COM3110 Sentiment Analysis Report - Alexander Ganvir

1 Implementation

A Naive Bayes Model has been implemented in order to perform sentiment analysis on a Rotten Tomatoes movie review dataset. Four Models have been developed, two each for a 3-class & 5-class sentiment analysis, firstly by using all the words from the dataset, and then contrastingly by use of specific selected features.

1.1 Preprocessing

Before classification, we preprocess all phrases across the dataset in the functions ***preprocess_phrase*** (preprocesses a single phrase) and ***preprocess_phrases_for_data*** (applies *preprocess_phrase* across the dataset) - it takes a phrase and removes stopwords, using the NLTK library's stoplist; then removes punctuation, keeping hyphenated words and exclamation marks; and then adds a negation prefix to words preceded by 'not' or 'never'. This function is applied to each phrase in the dataset, and then a list of all the words in the dataset is created from this. Each phrase in the dataset is also filtered for the features we are using, this is done for the training and dev/test set, depending on which we are testing, this is done in the functions ***apply_features_to_phrase*** (processes a single phrase) and ***process_phrases_for_feature*** (applies *apply_features_to_phrase* across the dataset).

1.2 Prior Probability Calculation

After preprocessing and feature selection, we compute the prior probabilities for each sentiment class, this is done by counting in the training set the total number phrases associated with a sentiment, divided by the total number of phrases. These prior probabilities are stored in a dictionary, for use when calculating the posterior probability. This is computed in the function ***calculate_prior_probabilties*** - it takes the number of classes as well as the training data.

1.3 Likelihoods Calculation

To calculate the likelihoods of each feature for each sentiment, we count the times a feature is associated with the sentiment and divide this by the total amount of features associated with the sentiment, and create a dictionary of the likelihood of each feature for the sentiment, this is done in the function ***calculate_likelihood_for_sentiment***. Then in the function ***get_likelihood_product_for_phrase*** we check the likelihood of each word in the phrase and calculate the product sum of the likelihoods. Then to calculate the posterior probability for that sentiment given the phrase we multiply the product of the likelihoods by the prior probability for that sentiment, in the function ***calculate_post_probabilty***, then checking this posterior probability for each sentiment to estimate the sentiment classification of the phrase, this is done in the function ***calculate_estimate_sentiment***. Laplace Smoothing is implemented in ***calculate_likelihood_for_sentiment***, by adding one to the count and adding the size of the vocabulary of the features to the total amount of features associated with the sentiment.

1.4 Feature Extraction

Differing features can be used through a series of if statements in the ***main*** function, the first of these is part-of-speech tagging using the NLTK library, this allows features to be either adjectives, adverbs or verbs from the training set, and can be used together or separately. Then using the NLTK library for VADER we set features as those regarded as polarising words, either negative or positive. And then through the SCIPY stats library Chi-Square Tests were also implemented in order to find the most significant words from the training set when it came to sentiment analysis.

1.5 Macro F1 Score

The Macro F1 Score was implemented in the function ***compare_estimate_sentiment_to_real***, we create a dictionary for the F1 scores and the Macro F1 scores, comparing the real and estimated sentiment scores to analyse the effectiveness of the solution with True Positives (TP), False Positives (FP), True Negatives (TN) and False Negatives (FN) for each sentiment class, the occurrences of these are counted for each sentiment allowing us to use these to build confusion matrices to further analyse, and using the following formula to create the F1 Macro Score:

- **macro-$F1$ score**, i.e. the mean of the class-wise $F1$-scores:

$$\text{macro–}F1 = \frac{1}{N}\sum_{i=0}^{N} F1\text{–score}_i$$

where N is the number of classes. $F1$–score is calculated for each class i:

$$F1\text{–score}_i = \frac{2 * \text{Precision}_i * \text{Recall}_i}{\text{Precision}_i + \text{Recall}_i} = \frac{2 * TP_i}{2 * TP_i + FP_i + FN_i}$$

2 Feature selection

The main idea throughout the feature selection process was to create subsets of all the words in the set by narrowing down the set to features that are most likely to carry sentiment information.

First part-of-speech tagging was used for adjectives, verbs, adverbs, and superlative adjectives. Features here were decided upon to capture elements that contribute more significantly to sentiment expression, the idea that by isolating adjectives the chosen features would directly convey positive or negative sentiments more so than the whole word list; similarly verbs and adverbs may indicate the intensity of an action, providing additional context, viewing superlative adjectives may provide a deeper insight into the polarity of adjectives. However, it is possible that by only focusing on these types of words that the broader context of the sentiment is lost, where sentiment-bearing words may not be straightforward adjectives or verbs, this is especially true for idiomatic expressions. To create this subset, the NLTK library was used, extracting the chosen type of words from all the words in the dataset.

Secondly polarity of words was considered, through the NLTK library's pretrained sentiment analyzer VADER, words with a high or low polarity score were considered, and used as features, the idea is that these words are likely to demonstrate a strong sentiment, as well as capture the intensity of the sentiment, however VADER is best suited for language used in social media and may be less accurate when rating longer, structured sentences such as movie reviews.

The final method of features implemented was created using Chi Square Tests, with the SciPy.stats library; it works by finding the words that are most statistically dependent on sentiment classes, thus indicating a significant association with a particular sentimen, the words returned as most significant on the training set are then used as features. Whilst effective it is sensitive to the frequency of terms - rare words, even if highly informative, may not achieve statistical significance due to their infrequent occurrence in the dataset; as well as this the intensity of the words in not considered in Chi Square Tests, and there then may be an inability to capture the intensity of sentiment within a phrase.

Through exhaustive analysis, it was found that using part-of-speech tagging, polarity analysis and chi-square tests in conjunction with each other was the most effective method of sentiment analysis.

3 Results and Discussion

Comparing F1 Scores and Confusion Matrices between the selected features and all the words in the dataset as features, has given these results for the 3-class and 5-class sentiment analysis:

Confusion matrices:

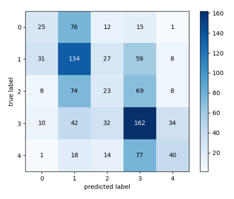


Figure 1: 5-Class All Words

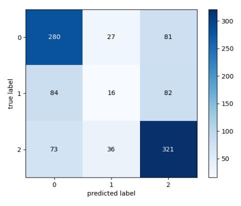


Figure 2: 3-Class All Words

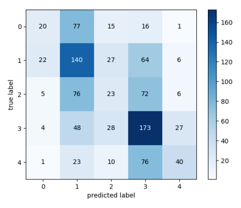


Figure 3: 5-Class Features

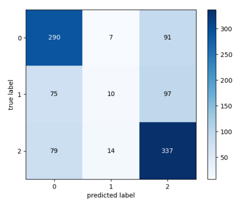
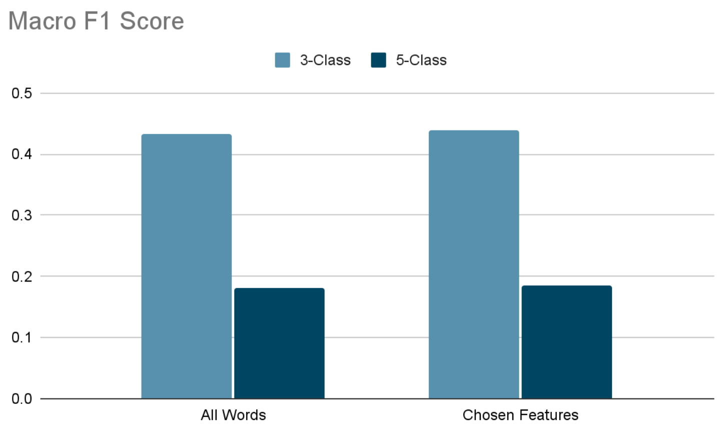


Figure 4: 3-Class Features

Macro F1 Score:
3-Class All Words : **0.432433**
5 Class All Words : **0.181470**
3-Class Features : **0.439076**
5 Class Features : **0.186159**

Time (Seconds):
3-Class All Words : **11.19**
5 Class All Words : **10.69**
3-Class Features : **141.64**
5 Class Features : **147.10**



Discussion:
We can see from the results that both all words and the selected features have a similar macro F1 score as well as a similar spread in the confusion matrix, the selected feature does very slightly edge in macro F1 score however it takes considerably longer to run for the selected features, due to the time complexity of the Chi-Square Test. The Similarity between these could be due to redundancy or overlap between the selected features and the words present in the all words approach. Due to superior time complexity, and minimal difference in F1 performance it is decided to use the All Words model as the best model trained.

4 Error Analysis

In order to analyse the errors of the designed model, four examples have been chosen where errors exist in the sentiment analysis, we will analyse the preprocessed phrase.

Class Tested: 5, Actual Sentiment = 4, Estimated Sentiment = 1: *"welcome lack pretension film simply sets entertain ends delivering good measure"* - an extreme example, this is likely to have occurred as the model does not negate for the word 'lack', this has meant that 'pretension' is most probably given the phrase a negative weight, as has the adverb simply. Adding in additional negation words could prevent this.

Class Tested: 3, Actual Sentiment = 2, Estimated Sentiment = 0: "*unfolds series achronological vignettes whose cumulative effect chilling"* - another example where the model estimates negatively, this time on a 3-class example, this is likely to have occurred due to rare descriptive words that are not common across the dataset; as well as a few words such as "achronological" and "chilling" that when taken out of context could be classified as negative.

Class Tested: 3, Actual Sentiment = 0, Estimated Sentiment = 1: *"irritates saddens martin lawrence latest vehicle explode obnoxiously screens something bubba ho-tep clearly evident quality may end languishing shelf somewhere"* - an example where the sentiment is clearly negative but has been estimated to be neutral in the 3-class example, demonstrating the somewhat lack of context that a smaller range of classes can give in potentially complex phrases such as this.

Class Tested: 5, Actual Sentiment = 0, Estimated Sentiment = 3: *"lazy filmmaking director taking hands-off approach shaped story show us compelling"* - words such as compelling seem to have overridden lazy in this example, using a sentiment lexicon could potentially improve the model and give words more semantic intensity.