University of Sheffield

Sentiment Analysis Report

Andreas Evripidou

COM3110: Text Processing

Department of Computer Science

December 15, 2023

1 Introduction

The project aimed to implement a multi-nominal Naive Bayes model for sentiment analysis of movie reviews, as described in chapter 4 of Speech and Language Processing book [1]. The program operates within a specific configuration framework, offering four potential configurations. These configurations result from employing either three or five sentiment classes and utilizing all words from the datasets or conducting feature extraction.

The input data includes three datasets: training, development, and testing. Throughout the project, it was necessary to process the data by applying any required preprocessing steps. After the pre-processing step, the model was trained using the training dataset, and its performance was assessed by calculating a macro F1 score through the classification and evaluation of the development dataset. Additionally, the test data underwent classification, although evaluation was not conducted.

For feature extraction, stopwords and lamination techniques from the Natural Language Toolkit (nltk) were employed independently and in conjunction with a self-created list of features created by three different techniques.

2 Implementation

The Naive Bayes model implementation is split into four distinct classes, each assigned specific tasks. These classes, namely Preprocessor, Feature_selector, Classifier, and Evaluator, are explained below. The complete implementation of the model is located in the NB_sentiment_analyser.py file from where the mentioned classes are used.

The initial phase involves data preprocessing within the Preprocess class. Before preprocessing, if the configuration is set to use three classes, the preprocessor class is also used to map their sentiment from a 0-4 to a 0-2 scale. All reviews undergo conversion to lowercase, contractions such as "isn't" are expanded to their original forms (e.g., "is not"), as well as numbers and special characters are removed, and finally, the reviews are tokenised using the nltk toolkit word_tockenize method. If the configuration is set to use feature selection (features option) and not all the features (all_words option), the tokens in the reviews are filtered out of stop words (using nltk's corpus stop words) and Lemmatized (using nltk's WordNetLemmatizer) to group variant forms of the same word.

The given NB_sentiment_analyser was extended to include arguments for which feature selection method to be used (["most_common", "count_difference", "chi_square"],) and how many features to be used. The Feature_Selector class creates a feature list based on the configured feature selection method and the length of the feature list. Both train and development data are used to create the feature list. Feature_Selector is also responsible for filtering out the words in the reviews data set that are not included in the feature list.

In the Classifier class during the training stage, the training reviews are categorized into their sentiment classes (5 or 3 classes), and a list of dictionaries for each sentiment class maps each word to their frequency. These elements are crucial for calculating priors and likelihoods. The prior probabilities for each class are computed by dividing the number of reviews in a given sentiment class by the total number of reviews. These probabilities are then stored in a list associating each sentiment class with its respective probabilities. The final training step involves computing the likelihood for each word in each class. This is achieved by dividing the frequency of a word's presence in a given sentiment class by the total number of words in that class. For better results, Laplace smoothing is used by adding one in the numerator and adding the number of unique words in the training dataset to the denominator. The results are stored in a list of dictionaries. Each entry for the list is a dictionary of a sentiment class, mapping each of its words to their likelihood probabilities.

The Classifier class also computes posterior probabilities for each review (development and test reviews) and classifies each review into the sentiment class with the highest posterior probability. The predict_class takes a review that calculates the posterior probabilities of each class using the pre-computed prior probabilities and likelihoods and returns the class with the highest probability. Additionally, this class offers an option to save predictions for development and test reviews, facilitated by a dedicated function.

The final class, Evaluate, generates the confusion matrix and calculates the macro F1 score for the development reviews. Furthermore, if the option for a confusion matrix is specified during program execution, this class handles the plotting of the confusion matrix.

3 Feature Selection

As previously stated, two methods were employed for feature selection to discern the optimal features within the test and development datasets. The methods are taken from "Efficient Feature Selection Techniques for Sentiment Analysis" by Avinash et al.[2] and developed from scratch.

3.1 Count Words

This approach involves quantifying the occurrence of individual words in the dataset. The features are selected based on their frequency, with more frequently occurring words considered as potentially significant indicators. Features with higher scores, i.e., more frequent occurrence across classes, are considered for selection.

3.2 Chi Square

The Chi-Square statistic for each word i can be computed as:

$$\chi^2(i) = \sum_j \frac{(n_{ij} - e_{ij})^2}{e_{ij}} \tag{1}$$

Where e_{ij} is the expected frequency of word i in class j, calculated as:

$$e_{ij} = \frac{\text{total occurrences of word } i \times \text{total reviews in class } j}{\text{total reviews}} \tag{2}$$

A higher Chi-Square value indicates a stronger association between the word and the sentiment class, making it a more valuable feature.

4 Results

For experimenting with the different configurations, the experiment.py script was developed to run NB_sentiment_analyser.py. The script can be found in the experiment sub-folder along with a csv file containing the results. Table 1 shows the best-performing models with different feature selection techniques, and Figure 1 shows the confusion matrices of the best-performing model for each set of classes.

Classes	Features	All Words	Word Count	Chi Square
3	1500	0.505625	0.510759	0.536531
5	1000	0.325937	0.323556	0.361067

Table 1: Macro F1 reults

		Predicted				
		Ne	SNe	Nu	SP	P
Actual	Ne	28	69	10	18	4
	SNe	41	117	26	65	10
	Nu	8	65	26	74	9
	SP	14	41	34	153	38
	P	2	16	13	71	48

(a) Confusion Matrix (5 Classes, 1000 features, Chi Square)

		Predicted		
		Ne	Nu	P
Actual	Ne	227	89	22
	Nu	85	312	33
	P	81	83	18

(b) Confusion Matrix (3 Classes, 1500 features, Chi Square)

Figure 1: Confusion Matrices

5 Discussion Error analysis

5.1 Discussion

When using all the words as features, most of the predictions depend on the prior probabilities since there is at least one word that is not present in all the classes in most cases. When using specific features instead of all the words, there is a substantial increase in the macro F1 scores for both 3 and 5 classes. The reason for this is that less predictions are dependent on the prior probabilities. This happens since words that appear most frequently in such reviews (training data) are considered only, so they are more likely to be present in the dev reviews, thus avoiding the multiplication by only the Laplace smoothing when a word is not present. Also, some popular words overlap in all the classes, so this multiplication occurs less frequently. For example, when using the 1000 most popular words for the 3 classes configuration, the total number of unique words obtained is 1742, which means more than one-third of the features are shared between negative, neutral and positive classes. In the two confusion matrices, we can still see that classes with higher prior in the train set are still more likely to be classified, but the effect has been decreased. Also, when not using all the features, a stop list removes words we know have no semantic value.

The results clearly show that the Chi-Square metric works better than just the frequency of the words. When using Chi-Square with three classes and 1500 features, the macro f1 score increases by roughly 3% from using all words and 2.5% from using Word Count for the selection of features. This must be due to the fact that the expected frequency makes an association between a word and a sentiment class in contrast with just the plain frequency of the word in the Count Words method.

When using three classes instead of 5 (with both all words and features configuration), the scores are better since there is less margin of error. There are fewer choices to classify the reviews in, which means fewer sentiments are incorrectly classified. The probability to classify correctly a review is $\frac{1}{25}$ when using five classes, and it increases to $\frac{1}{9}$ when using just three classes.

5.2 Error Analysis

From the result, we can see that the system has a lot of room for improvement. For example, in the review "Not the best Herzog perhaps, but unmistakably Herzog." neutral was classified as Positive when using three classes and somewhat positive with five classes. This is due to the system not handling negation. To improve the model we can add a pre-processing step which companies the negation with the next word so instead of having "best", to have "not_best" as a feature.

Another example is the review "It bites hard" which is again neutral and it was classified as Somewhat Negative when using five classes and Negative when using three. This is because the classification is based on individual tokens thus the classifier only considers individual words and not phrases. To take into consideration phrases, we can use knowledge engineering and contract a lexicon of phrases. Then the system should replace the phrases with a token; in this case, change the tokens "bites" and "hard" with "bites_hard"

The phrase "Catch it ... if you can!" refers to the movie "Catch Me if You Can", and since it shows enthusiasm with the exclamation mark, it is Somewhat Positive. The phrase was classified as Somewhat Negative by the five-class model and Neutral by the three-class model. The exclamation mark captures the sentiment in the phrase, but since we remove all special characters in pre-processing, the sentiment is lost. To avoid this, the pre-processor should not remove exclamation marks.

The last example of an error phrase is "A well-crafted letdown," a somewhat negative review that uses sarcasm to express the sentiment. The phrase was classified as Somewhat Positive and Natural. Since sarcasm can not be detected by pre-processing, the system will need to use a Natural Language Processing technique to identify the sarcasm. Thus, the Naive Bayes Classifier can not be improved to reduce the error caused by sarcasm.

Bibliography

[1] D. Jurafsky and J. H. Martin, *Speech and language processing : an introduction to natural language processing, computational linguistics, and speech recognition.* Upper Saddle River, N.J.: Pearson Prentice Hall, 2009. [Online]. Available: http://www.amazon.com/Speech-Language-Processing-2nd-Edition/dp/0131873210/ref=pd$_b$$xgy_{bi}$$mg_y$

[2] A. Madasu and S. E, "Efficient feature selection techniques for sentiment analysis," 2020.