Probabilistic vs Deterministic Short Text Sentiment Classifiers

Ramaneek Gill
Department of Computer Science
University of Toronto
ramaneek.gill@mail.utoronto.ca

Alexandros Tagtalenidis
Department of Computer Science
University of Toronto
alex.tagtalenidis@mail.utoronto.ca

Abstract

We investigate how the independence assumptions made by Bayesian Networks influence the precision and recall of a short text sentiment classifier when compared to the Gaussian class-conditional density of a Linear Discriminant Analysis model. We then compare the results of the probabilistic models to common deterministic models to see if generative algorithms outperform deterministic models with the task of classifying positive or negative sentiment. Our investigations show that the naive independence assumptions made by the Bayesian network among the presence of words in a short text are only minimally disadvantageous when compared the LDA model, when using a small amount of data. Further research also shows that generative models vastly outperform deterministic models when using the area under the precision-recall curve as a metric for model strength.

1 Introduction

Twitter is an online social networking service that allows users to create short texts (tweets) about any topic they desire. These short texts are limited to 140 unicode characters. We purpose several machine learning methods for classifying the sentiment of a tweet. We then investigate the strength of these models in order to determine whether the assumptions made by each of the models are advantageous and give them an edge over the others.

Sentiment analysis on short texts is a very interesting problem because of the lack of data available for a model to extract high quality features. In a tweet which contains a hard limit of 140 characters the user cannot fit many words, which results in a limited amount of features for a machine learning model to base the sentiment classification on. There is also the issue of noise. Many users use emoticons or misspell words or repeat certain characters extensively. A combination of noise and limited data can produce interesting problems for many sentiment classifiers.Throughout this paper we refer to a short text's sentiment. In our research we only predict positive or negative sentiment since we removed neutral tweets from our corpus.

Sentiment analysis has been researched many times before in the field of artificial intelligence and machine learning [1]. The focuses of research so far have been on blog posts or movie reviews. Many companies and organizations often care about the public's perception of them as a company or their specific products, but they usually classify large amounts of text, which gives them far more information about the sentiment of the document. Twitter sentiment analysis can provide marketers and companies information on how the twitter population reacts to their latest product releases or news, based on reactions of smaller length, which are usually the most common.

1.1 Machine Learning Methods Used

We will test various different classifiers including generative and deterministic models: Naive Bayesian Network, Linear Discriminant Analysis, Support Vector Machines, Feed-forward Neural Networks. All of these models will be trained and tested on the same features extracted from the corpus.

1.2 Corpus

The corpus is based off of 1.6 million tweets from Twitter. It can be found and downloaded at http://cs.stanford.edu/people/alecmgo/trainingandtestdata.zip [2]. It comes in a CSV format with 6 columns, the columns represent in order: sentiment of tweet, tweet id, date of tweet, query for finding the tweet, the username of the tweet creator, and the tweet text. The columns we used for training and testing purposes were only the sentiment and tweet text.

1.3 Challenges

Since we are using a big dataset the costs of training are large. We ran multiple times into situations where our hard disk and ram ran out of space due to large amounts of swap space being used. To mitigate these issues we decided to only train on a tenth of the corpus provided. Since even a tenth of the corpus is still significant (160,000 tweets) we expect the experimental findings will portray accurate results. We also used a Principal Component Analysis for dimensionality reduction for the deterministic models.

1.4 Tools Used

We used the following tools to run our experiments quickly with little overhead: NumPy, sci-kit learn, NLTK, PyBrain. Everything was coded in python.

1.5 Source Code

Source code can be found here: https://github.com/RamaneekGill/Twitter-Sentiment-Analysis .

2 Feature Extraction

To reduce our feature space for computational purposes and reduce noise to have better confidence in our models' predictions we removed punctuations entirely since they can be ambiguous. We also removed neutral tweets from the corpus, since we believe those would just add noise [2]. Finally, we removed stop words from the corpus (words like: 'and', 'it', 'the') since these words are also ambiguous in defining sentiment [3]. For each of our models, we consider a feature as the presence of a word in our global vocabulary of all tokens. stripped of punctuation [4]. We then used only the presence of the top 2000 most common words in the corpus as features in order to, yet again, reduce the dimensionality of our data for efficient computation purposes.

3 Experimental Evaluation

For each of the machine learning methods proposed in section 1.1 we calculated the precision and recall curves of how the best model performed on the test set. Each of the models, except for the neural network, were trained on a tenth of the corpus size and their parameters (defined in each of the subsections below) were fine tuned using ten randomized cross validation iterations. We also have calculated the area under the precision recall curve [5].

3.1 Naive Bayesian Network

Figure 1 shows a probabilistic graphical model representation of a Naive Bayesian Network. The assumption that this model makes, is that each feature is independent of all other features, given its class. That makes training a lot simpler and faster than the rest of the models, but also limits the complexity of the data that can be represented with it. We used the multinomial naive bayes implementation in scikit-learn. The parameter that was fine tuned was the additive (Laplace/Lidstone) smoothing parameter. Figure 2 shows the precision recall curve, the area under the curve is 0.80.

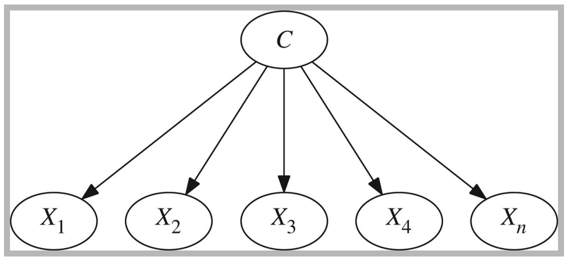

Figure 1: Naive Bayesian Network

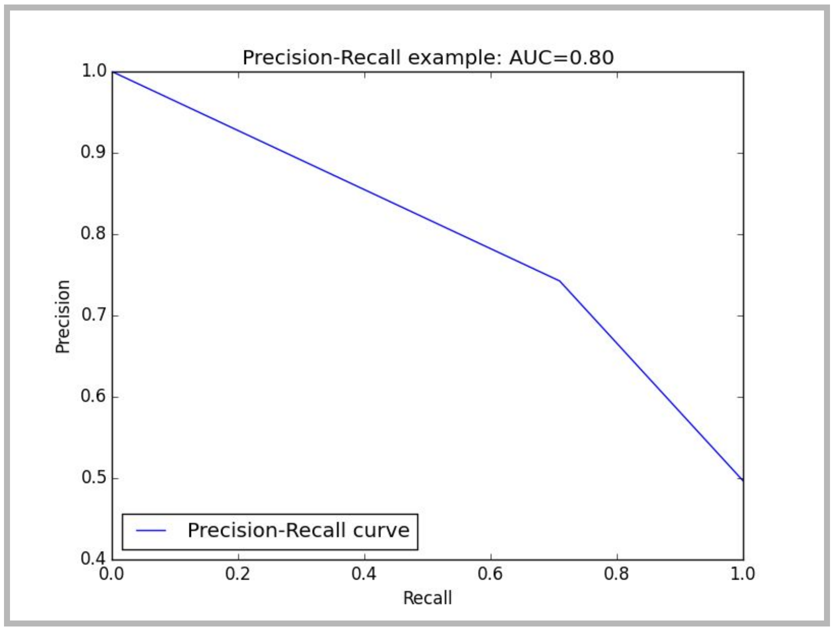

Figure 2: Precision-Recall curve for the test set for the Naive Bayesian Network

3.2 Linear Discriminant Analysis

This model assumes that the conditional probability density functions of each class are normally distributed and that those distributions they have identical covariances. This complicates the model compared to before, thus allowing it to describe more complex relations as well as making training more difficult (although it still is simpler than in a quadratic discriminant model). We used the Linear Discriminant Analysis model with Singular value decomposition implementation in scikit-learn. The parameter that was fine tuned during 10 iterations of randomized cross validation was the threshold used for rank estimation in Singular value decomposition solver. Figure 3 shows the precision recall curve, the area under the curve is 0.81.

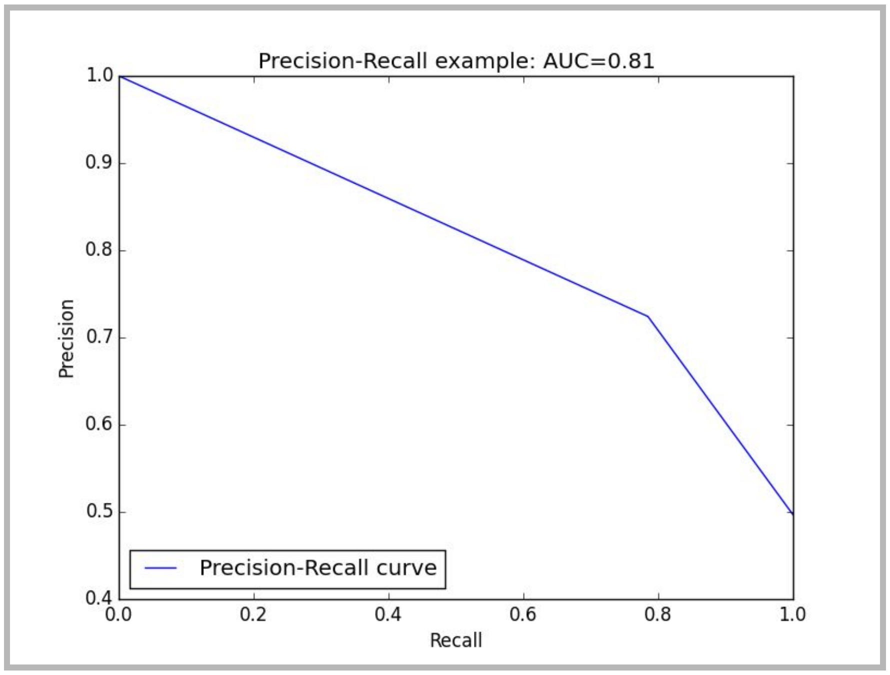

Figure 3: Precision-Recall curve for the test set for the Linear Discriminant Analysis model

3.3 Support Vector Machines

Given a set of training examples with class labels, a support vector machine model tries to assign any new examples in those classes, so it is a non-probabilistic binary classifier. Figure 4 shows an example of a binary support vector classifier with a radial basis function (Gaussian) kernel, which is what we used. The support vectors are calculated using the edge cases of each class in an attempt to separate the two classes with equal distances from each. We used the support vector classifier implementation in scikit-learn. This model was slow to train so we used a randomized PCA with 200 components on the extracted features to make training on 10% of the corpus possible. The parameters for this one were fine tuned using grid search and those where the penalty parameter of the error term (C) and the kernel coefficient (gamma). Figure 5 shows the precision recall curve, the area under the curve is 0.77.

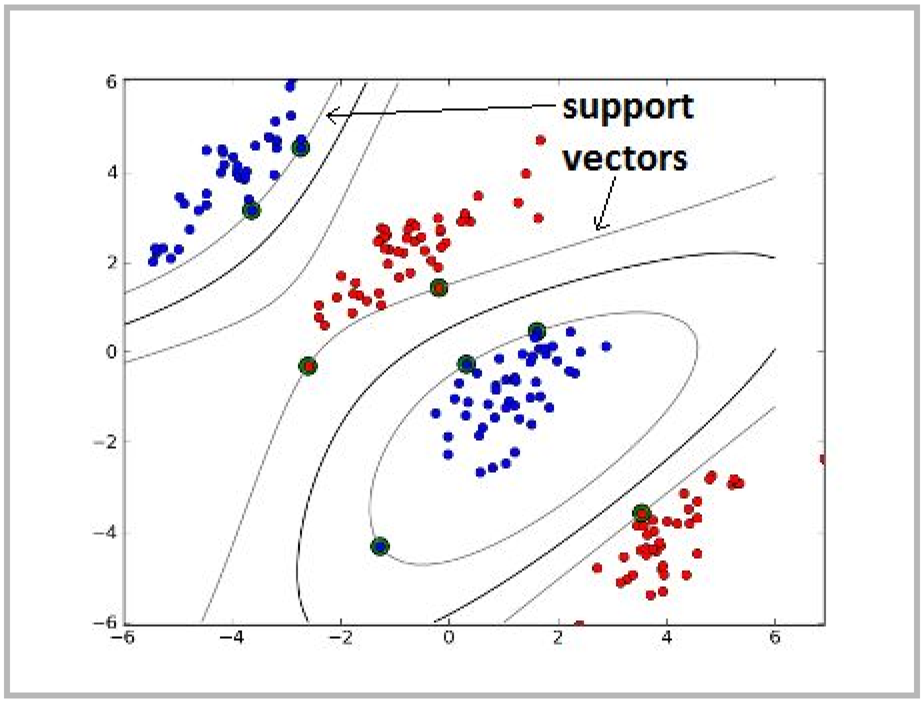

Figure 4: Binary Support Vector Classifier with Gaussian kernel[6]

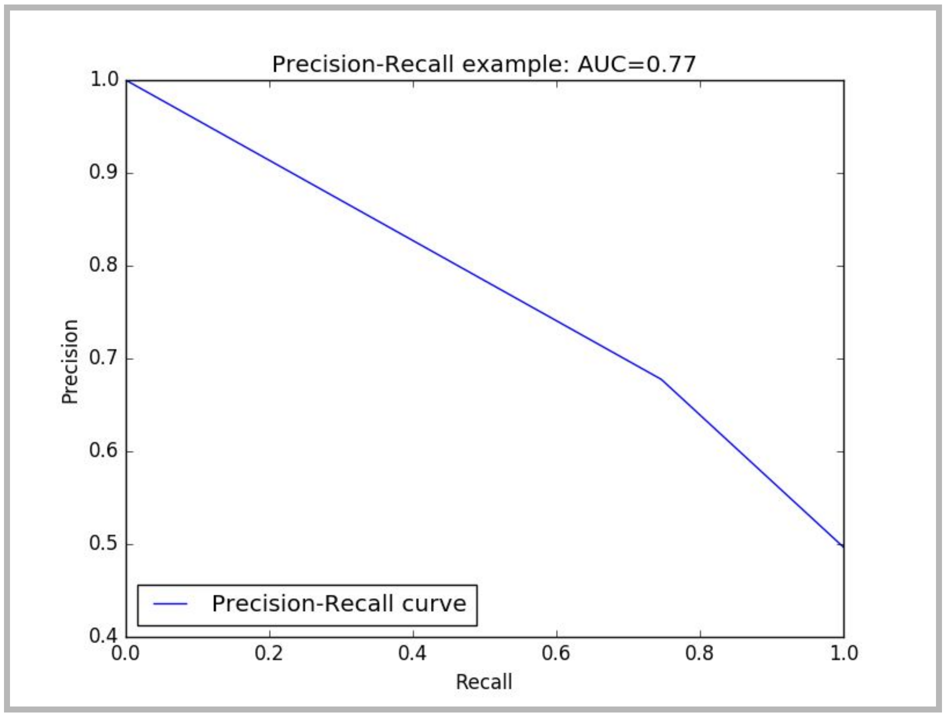

Figure 5: Precision-Recall curve for the test set for the Support Vector Classifier model with parameters C=1.0 and gamma = 1/2000 (number of features).

3.4 Neural Network

Figure 6 shows an example of a feed-forward neural network, each output node represents a class and its value is the confidence(percentage) of the model that the input is of that class, so we used a softmax on the output layer, looked for the highest confidence to derive the class of each new example. This model requires more memory than the SVM and even though we reused the randomized PCA with 200 components to reduce the size of the features, we had to train the model a few times for small amounts of training and handpick the parameters, instead of doing cross-validation (10-fold cross-validation on a sizeable portion of the data set did not seem it would finish within even a week). Since training on 10% of the corpus used more than 16Gb of RAM and would freeze the machine, we trained on 5%. The parameters to be fine tuned in a model such as this one are the number of hidden units, the learning rate and the momentum, figure 7 shows a few of the configurations we tested. In the end, we used 32 hidden sigmoid units, a learning rate of 0.01, a momentum of 0.1, trained the model for 20 epochs and selected the one with minimum valid error (Figure 8).Figure 9 shows the precision recall curve, the area under the curve is 0,73. Increasing the number of hidden units made the model more difficult to train while not showing any improvement in performance.

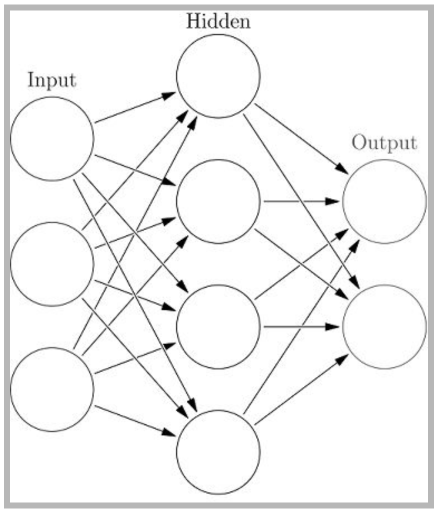

Figure 6: A neural network with 4 hidden units and 2 output nodes.

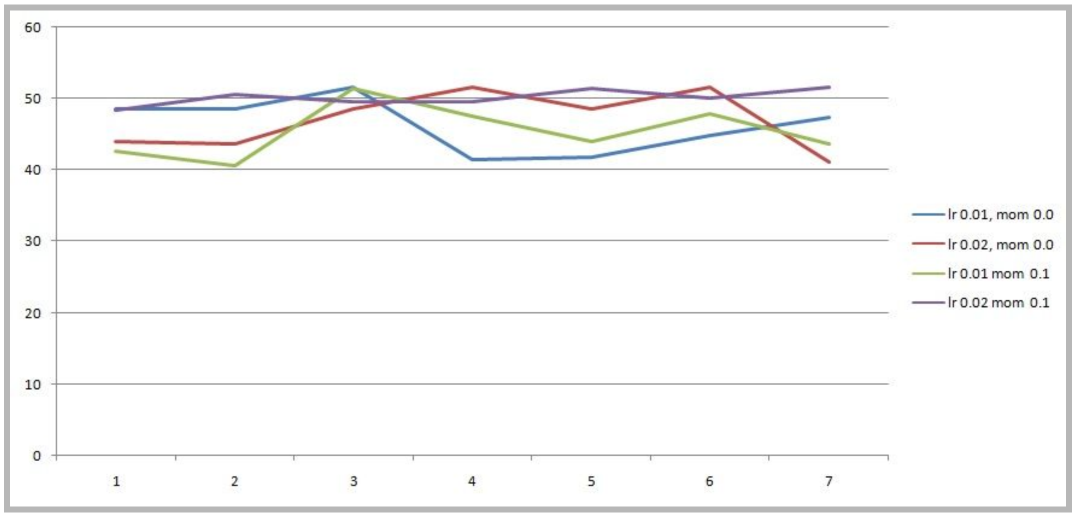

Figure 7: Test error percentages of different configurations for 7 iterations.

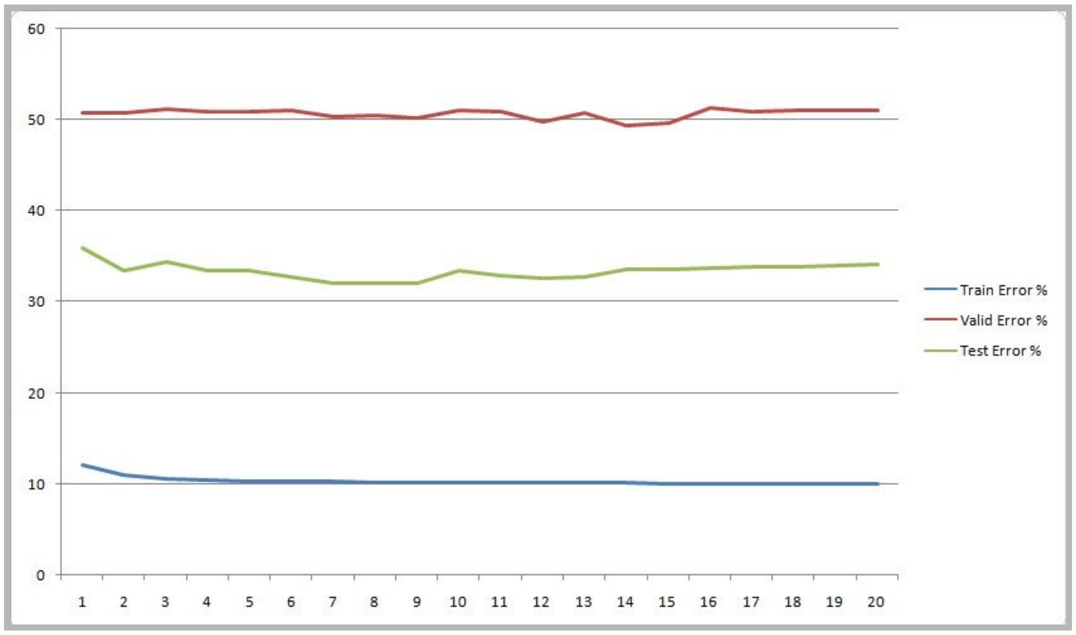

Figure 8: Test and train error percentages of selected model per iteration

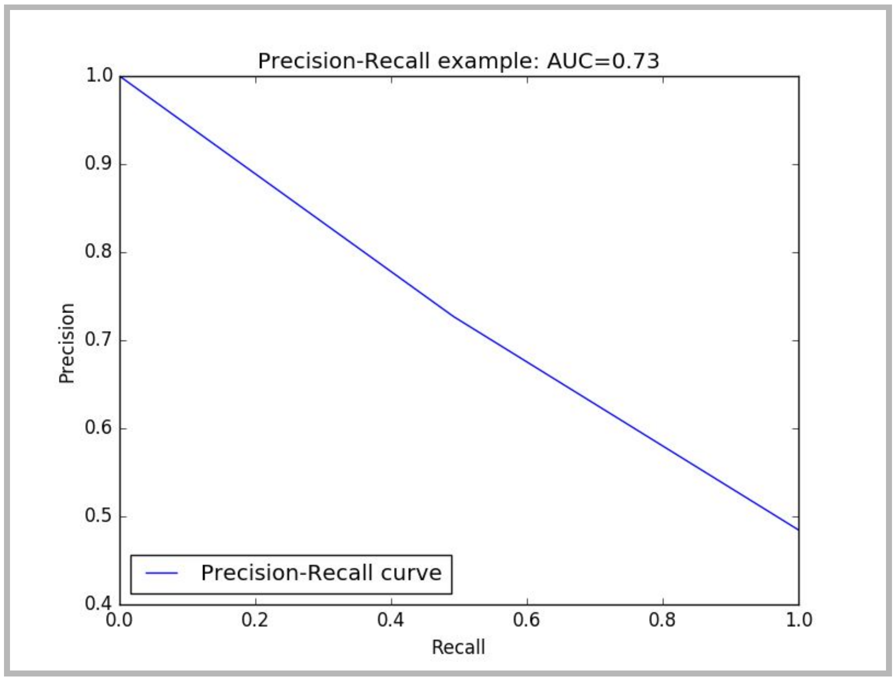

Figure 9: Precision-Recall curve for the test set for the Neural Network model

4 Comparison of Probabilistic Assumptions

The Naive Bayesian Network trains very quickly compared to the LDA model. The computation times for even a subset of the training set and extracted features are very high. Since the performance of both models is similar in practise, it makes more sense to use a Naive Bayesian network because of the cheap computation costs, thus faster training, when compared to the LDA which suffers heavily when using a dataset with many dimensions. On the flip side LDA does provide better results and offers a prosperous chance of improving model strength, when fed more data [7], compared to a Naive Bayes model. Choosing between an LDA or Naive Bayes classifier would highly depend on what requirements and resources a project has.

5 Comparison of Deterministic Assumptions

Being easier to train, the SVM has an advantage over the neural network because we could run cross validation on it. The neural network was also limited by the amount of epochs we could train it for, since the training used only one core of the CPU and took a lot of time, as well as the amount of training data we could afford to use (it trained on half the data the other models did). This seems to be a major problem for the neural network since it seemed to be over-fitting on the training set after a while and better tuning the parameters as well as increasing the training set size are possible solutions to that problem.

To sum up, since the SVM is using the extreme cases of the data to separate the classes, while the neural network tries to learn a meaning for each feature, its connection to the sentiment of the text, in order to get more general rules, it makes sense that the former would outperform the latter when the training data is limited while the feature space is so large [8].

6 Probabilistic vs Deterministic Methods

Probabilistic models are often the popular choice for classification tasks when a dataset has very high dimensions or is sparse because it accounts for the unknown cases/data the model has not been trained on. Being aware of the unknown is highly advantageous when dealing with sparse datasets or high dimensional data since data can be left out without affecting the strength model. Deterministic models do not have this ability since they can only draw on information that has been seen. When using features such as the presence of a popular word deterministic models seem to fall flat when compared to probabilistic models since there is a huge amount of latent variables that generative models can consult when making a prediction.

Naive Bayesian Networks seem to perform very similarly to LDA, this is most likely due to the relatively small feature space we have decided to train on [7]. Since the size of the vocabulary of the spliced dataset is very large the independence assumption seems to work fine for the task of classifying positive or negative sentiment.

Subsequent runs with different portions of the corpus showed that the SVM and the neural network have a lot to gain from more data. The SVM in particular showed 5% more accuracy when doubling the data from 5% to 10% of the corpus. In addition, the neural network model would benefit from a more extensive search for the right parameters. Overall, the deterministic methods seem to be a lot more power and space hungry, but showed promise of improvement if those requirements are met.

The key difference of the two sets of models is the way they use the data to from an understanding of the feature space. Many combinations of words can occur and deterministic models rely heavily on having encountered similar examples before, but since the feature space is huge and the extracted examples are only 10% of the corpus, we don't cover many possibilities. Generative models seem more stable since they are flexible and can generate their "own" sentences for sentiment.

7 Conclusion

We have determined that generative machine learning models are the way to go when classifying sentiment for short texts, because of the high dimensionality of the feature space combined with the inability to use enough samples. We have also determined that even though LDA is a more sophisticated mode, it offers little benefit in this use case when compared to a Naive Bayesian Network.

8 Future Work

Other probabilistic models need to be researched to see how they stack up against a Naive Bayesian Network and LDA. When comparing other models it would be worthwhile to compute the precision-recall curves when varying the size of the dataset. This will require great computation power and time, but the insights learned can help explain why, such a naive assumption of independent conditional probabilities, results in such high performance when classifying the sentiments of short text.

Acknowledgements

We'd like to thank the creators of http://www.sentiment140.com/: Alec Go, Richa Bhayani, and Lei Huang for their public dataset and inspiration to begin this research.

References

[1] Pang, Bo, Lillian Lee, and Shivakumar Vaithyanathan. "Thumbs up?: sentiment classification using machine learning techniques." Proceedings of the ACL-02 conference on Empirical methods in natural language processing-Volume 10. Association for Computational Linguistics, 2002.

[2] Go, Alec, Richa Bhayani, and Lei Huang. "Twitter sentiment classification using distant supervision." CS224N Project Report, Stanford 1 (2009): 12.

[3] Yang, Yiming, and Jan O. Pedersen. "A comparative study on feature selection in text categorization." ICML. Vol. 97. 1997.

[4] Liu, Bing. "Sentiment Analysis and Subjectivity." Handbook of natural language processing 2 (2010): 627-666.

[5] Hanley, James A., and Barbara J. McNeil. "The meaning and use of the area under a receiver operating characteristic (ROC) curve." Radiology 143.1 (1982): 29-36.

[6] Mathieu Blondel, “Support Vector Machines in Python” Mathieu’s Log, September 19th 2010, Web.http://www.mblondel.org/journal/2010/09/19/support-vector-machines-in-python/

[7] McCallum, Andrew, and Kamal Nigam. "A comparison of event models for naive bayes text classification." AAAI-98 workshop on learning for text categorization. Vol. 752. 1998.

[8] Sebastian Raschka, “How can I know if Deep Learning works better for a specific problem than SVM or random forest?”, Book, “Python Machine Learning”, Feb 13th 2016, Web. https://github.com/rasbt/python-machine-learning-book/blob/master/faq/deeplearn-vs-svm-randomforest.md