuracy			0.82	12000
macro avg	0.71	0.69	0.70	12000
weighted avg	0.81	0.82	0.82	12000

Figure 3: Results With Feature Selection and Tuned Parameters

3 Text classification with text embedding

3.1 Models tested

We've decided to test 5 models from huggingface which include: MiniLM-L6-H384-uncased, MiniLM-L12-H384-uncased, distilroberta-base, all-mpnet-base-v and paraphrase-albert-small-v2. We first tested them just using random forest and used classification report to report all metrics. We got the following results:

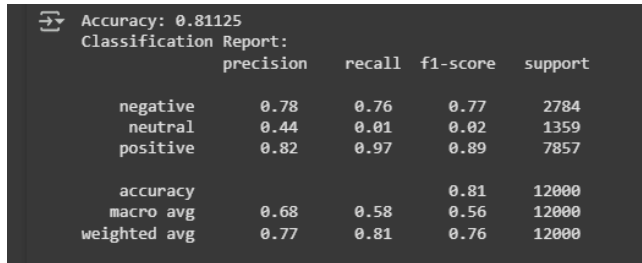


```
Accuracy: 0.79075
Classification Report:
      precision    recall  f1-score   support

 negative      0.77      0.68      0.72      2784
  neutral      0.61      0.01      0.03      1359
 positive      0.80      0.97      0.87      7857

 accuracy              0.79      12000
 macro avg      0.72      0.55      0.54      12000
 weighted avg      0.77      0.79      0.74      12000
```

Figure 4: MiniLM-L6-H384-uncased

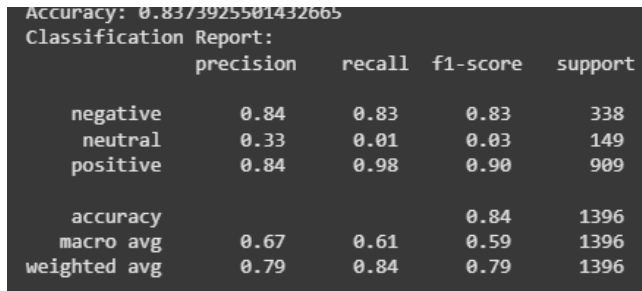


```
Accuracy: 0.81125
Classification Report:
      precision    recall  f1-score   support

 negative      0.78      0.76      0.77      2784
  neutral      0.44      0.01      0.02      1359
 positive      0.82      0.97      0.89      7857

 accuracy              0.81      12000
 macro avg      0.68      0.58      0.56      12000
 weighted avg      0.77      0.81      0.76      12000
```

Figure 5: MiniLM-L12-H384-uncased

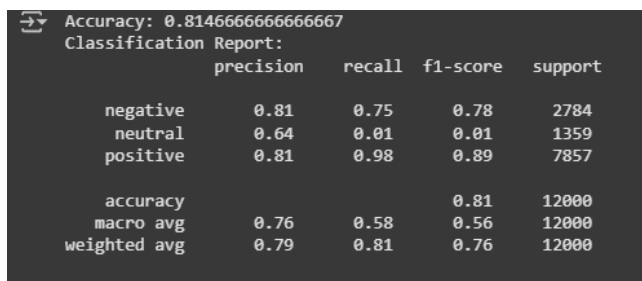


```
Accuracy: 0.8373925501432665
Classification Report:
      precision    recall  f1-score   support

 negative      0.84      0.83      0.83      338
  neutral      0.33      0.01      0.03      149
 positive      0.84      0.98      0.90      909

 accuracy              0.84      1396
 macro avg      0.67      0.61      0.59      1396
 weighted avg      0.79      0.84      0.79      1396
```

Figure 6: distilroberta-base

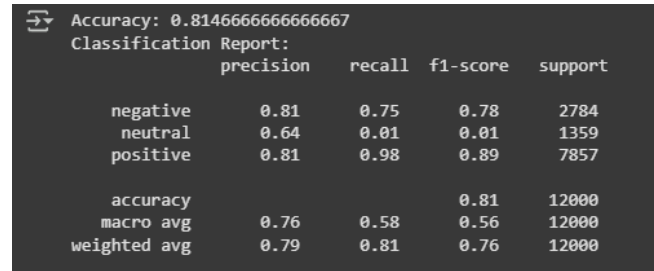


```
Accuracy: 0.8146666666666667
Classification Report:
      precision    recall  f1-score   support

 negative      0.81      0.75      0.78      2784
  neutral      0.64      0.01      0.01      1359
 positive      0.81      0.98      0.89      7857

 accuracy              0.81      12000
 macro avg      0.76      0.58      0.56      12000
 weighted avg      0.79      0.81      0.76      12000
```

Figure 7: all-mpnet-base-v



```
Accuracy: 0.8146666666666667
Classification Report:
      precision    recall  f1-score   support

 negative      0.81      0.75      0.78      2784
  neutral      0.64      0.01      0.01      1359
 positive      0.81      0.98      0.89      7857

 accuracy              0.81      12000
 macro avg      0.76      0.58      0.56      12000
 weighted avg      0.79      0.81      0.76      12000
```

Figure 8: paraphrase-albert-small-v2

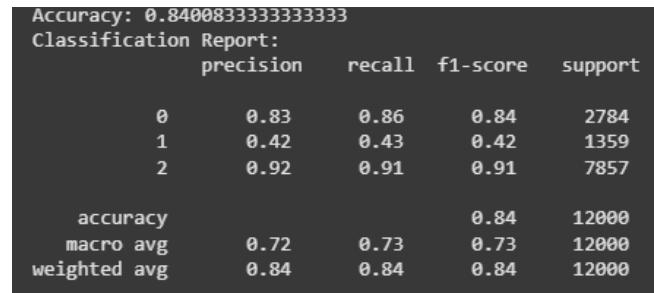
3.2 Reasoning and Discussion

Computing each model took a long time. Model 4 which was the all-mpnet-base-v model took the longest at 30 minutes, for not much improvement which was disappointing.

After comparing each model, we concluded that model3 which was distilroberta-base which had the highest average f1 score of 0.83. We then decided to use distilroberta-base and try to increase its f1 score by trying to rebalance the dataset and change classifiers. The F1 score of the neutral class was also very low at 0.01 for most of the models, meaning rebalancing was needed

To rebalance the dataset we used SMOTE which added more information to the lower classes. As this dataset had 3 classes, negative, neutral and positive, it was favoring the most popular class which was positive. Before any rebalancing, it only returned negative and positive classes. Using SMOTE to rebalance was essential to getting more accuracy from the classifier.

On the choice of classifier, from previous assignments we are now aware that xgboost is better and more accurate than random forest, which was shown in the results.



```
Accuracy: 0.8400833333333333
Classification Report:
      precision    recall  f1-score   support

    0      0.83      0.86      0.84      2784
    1      0.42      0.43      0.42      1359
    2      0.92      0.91      0.91      7857

 accuracy              0.84      12000
 macro avg      0.72      0.73      0.73      12000
 weighted avg      0.84      0.84      0.84      12000
```

Figure 9: distilroberta-base with xgboost

Here we can see that the f1 scores of each class (0 = negative, 1 = neutral, 2 = positive) have all improved from the random forest without smote. We can see a significant increase of the f1 score on the neutral class, which is because of the smote rebalancing. Overall the accuracy has also improved to an average of 84 from 83, which is marginal. With these improvements we were able to overall improve the metrics compared to the original.

4 Comparison

One significant difference between BOW and text embeddings is the computational time. On average, the BOW method takes

around 5 minutes to train the classifier while the best model for text embedding takes 30 minutes. The reason for this is because BOW involves counting the frequency of words in the text, which is a relatively simple and fast process. It generates sparse matrices based on word counts or term frequencies, which are easy to compute. On the other hand, text embeddings require complex models that capture the semantic meaning of words and generate dense vector

representations. It comprises of multiple layers of transformations and computations making it time-consuming and expensive.

Another difference is the accuracy and F1 score differences. We can see from the BOW implementation that it had lower F1 scores and accuracy than the text embedding implementation. Though the difference between the two is not that large, the text embedding implementation still yields higher metrics.