Assignment 2 COMP30027 Report

Anonymous

1. Introduction

Understanding and deriving meaning from words, be it spoken or written, is a natural and integral part of a person's day – without it, simple tasks as learning a recipe or reading a book will be impossible. Now, we are sharing this ability with machines and Artificial Intelligence, through a field known as Natural Language Processing (NLP).

In an article by Gupta, a popular feature, especially among businesses and organisations, is the analysis of text and determining whether the underlying sentiment is positive, negative, or neutral: Sentiment Analysis. Sentiment Analysis can allow companies to analyse online conversations posted on social media or reviews posted on Yelp and determine how customers view their products. Extensions of Sentiment Analysis include Intent Analysis and Contextual Semantic Search.

2. Vectorizing the data

Unlike humans, for machines to "understand meaning behind words", we first need to transform our unstructured data into a numeric representation of the text through vectorization (Kub). The basic version is to count the frequencies of word occurrences in a text using the BagOfWords algorithm, facilitated through the use of CountVectorizer.

```
('rude', 1.2419884655366749)
('worst', 1.189477580476503)
('terrible', 0.9816836278499985)
('overpriced', 0.9619470822144517)
('horrible', 0.9346316907284365)
('delicious', -0.9538910492530771)
('amazing', -0.832934851492662)
('awesome', -0.7057677171085484)
('excellent', -0.6301429429540839)
('best', -0.6245928513095459)
```

Figure 1- Top 5 positive and negative discriminating words when using CountVectorizer

While we can easily remove words such as "if", "but", "she", etc. through the tweaking of the stop_words hyperparameter, some words such as "restaurant" do not really fall into the category of stop words, but it obviously has a very low significance in a list of reviews all on restaurants. This is the basis of term frequency-inverse document frequency (tf-idf or TFIDF), a numerical statistic used as a weighting factor to determine the significance of a word to a text body in a corpus, increasing in proportion to the word frequency in a text, offset by the number of documents in the corpus that the word appears in. This helps to reduce the bias found in CountVectorizer towards these insignificant words.

```
('rude', 3.0223825830725515)
('worst', 2.8214919188119896)
('horrible', 2.3078442181456813)
('terrible', 2.222437908898581)
('money', 2.1203088400115893)
('great', -3.013220894430284)
('delicious', -2.5803500532609838)
('good', -2.2531424932560147)
('amazing', -2.1743228408116893)
('best', -2.153717814993642)
```

Figure 2- Top 5 positive and negative discriminating words when using Tf-idf Vectorizer

2.1 Feature Selection

Without discriminating between which vectorizer to use yet, both vectorizers resultantly provide us a vocabulary dictionary in excess to 40000 words. These 40000+ words are likely to include words with spelling errors or words with low information value. Our vectorizers transform our data into a sparse matrix representation and these insignificant words can significantly impede on our computation time. This prompted the use of either dimension reduction through Principal Component Analysis (PCA) or feature selection.

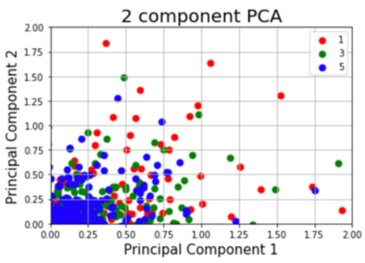

Figure 3- Principal Component Analysis on the original dataset.

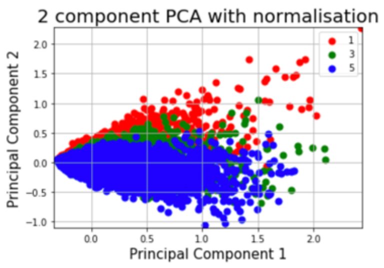

Figure 4- Principal Component Analysis on the normalised dataset.

When visualising our data using PCA, it was very hard to discern one class from another, but once paired with a normalised version of our data, one can see that it looks reasonably separable. That said, when we transform the data itself, it results in sub-par results that are reasonably similar to a Zero-R model, or even worse depending on which classifier is used (further explained in later parts of the report).

This shows that using PCA to reduce the dimensionality of the feature is very bad as it removes too many features. Hence, a feature selection method can be applied using the chi-squared or mutual information methods. Both methods produce very similar results, however the chi-squared method is computationally a lot faster than the mutual information and was hence opted for. After a few rounds of experiments, the final value of *k* used was 15000 for both vectorizers. Values of *k* lower than 10000 resulted in the accuracy being greatly impeded as there was too much information being lost and values above 17500 were slightly lower and was deemed inefficient.

3. Classifiers that we can use

This is where NLP generalises into a basic machine learning function. With so many classifiers available, it becomes question of which classifier is best suitable for the data we have, and one of the models most suited to handle sparse data would be a Logistic Regression model. This model can not only learn very fast but is also easy to interpret as it directly relates how important a word is when classifying the review and provides the direction of its association. It is mainly used in a binary classification problem, but it can be easily modified to support multiclass classifications.

A Support Vector Machine model with a linear kernel is also efficient in sparse data such as this and is also very computationally efficient. On top of that, while the data may not be linearly separable (see Figure 4), it would be reasonable to hypothesise that a softer margin could help as a workaround.

Other models such as Naïve Bayes, Decision Tree, and kNN were also considered but such models are biased against sparse data such as this. That said, a multinomial Naïve Bayes may be able to capture parts of the classification that the Logistic Regression and SVM models may not and was hence added into a stacking classifier. This stacking classifier has the Logistic Regression, SVM and multinomial Naïve Bayes models as the first tier of classifiers, and the Decision Tree model as the metaclassifier.

4. Combining and Tuning the Hyperparameters

With two vectorizers to transform our data, multiple classifiers as well as feature selection options at our disposal, it was time to combine them and determine which pair of vectorizer and classifier works best. This also includes experiments conducted to determine the right hyperparameters.

4.1 *n*-grams

One such hyperparameter included the *n*-grams hyperparameter in the vectorizers. As humans, we know that the sequence of the words that appear in a sentence itself plays a significant role in expressing one's meaning, i.e. words are joined to form phrases and our focus should be on reviewing those phrases. For example, a review containing "didn't enjoy food" might be classified as a positive review had we only considered "enjoy" on its own.

That said, a larger value for *n* may not necessarily improve our model. Another factor to consider is that the size of our sparse matrix grows exponentially with a larger *n* (Kub). Hence, during the experimenting phase, only bigrams (word pairs) were considered. Tweaking this hyperparameter resulted in a decent increase in the accuracy score – an increase even higher than that of the feature selection. However, when feature selection was combined with an increase in *n*-grams, there was a slight decrease in accuracy as compared to modelling using the full matrix produced with the increase *n*-grams. This prompted in the prioritisation of the *n*-grams over the feature selection, and in the final submissions, an *n*-gram value of 3 was selected.

4.2 Regularisation – Soft margins

As we know, the data is not linearly separable (see Figure 4), but with tuning of the *c* hyperparameter in the Logistic Regression and SVM models, not only can the SVM model better separate the data and classify the labels, but this also ensures that all our classifiers do not overfit our training data, thus reducing bias towards noise. That said, the different values that *c* could take were the main contributor to the variance in the accuracy, i.e. an inappropriate value of *c* would reduce the accuracy significantly.

After many rounds of experiments, the final values used were as follows:

	Count Vectorizer	Tf-idf Vectorizer
Logistic Regression	0.25	100
SVM	0.5	5

Table 1- Final values for *C*

These values were also used in the stacking classifier, though the stacking classifier did not do better than our other classifiers. This may be due to the fact that the classifiers present similar results.

5. Final Result

With the conclusion of the experiments, it became apparent that the CountVectorizer paired best with the Logistic Regression classifier while the Tf-idf Vectorizer paired best with the SVC with linear kernel.

When determining the model bias and variance, it is good to first visualise the distribution of the classes.

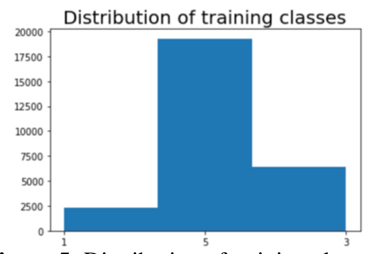

Figure 5- Distribution of training classes.

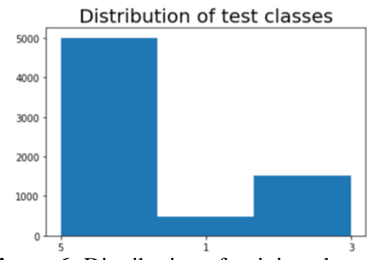

Figure 6- Distribution of training classes.

As shown above, one can see that the models do a decent job at maintaining the distribution of the classes, only being very slightly biased towards the majority class.

The average cross validation accuracy of the models on the training set did not vary much from the final accuracy of the model on unseen data. This is highly likely due to the train-test split being of an appropriate number, in accordance to the Inductive Learning Hypothesis, as well as the implementation of the cross-validation strategy. In the experiments above, a split of 75-25 was used. This leads to the evaluation bias and variance being significantly reduced.

6. Conclusions

In conclusion, the best pair that worked was the Tf-idf Vectorizer with the SVM model with linear kernel. That said, the resultant accuracies of each pair did not vary much.

7. References

Ceballos, F., 2019. Stacking Classifiers for Higher Predictive Performance. [online] Medium. Available at < https://towardsdatascience.com/stacking-classifiers-for-higher-predictive-performance-566f963e4840 >.

Grootendorst, M., 2019. Stacking made easy with Sklearn. [online] Medium. Available at < https://towardsdatascience.com/stacking-made-easy-with-sklearn-e27a0793c92b >.

Gupta, P., 2017. Regularization in Machine Learning. [online] Medium. Available at < https://towardsdatascience.com/regularization-in-machine-learning-76441ddcf99a >.

Gupta, S., 2018. Sentiment Analysis: Concept, Analysis and Applications. [online] Medium. Available at <https://towardsdatascience.com/sentiment-analysis-concept-analysis-and-applications-6c94d6f58c17 >.

Kub, A., 2018. Sentiment Analysis with Python (Part1). [online] Medium. Available at <https://towardsdatascience.com/sentiment-analysis-with-python-part-1-5ce197074184>.

Kub, A., 2019. Sentiment Analysis with Python (Part2). [online] Medium. Available at <https://towardsdatascience.com/sentiment-analysis-with-python-part-2-4f71e7bde59a>.

Mukherjee, A., Venkataraman, V., Liu, B. & Glance, N. What Yelp fake review filter might be doing? 7th

International AAAI Conference on Weblogs and Social Media, 2013.

Rayana, S. & Akoglu, L. Collective opinion spam detection: Bridging review networks and metadata. Proceedings of the 21th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining, 2015. 985-994.

Yse, D. L., 2019. Your Guide to Natural Language Processing (NLP). [online] Medium. Available at <https://towardsdatascience.com/your-guide-to-natural-language-processing-nlp-48ea2511f6e1 >.