Emotion Classification using Naive Bayes and Logistic Regression

Nam Le

I. Introduction

In the real world, all companies strive to improve user experience with their application. The traditional way of doing this would be surveying customers for feedback, or sending people to do research online for opinions, both of which are greatly demanding in manpower. A company can greatly reduce costs by instead using a Natural Language Processing (NLP) model tuned for sentiment analysis to gather this data. [1]

Another example would be stock predictions in finance. As stocks gets affected by news and public sentiment, an investment firm might deploy traditional techniques (like the ones mentioned above) to better predict trends. Once again, we can use sentiment analysis to do the majority of the work. [2]

The main motivation for doing this project is for understanding the fundamental building blocks of ChatGPT following its massive success [3].

II. Project details and framing

We can observe that in most piece of text, the author writes it with an emotion in mind. As such, one should be able to extract this information from the text.

Given a textual input, categorize the emotion present in the text as one of the following: Sadness, Joy, Fear, Anger, Love, Surprise

In more technical speak, this is a supervised learning, multi-class classification problem with the following specification:

- Input: A piece of text of any length
- Output: One of six possible emotions most likely to be associated with it

We determine a successful outcome of a given text input based on whether the output emotion produced matches the human assigned emotion. We will mainly use accuracy to determine a model's success, though an analysis of other evaluation statistics will be considered.

A. Proposed ML system

To solve this machine learning task, we follow a basic workflow as seen in Figure 1. In blue is the data preparation and processing aspect, and in orange is all that has to do with the ML model.

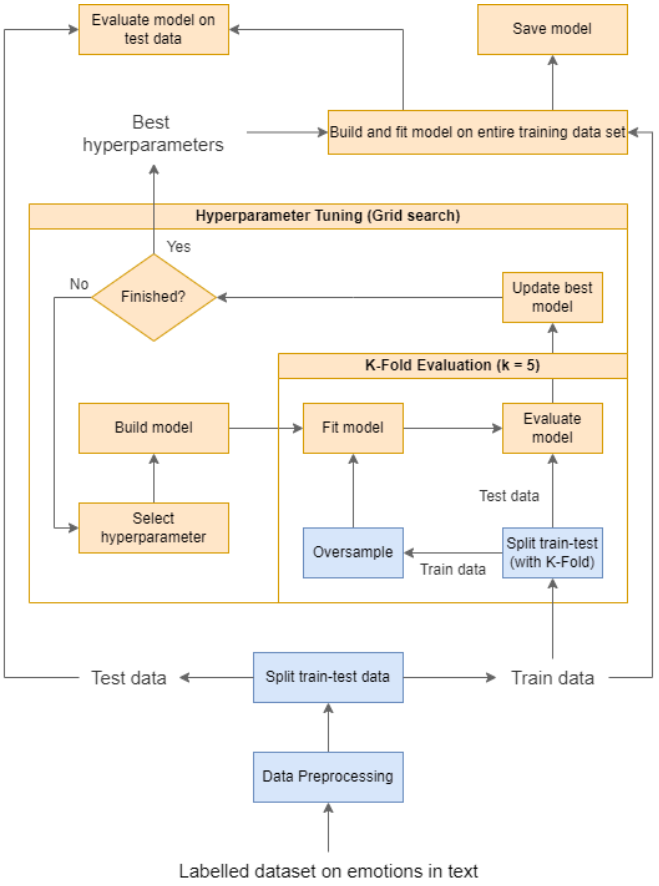

Figure 1: Machine Learning System

III. Data and Experimental setup

A. Data set description

The dataset used [4] contained three .txt files: train.txt, test.txt and val.txt.

In each of these files, the data is presented in this manner:

- *sentence;emotion.*

For example im feeling rather rotten so im not very ambitious right now;sadness

In total, there are 20,000 instances, but only 19,948 unique instances.

This dataset contains 6 different labels. All instances are labelled with one of these 6. There is no 'neutral' class. As for the quality of the data, kaggle rates the data a usability score of 10/10.

Emotions	Counts	% of total
Sadness	5797	28.98%
Joy	6761	33.80%
Fear	2373	11.87%
Anger	2709	13.54%
Love	1641	8.21%
Surprise	719	3.60%

Table 1: Counts of emotions before cleaning

Table 1 also shows an inherent flaw with the dataset, which is that it is moderately imbalanced, with the minority class taking up only 3.6% of the total. An ideal data set should give every class around 16.7%% of the total.

B. Data processing and preparation

As we want to split the data in our own way, we decided to merge the three .txt files together. Also, we decided to save it as a csv file to more easily read using pandas.

We mentioned above how in the data set there are less unique than total instances. This indicates duplicate data, and we have hence removed those.

In any NLP task, we must remove stop words as they contain no discernable information and makes our model slower to train, due to their high frequency in text [5]. We also should lemmatize words, as this helps reduce the size of our corpus, while removing redundant information (such as conjugation) [6].

To do this, we make use of an external library called Spacy and its model. As this task is quite time-consuming, we store the result in another csv file for ease of use.

As we need to pass our labels into the ML models, we must convert it from a text format into a numerical format. We do this by mapping each value to a number. There is no need to do one-hot encoding as our ML models only use the labels to check for correctness and does not do any learning on the labels itself.

Emotions	Counts	% of total
0	5794	29.05%
1	6743	33.80%
2	2366	11.86%
3	2704	13.56%
4	1628	8.16%
5	713	3.57%

Table 2: Counts of emotions after cleaning

Finally, we then split the entire data set into training and test with an 80-20 ratio. Notice that there is no validation set here. This is deliberate, as we will be using K-Fold evaluation, which serves as our validation set for tuning hyperparameters.

C. Experimental Setup

1) *Oversampling:* The main issue at hand to deal with is imbalanced data. To handle this, we use oversampling, a method which duplicates instances of under-represented classes to get an equal distribution. This step, however, should only be applied to the training data set, not the validation and tests. We used imblearn's Random Over Sampler.

2) *Feature extraction:* We use TF-IDF to convert the processed data into vector form and extract features from it. TF-IDF works by taking into consideration the frequency of words in the training data set (TF), as well as measuring how important a term in with respect to the entire corpus (IDF).

We opted for TF-IDF instead of Bag Of Words, or Word2Vec, as it is a middle ground between simple and complex. Perhaps Word2Vec would perform better, but as our models and problem is quite simple, TF-IDF should be efficient enough.

Note that this feature extraction step is part of our model and so is not shown as a separate step in Figure 1.

We used sklearn's TFIDFVectorizer function in our implementation.

3) *Evaluation metrics:* For our task, a high rate of false positives or false negatives is not really an issue, unlike in ML tasks like medical diagnosis or fraud detection where these metrics are more important. As such, we place a higher emphasis on accuracy and f1 score. It is for this reason that when calculating score in K-Fold, we take higher weights for them compared to the others.

Another important metric to consider is AUC score, which, like accuracy and f1, gives us a good indication of how well the model can distinguish between classes and performs in general.

IV. Model Evaluation

A. Model selection

For our baseline model, we decided on using Naive Bayes (NB) due to its simplicity, yet decent performance on these categorizations task. In particular, we used skearn's MultinomialNB function for this.

The other model chosen is Logistic Regression (LR). Naive Bayes makes the assumption that all features are independent to each other which, while effective, is not the case in actual text. As such, we decided on using a model which can capture more complex relationships between words. Also, LR is a model which performs better on larger inputs, so this model should also scale much better than NB [7]. We used sklearn's LogisticRegression function for this model.

B. Training process

For both models, we use the following pipeline:

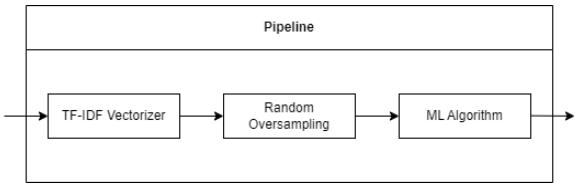

Figure 2: Model pipeline

As Random Oversampling is not part of the sklearn library, we cannot use its Pipeline, but instead we need to use imblearn's Pipeline function. After fitting this pipeline on some data, we get back a functioning model ready for predictions.

For Naive Bayes, we do not really have a loss function, as all the algorithm does is calculate probabilities from the training data set. Also, as we oversampled the data, talking about class prior is also redundant. The only thing that we can talk about is the feature log probability, which has a shape of (6, 89049).

For Logistic Regression, as we are working with multi-class, our loss function is Cross-Entropy Loss, which is a generalization of Log Loss for multi-class cases. We see in the parameters of our trained LR model the intercept values, which is greater for underrepresented classes.

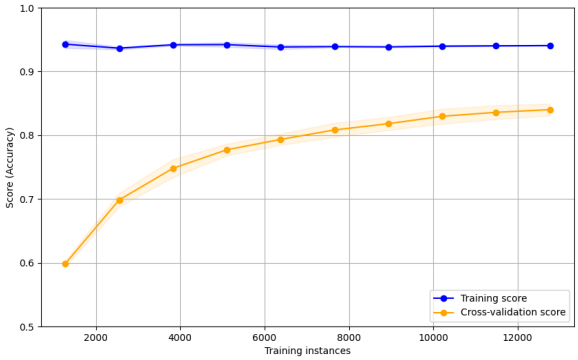

Figure 3: Naive Bayes learning curve

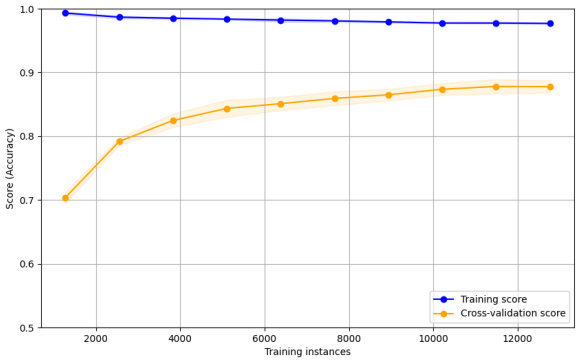

Figure 4: Logistic Regression learning curve

Figure 3 and Figure 4 shows that our model overfits quite a bit as the training score is quite a bit higher than cross-validation, but this is expected as we had to use oversampling to make up for the imbalanced data.

We can see that both graphs shows the cross-validation scores eventually reaching a plateau as more data is learnt. We can assume from this that the optimiziation converges around 0.84 for NB and 0.89 for LR at around 13,000 instances.

C. *Hyperparameter tuning*

The hyperparameters that we need to consider for both models comes mainly from TF-IDF and the model algorithm.

- **TF-IDF**: We consider max features, ngram range, max-df, norm used and smooth-idf.
- **Naive Bayes**: The only hyperparameter for this is alpha values.
- **Logistic Regression**: We consider the solver used, C values, multi-class options (ovr vs multinomial), tolerance value and max iterations.

We find the optimal values using Grid Search, and running through all combinations. We use accuracy to decide the best model, and we also use a CV value of 5. This means using K-Fold to evaluate the model for each hyperparameter.

D. *Model comparison*

By using sklearn's classification report, we get the following:

Emotions	Precision	Recall	F1
Sadness	0.94	0.88	0.91
Joy	0.96	0.81	0.88
Fear	0.80	0.84	0.82
Anger	0.85	0.91	0.88
Love	0.66	0.91	0.76
Surprise	0.52	0.91	0.66

Table 3: Naive Bayes Result (0.86 Accuracy)

Emotions	Precision	Recall	F1
Sadness	0.95	0.91	0.93
Joy	0.93	0.89	0.91
Fear	0.87	0.83	0.85
Anger	0.88	0.91	0.89
Love	0.76	0.90	0.82
Surprise	0.70	0.88	0.78

Table 4: Logistic Regression Result (0.89 Accuracy)

We can see from Table 3 and Table 4 that while there is only a 3% difference in accuracy, LR has much better precision for under-represented classes (Love and Surprise), and has a better F1 score across all classes. This can be further seen in Figure 5 and Figure 6, where LR misclassify love and surprise less.

Figure 5: Naive Bayes confusion matrix

Figure 6: Logistic Regression confusion matrix

E. Model finalization

After finding an optimal set of hyperparameters, we train the model on the entire training data set (that has been oversampled), and get the performance metrics by running it on the testing data set. We also save the model for later use.

V. Summary

In conclusion, we find that Logistic Regression is better than Naive Bayes for emotion classification, beating it in not only accuracy by around 3-4% but F1 score, especially for under-represented classes.

However, there are some limitations in this project. The main point of concern is both models overfitting the training data, caused by oversampling. However, without oversampling, both models fail miserably to predict any classes besides the most over-represented ones. Also, our model does not have a 'no emotion' label.

A lesson to takeaway from this is to choose a good data set to begin with in order to avoid overfitting and get text with no-emotion labels. Also, using a more complicated library would allow for more liberties like plotting loss curves.

VI. Self Evaluation

From the lectures, I learnt what ML actually is about, as well as the workflow present in every ML task. I also learnt about many different tools I could apply in each stage of the workflow. Examples of this would be in the model selection section, where I was taught about the many different options available, or in the model evaluation, where I learnt about different metrics for determining a model's success (or its failure). Furthermore, I got a brief look as to how the maths works out in these models.

The coursework was a natural progression from the lectures, allowing me to apply the knowledge learnt in a real project. I would say I enjoyed it a lot more as I got to play around and get a hands-on experience with ML. Learning how to write a formal paper was also very fun (and will most likely be useful in the future).

I think the most difficult part of the module was the heavy mathematical aspects behind all these models, and trying to understand it thoroughly. I get the intuition behind the models, but not really how it technically works, and would hence not be able to implement it in just NumPy, for example.

If I had to do this module again, I would try to understand the workflow more. As I did not really understand why it was there before, I got quite confused at the structure of the course and could not follow everything perfectly well.

I believe that this project was quite normal compared to other existing ML works.

References

[1] C. N. Dang and M. N. Moreno-García, "An Approach to Integrating Sentiment Analysis into Recommender Systems". Aug. 23, 2023. [Online]. Available: https://www.mdpi.com/1424-8220/21/16/5666

[2] Q. Xiao and B. Ihnaini, "Stock trend prediction using sentiment analysis". Mar. 20, 2023. [Online]. Available: https://www.ncbi.nlm.nih.gov/pmc/articles/PMC10403218/

[3] F. Duarte, "Number of ChatGPT Users". [Online]. Available: https://explodingtopics.com/blog/chatgpt-users

[4] Praveen, "Emotions dataset for NLP". [Online]. Available: https://www.kaggle.com/datasets/praveengovi/emotions-dataset-for-nlp/data

[5] wisdomml, "What are Stop words in NLP and Why we should remove them?". [Online]. Available: https://wisdomml.in/what-are-stopwords-in-nlp-and-why-we-should-remove-them/

[6] engati, "Lemmatization". [Online]. Available: https://www.engati.com/glossary/lemmatization

[7] S. Deb, "Naive Bayes vs Logistic Regression". [Online]. Available: https://medium.com/@sangha_deb/naive-bayes-vs-logistic-regression-a319b07a5d4c