Sentiment Analysis using Unigrams and Bigrams

Urvashi Uberoy
Department of Computer Science
Princeton University
uuberoy@princeton.edu

Divyanshu Pachisia
Department of Mechanical Engineering
Princeton University
divyanshupachisia@princeton.edu

Abstract

The classification of reviews into positive and negative classes is important because it lets us filter and sort based on the sentiment associated with a product or experience. In this assignment, we train different classifiers on 2400 product reviews and evaluate the performance of the classifier on a testing set with 600 product reviews. After training the classifiers on both unigram and bigram representations of each review, with and without feature selection, we find that the bigram representation results in a higher accuracy score, with an average increase in accuracy of 13.5% across classifiers. While the bigrams representation of the reviews initially increases the complexity of our model, we show that feature reduction techniques can be used to reduce this bigram model to the same complexity as the unigram one without sacrificing the increased accuracy.

1 Introduction

To perform sentiment analysis on a review we are first faced with the fundamental question about how to extract features from the data. One common approach is to extract unigrams, where individual words in the reviews are extracted and tallied. Another approach is to instead extract bigrams,[1] i.e., pairs of words that are appear next to each other, and tally them for each review. In this assignment, we compare the unigrams and the bigrams representations based on their performance in classifying reviews, while also keeping in mind the number of features required (model complexity) for the performance.

2 Related Work

The classification of reviews based on sentiment is a well-studied task. The inclusion of bigrams in this paper is inspired by Wang & Manning (2012), where they show that bigrams enhance the performance of classification. [7] To counter the additional model complexity that bigrams add (in terms of number of features), we use common feature selection techniques that are available in SciKitLearn. This past work allows us to investigate whether a feature-reduced and thus, a complexity-reduced bigram model still performs better than the unigram one.

3 Methods

3.1 Initial Data Exploration

While Wang & Manning showed that the bigrams increased performance on their dataset, we wanted to gain intuition about whether it would have the same effect on ours. Figure 1 below shows that while unigrams, such as "problem," are ambiguous in their sentiment (50-50 split in positive and negative reviews), a bigrams representation, such as "no problem" has a clearer sentiment (mostly positive reviews). This indicates that the bigrams representation is a valuable path to pursue.

3.2 Data processing: Cleaning the raw data to extract feature set

The preprocessor.py script given to us explored three main cleaning techniques:

[1] Note that in our bigrams representation, we include unigrams as well as bigrams, so hereafter "bigrams" will refer to a combination of unigrams and bigrams.

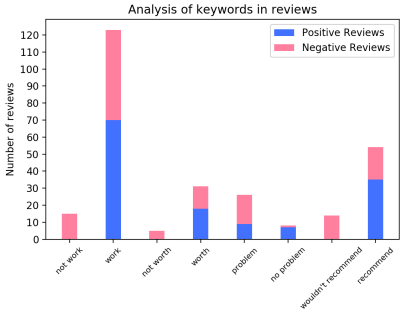

Figure 1: Sentiments associated with Unigrams and Bigrams

- Removing stop words using NLTK's list of stopwords in the English language, which removes words like 'the,' 'no,' and 'very' that do not contribute to the sentiment of a review.
- Stemming using NLTK's Porter Stemmer, which reduces words to their stem. For instance, "happiness" and "happy" would both be reduced to "happi."
- Lemmatizing using NLTK's WordNet Lemmatizer. This reduces each word to its base form, so "are" would be reduced to its infinitive: "be."

All these techniques were used in combination to identify 541 key tokens for the unigrams representation of the data. For the bigram approach, however, some stopwords, such as "no", were included because they are used in conjunction with other words, such as "problem", to add meaning. To produce the list of 4387 bigrams, we used SciKitLearn's CountVectorizer function with the parameter "ngram_range" set to (1, 2).

3.3 Feature Selection: Reducing the dimension of the feature set for model simplification

We used three different feature selection methods from the SciKitLearn Python Libraries. [2] [5]

1. *Principal Component Analysis* (PCA): Uses Singular Value Decomposition to find linear combinations of features to lower dimensionality while maintaining variance. This is a class-independent feature selection method.
2. *Linear Discriminant Analysis* (LDA): Finds the line that best separates our two classes and then projects each data point onto this line, reducing the dimension to one. This doesn't work when the features are co-linear so PCA was used prior to LDA.
3. *Chi-Square Test*: Uses the Chi-Square test between features and the classes to score the level of dependency. The 200 features with the highest scores were kept.

3.4 Classification Methods

We used four different classification methods from the SciKitLearn Python Libraries. [2] [5]

1. *Logistic regression with ℓ_2 penalty* (LR): Described in 3.5

   Hyperparameter: C (inverse of regularization strength)
2. *Support vector machine* (SVM): Finds the separating line/hyperplane between data from two classes to maximize the margin between them

   Hyperparameters: C (penalty parameter to prevent overfitting), kernel type (nature of decision boundary), Gamma (kernel coefficient)
3. *Multinomial Naive Bayes classifier* (NB):

   Hyperparameters: Alpha (smoothing parameter in the form of a pseudocount)
4. *Random Forests* (RF):

   Hyperparameters: Maximum Depth (of the tree), n-estimators (maximum trees)

Instead of using the default parameters for each classifier, we used SciKitLearn's GridSearchCV function to tune hyperparameters. This function iterates through different values for each hyperparameter and returns the parameters that produce the highest accuracy score.

3.5 One Method in Detail: Logistic Regression [1]

The basic premise underlying logistic regression is finding a set of weights corresponding to the set of features in order to minimize the error in the classification. To understand the mechanics of this technique, we first need to define the following variables.

- $\mathbf{X} = \{\mathbf{x_n}\}_{n=1}^{N}$, where $\mathbf{X}$ denotes the entire body of training data with dimension N×D, and $\mathbf{x_n}$ denotes a single data point with dimension D×1.
- $\mathbf{Y} = \{\mathbf{y_n}\}_{n=1}^{N}$, where $\mathbf{Y}$ denotes the training labels or classes with dimension N×1, and $\mathbf{y_n}$ denotes the class of the nth data point.
- $\mathbf{w} = \{\mathbf{w_d}\}_{d=1}^{D}$, where $\mathbf{w}$ denotes the weight vector with dimension D×1 which has a value (or weight) corresponding to each feature. In the case of binary classification, negative weights are assigned to features associated with class 0, while positive weights are assigned to features associated with class 1. More about how the weights are computed will follow below.

We also define the *logistic function* $\sigma(z)$ and use it to form our predictions ($\hat{y}$).

$$\sigma(z) = \frac{1}{1+e^{-z}} \qquad \hat{y} = \sigma(\mathbf{w}^T\mathbf{x}_n)$$

Here, we see how the weights corresponding to the features determine our prediction - negative weights drive the prediction to zero while positive weights drive the prediction to 1, given that $\mathbf{x}_n$ only contains positive values. Therefore, features with negative weights should correlate to a negative sentiment (y_n = 0) and vice versa. The method to determine weights is rooted in the probability of predicting the actual value of y_n given the weights, $\mathbf{w}$ and data, $\mathbf{x}_n$. We take the log of this probability and see that the gradient of this log probability gives us an intuitive error function:

$$\Pr(y_n|\mathbf{w},\mathbf{x}_n) = \sigma(\mathbf{w}^T\mathbf{x}_n)^{y_n} \cdot (1-\sigma(\mathbf{w}^T\mathbf{x}_n))^{1-y_n}$$

$$\nabla_{\mathbf{w}}(\log \Pr(y_n|\mathbf{w},\mathbf{x}_n)) = \mathbf{x}_n(y_n - \sigma(\mathbf{w}^T\mathbf{x}_n)) = \mathbf{x}_n \cdot \underbrace{(y_n - \hat{y}_n)}_{\text{error in prediction}}$$

Through a method called *Stochastic Gradient Descent* we find the weights to minimize this error function. The algorithm used to do this is shown below:

$$\mathbf{w}^{(t+1)} \leftarrow \mathbf{w}^{(t)} + \alpha\mathbf{x}_n(y_n - \sigma(\mathbf{w}^{(t)T}\mathbf{x}_n))$$

In each iteration, the value of n is drawn from a uniform distribution across $1 \ldots N$, leading to a less computationally-intensive algorithm than full gradient descent. We must also consider the rate at which we perform the gradient descent (referred to as the learning rate, α) and the danger of over-fitting the weights to the training data. In order to correct for this we scale the change in prediction by α and add a regularization penalty, λ. The value of α is usually between 0 and 1 which reduces the rate at which we change the weights and λ is set to ensure the weights do not become too large or small and result in over-fitting. We modify our algorithm to find the weights to include these two factors in the form shown below.

$$\mathbf{w}^{(t+1)} \leftarrow \mathbf{w}^{(t)} + \alpha\Big(\mathbf{x}_n(y_n - \sigma(\mathbf{w}^{(t)T}\mathbf{x}_n)) - \frac{\lambda}{N}\mathbf{w}^{(t)T}\Big)$$

With this final algorithm, we first fit a logistic regression model on the training data to find the weights for each feature that minimizes the classification error, while guarding against overfitting. We tune the hyperparameter C (C = 1/λ), using SciKitLearn's GridSearch CV which cross-validates on the training set to find an optimal C. We then apply these weights to the testing set, to output our predictions for the testing set. Despite having the ability to choose and tune C, overfitting is still a concern in logistic regression as C is tuned manually which may result in some overfitting.

There are also some key assumptions that logistic regression relies on that must be taken into account. It assumes that each data point is independent of the others which may not be the case - for example, if one person writes a series of reviews on the same product and they became more dissatisfied over time. Additionally, it works best only if all the relevant independent variables are in the model. In the case of reviews, feature reduction may take away some key independent variables and some features like punctuation may not even be taken into account as relevant variables.

4 Results

Table 1 shows some key words in reviews and their associated metrics that will be discussed in the next section. Table 2 shows the accuracy and F1-Score of the classifiers on both the unigrams and bigrams. Figure 2 displays the ROC curves for the classifiers with no feature reduction.

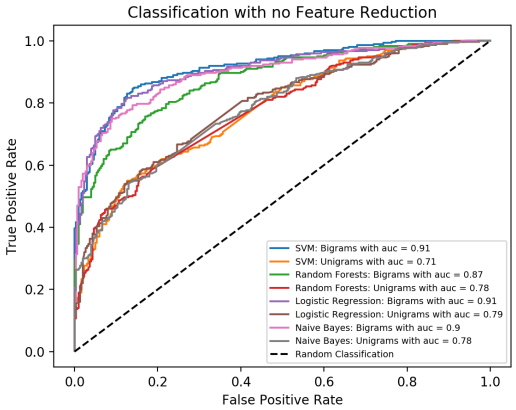

Figure 2: ROC Curve with no feature reduction

Key Word(s)		"worst"	"not good"	"good place" ... "garbage"
Prediction Accuracy	Unigram	Accurate	Not Accurate	Not Accurate
	Bigram	Accurate	Accurate	Not Accurate
LR Weight	Unigram	-2.01	1.447	0.783
	Bigram	-1.24	-0.792	0.892
LDA Feature Value	Unigram	-0.25	0.28	0.68
	Bigram	-1.86	-1.8	0.26

Table 1: Analysis of LR weights and LDA features with specific keywords in reviews

	Unigrams				Bigrams			
	LR	SVM	NB	RF	LR	SVM	NB	RF
No Feature Reduction	0.7033	0.71	0.6966	0.6583	0.8433	0.8417	0.8267	0.7783
	0.775	0.7658	0.7675	0.7192	0.8092	0.8129	0.8188	0.7617
	0.7	0.7	0.69	0.62	0.84	0.84	0.83	0.76
PCA	0.695	0.6833	-	0.6317	0.7983	0.8033	-	0.6967
	0.767	0.7633	-	0.6542	0.7783	0.7654	-	0.6821
	0.69	0.67	-	0.62	0.8	0.8	-	0.7
PCA then LDA	0.6833	0.685	-	0.6867	0.8167	0.815	-	0.8217
	0.8412	0.8383	-	0.8379	0.9417	0.9429	-	0.9404
	0.68	0.67	-	0.68	0.82	0.81	-	0.82
Chi-Square Test	0.7845	0.77	0.786	0.7083	0.7983	0.7967	0.795	0.7567
	0.8	0.76	0.79	0.7	0.8	0.7838	0.8	0.76
	0.784	0.7363	0.7945	0.685	0.8	0.8	0.8075	0.7197

Accuracy Score | Cross-Validated Average Accuracy Score | F1 Score

Table 2: Classification Results

5 Discussion and Conclusion

The results show that bigrams perform consistently better than the unigrams at the classification tasks, based on the accuracy (13.5% higher), F1-Score and the area under the ROC curve (AUC). This is evident from the ROC Curves in Figure 2 where there is a clear separation between bigrams and unigrams. This enhanced performance is due to a number of factors including:

1. Some sentiments are better represented with bigrams than unigrams as shown in Figure 1. The logistic regression weight corresponding to ”not good” (Table 1) drives home this point. For unigrams, the average weight of ”not and ”good” is 1.447. Since ”good” has a higher weight than ”not”, this leads to misclassification. However, the bigram weight is adjusted to be ”-0.792” which correctly predicts the negative sentiment.

2. The bigrams representation encodes the dependence between separate unigrams. This is especially useful to the Naive Bayes classifier, since the classifier assumes independence between features.

However, it is important to note that there are cases where both models predict incorrectly, as shown in Table 1. The word "garbage" does not appear in the training set and so the prediction for neither method is accurate. Additionally, the better performance of the bigrams could be because it has 8 times more features (4387 bigrams) than the unigrams model (541 unigrams). We can imagine a situation where this increased model complexity may not be worth the additional performance. Therefore, to compare the representations more robustly we compare their performance after feature selection.

5.1 Classification after PCA

This method linearly combines features to form new ones based on their dependency, making it difficult to interpret their meaning. Since PCA is class independent, it does not suffer from overfitting, as evidenced by the cross-validated scores being close to the accuracy scores. Additionally, applying PCA to both models to reduce the number of features to around 300 produced better results for bigrams, indicating that the bigrams model can be reduced in complexity while maintaining performance.

5.2 Classification after PCA and LDA

With this feature reduction, each review is represented as one number based its projection on the line that best separates the classes, which is plotted in Figure 3 for the bigrams. From the separation in classes shown in the figure as well as the higher cross-validated accuracy score, we see that LDA suffers from overfitting. Marron, Todd and Ahn, in their paper *Distance Weighted Discrimination*, describe "data piling," a phenomenon where multiple points are projected onto identical values which is observed in our case. However, despite this overfitting and "data-piling" it is remarkable that the drastic feature reduction to one feature only lowers the accuracy by $\sim$2% for NB and SVM and even increases it with RF. Additionally, with this simplified model complexity the bigrams model performs much better than the unigrams one, which makes a case for bigram representation.

5.3 Classification after Chi-Square Test

Figure 4 below highlights the 25 Bigrams with the highest p-values after the Chi-Square Test which tests dependency between feature and class. While most of the words on the list are unigrams that convey strong sentiment, such as "great" and "bad," we also have bigrams like "work great" and "the worst." Nevertheless, the high density of unigrams in top 25 bigrams explains why, after selecting the top 200 unigrams and bigrams, the classifiers for each representation performed with near-equal accuracy (Table 1).

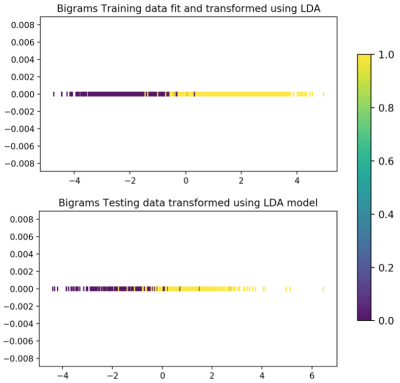

Figure 3: Feature Reduction Using Linear Discriminant Analysis [6]

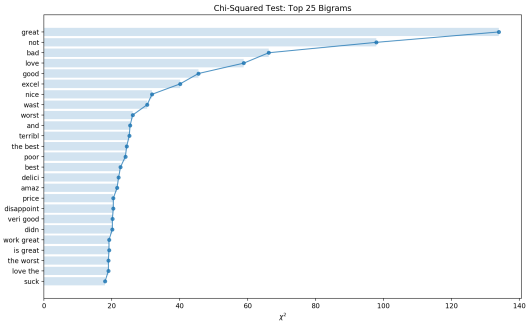

Figure 4: Chi-Square Test showing the top 25 Bigrams [3]

5.4 Concluding Remarks

We have shown that across classifiers, the bigrams representation performs better than the unigrams representation. This increased performance is maintained, despite feature (model complexity) reduction, especially through LDA which reduces each data point to just one feature. However, there are still concerns about overfitting, data-piling and reviews where both representations classify incorrectly. In the future, this approach could be applied to larger data sets, features like emojis and punctuation could be included and unsupervised learning techniques could be explored.

Acknowledgments

Professor Barbara Engelhardt
Jonathan Lu
Diana Cai
The SciKit Learn library which was used for all the code for this assignment

References

[1] Adams, Ryan. "COS 324: Week 5: Logistic Regression." Computer Science 324, 9th October 2018, Princeton University.

[2] Buitinck et. al., "API design for machine learning software: experiences from the scikit-learn project." ECML PKDD Workshop: Languages for Data Mining and Machine Learning, 2013

[3] Kim, Ricky. Another Twitter Sentiment Analysis with Python–Part 8 (Dimensionality Reduction: Chi2, PCA). Towards Data Science, 25 Jan. 2018, towardsdatascience.com/another-twitter-sentiment-analysis-with-python-part-8-dimensionality-reduction-chi2-pca-c6d06fb3fcf3.

[4] Marron, J. S., et al. Distance-Weighted Discrimination. Journal of the American Statistical Association, vol. 102, no. 480, 2007, pp. 12671271. JSTOR, www.jstor.org/stable/27639976.

[5] Pedregosa et. al., "Scikit-learn: Machine Learning in Python" Journal of Machine Learning Research, 2011

[6] Stack Overflow, Binary Classification Using radial basis kernel SVM with a single feature, https://stats.stackexchange.com/questions/86458/binary-classification-using-radial-basis-kernel-svm-with-a-single-feature

[7] Wang, Sida and Manning, Christopher D. Baselines and bigrams: Simple, good sentiment and topic classification. In Proceedings of the 50th Annual Meeting of the Association for Computational Linguistics: Short Papers-Volume 2, pp. 9094. Association for Computational Linguistics, 2012.