Introduction

The objective of this project is to build a model that will determine the polarity (neutral, positive, negative) of tweets. To do this we will train the model on the provided data. The resulting model will have to determine the class (neutral, positive, negative) of new data (test data that were not used to build the model) with maximum accuracy and fscore.

Data description

During the baseline experiment phase and enhancement experiments we used the provided training data and we got acceptable accuracy (maximum around 66% and when we submitted the best model on kaggle we got around 66% fscore)

opinion	Instances
negative	2374
neutral	6840
positive	6827

After that we've downloaded another dataset from kaggle which unfortunately doesn't contain neutral instances but we tried to merge sample of this large downloaded dataset(only 30%) to the already provided training data to get higher accuracy and fscore and we've successfully got higher accuracy(around 76%) but the fscore wasn't as expected so using the downloaded data wasn't a part of our winning recipe which we will discuss later in the documentation. Number of instances after adding external data is as below noting that we've also tried to make instances in the three classes equal by just adding negative instances but fscore didn't improve on kaggle competition.

opinion	Instances
negative	242150
neutral	6840
positive	247051

For the whole project we used 70 to 30 training test split technique noting that we used stratify parameter for some time but it didn't show great change.

Baseline experiments

Our goal in baseline experiment is to choose which classifier we will use to get highest accuracy/fscore also we needed to judge which is better to be used as vectorizer count vectorizer or tfidf vectorizer.

Based on the results obtained we can conclude that logistic regression is the best classifier and linear SVM follows while Naïve bayes and rbf-kernel SVM is much worse so we decided to use logistic regression in the enhancement experiment part.

Also based on choosing logistic regression classifier we decided to use count vectorizer (unigrams and bigrams will be tested) instead of tfidf vectorizer. A notebook named **baseline experiment** records the experiments result.

Enhancement experiments

We will discuss below each step we made during enhancement experiment separately noting that all the below steps can be found in **enhancement experiment** notebook. Logistic regression was used for all the below steps.

1. Data preprocessing

After several trials to get the best accuracy using different combinations we decided to use casefloding, stopwords removal, digits removal and trimming noting that stemming and punctuation removal didn't show good results.

During baseline experiment and using logistic regression we got accuracy of 50.7% and fscore of 47.3% which is significantly enhanced after data preprocessing as accuracy reached 64.5% and fscore 64.44% as shown in below table

precision	recall	fscore	accuracy
65%	65%	64.44%	64.5%

Table 2 – Data preprocessing Experiment results

2. Hyperparamter tuning

We used grid search in order to get the best parameters to be used in the further experiments and we changed many parameters like C, penalty(l1,l2), solver(lbfgs, liblinear) and multi_class(ovr, multinomial) and we can conclude that changing C value to 0.6951927961775606 slightly enhance accuracy with no clear impact on fscore, precison or recall.

precision	recall	fscore	accuracy
65%	65%	64.44%	64.96%

Table 3 – Hyperparameter Experiment results

3. New features addition

During the previous steps we applied logistic regression in the vector resulted from countvectorizer but we need to enhance evaluation metrics more so we decided to add some features that can help us in our target to get the maximum accuracy, fscore.

We will mention below the features added one by one.

a. Word Embedding

In this step we decided to use w2v instead of countvecotrizer but we didn't get good results and both fscore and accuracy decreased as shown below.

precision	recall	fscore	accuracy
63%	63%	62%	63.14%

Table 4 – w2v Experiment results

b. Word Embedding+Count vectorizer

After merging count vectorizer vector and the vector resulting from word embedding we found enhacment in both accuracy and fscore(around 1% for each compared to hyperparameter tuning step) as indicated in the below table.

precision	recall	fscore	accuracy
66%	66%	66%	66.09%

Table 5 – w2v+countVectoizer Experiment results

c. Using Unigrams+Bigrams for countvectorizer

Instead of using only unigrams to generate countvectorizer we added Bigrams as well and we got enhancement compared to hyperparameter tuning experiment but enhacement is less that what we get from using Unigram countvectorizer + w2v.

precision	recall	fscore	accuracy
66%	65%	65%	65.47%

Table 6 – (Unigrams+Bigrams)countVectoizer Experiment results

d. Using Hashing vectorizer

Finally we decided to use hashing vectorizer instead of countvectorizer as Hashing vectorizer helps in dimensionlaity reduction but as the data itself isn't that big we didn't notice enhancement compared to using w2v + countvectorizer.

precision	recall	fscore	accuracy
65%	65%	65%	65.3%

Table 7 – HashingVectorizer Experiment results

4. Bagging, boosting and Ensemble voting

Now it is time to involve ensemble learning. We started with bagging(count vectorizer only and unigrams) and we made n_estimator tuning where we get good accuracy and fscore results while we didn't get better results when we used boosting or ensemble voting(combining logistic regression and linear SVM). Results were recorded in the below tables.

precision	recall	fscore	accuracy
66%	66%	65%	66%

Table 8 – Bagging Experiment results

precision	recall	fscore	accuracy
63%	62%	61%	62%

Table 9 – Boosting Experiment results

precision	recall	fscore	accuracy
66%	65%	65%	65.15%

Table 10 – Ensemble voting Experiment results

After finishing the above experiments we tried to get the best result on kaggle test data and we got fscore of 66.27% when we used countvectorizer+w2v+logistic regression but we aimed to get better results after noticing that all classifiers results in low recall for negative class so we decided to add more training data and we found that data on kaggle website as well in the following link
https://www.kaggle.com/kazanova/sentiment140#training.1600000.processed.noemoticon.csv

Our main problem with this data as discussed earlier that it doesn't contain neutral instances so we took 30% sample of this data and we added it to the training data that we already have. Then we got perfect accuracy, fscore when we applied it on validation data(test data output from train_test_split) as you can see below noting that you can see these results in **enhancement experiment with additional data** notebook.

precision	recall	fscore	accuracy
76%	76%	76%	76%

Table 10 – Additional data+logistic regressionExperiment results

When we applied this model on kaggle test data we surprisingly got very low fscore values but when we further checked the reason we found that recall, fscore for neutral class were very poor(recall is 40%) and it seems that the test data includes too many neutral class instances so we decided to adhere to the training data and make extra experiments to get higher fsocre in the competition.

External Resources

As mentioned above we used sample of the below kaggle dataset data to get more training data.

https://www.kaggle.com/kazanova/sentiment140#training.1600000.processed.noemoticon.csv

Overall Conclusion

Using Bagging + logisiticRegression + countVectorizer + w2v resulted in highest fscore of 66.59% when submitting on kaggle. However, we got perfect fscore, accuracy values when we added additional training data(76% for both) but with some weakness on fscore values of neutral class which seems to be heavily presented in test data and that's the reason we didn't get higher fscore values when we added additional data as training.

The model that resulted in best fscore will be found in **best model** notebook.