L90 Assignment 2: Improving Sentiment Classification of Movie Reviews Using Support Vector Machines and doc2vec

Jennifer White (`jw2088`)

Abstract

This works aims to improve work from the previous L90 Assignment on sentiment classification of movie reviews by using Support Vector Machines (SVMs) and doc2vec. It compares the performance of the SVM with and without the use of doc2vec to create document embeddings to use as input. It also compares a variety of models created with `doc2vec` and assess the impact that each one has on the performance of the classifier. It finds that SVMs using Bag of Words vectors perform comparably to the best Naive Bayes models. Additionally, it finds that with the use of a document embedding results can be improved further, reaching up to over 90% accuracy on the blind test set.

1 Introduction

Sentiment classification is a common task within Natural Language Processing (NLP). It involves analysing a comment on a product or service, analysing it and classifying the nature of the sentiment it conveys, commonly into the binary classification of "positive sentiment" or "negative sentiment". Thanks to online movie review websites, a large number of movie reviews created by real people are freely available online, along with star ratings that tell us whether they had broadly positive or negative feelings about the movie. As a result, sentiment classification of movie reviews is a common task with easily available datasets.

In the previous L90 Assignment, this task was approached using a Naive Bayes model with Laplace smoothing. Different configurations of the model were tested and evaluated in order to find the model that gave the best performance.

However, Naive Bayes is a simple model and is unlike to allow us to achieve the best possible performance on this task. Additionally, the data for each review was being given to the model as a Bag of Words (BoW) vector either showing which words were present in the review or additionally providing the frequency of each word in the review. This does not allow the model to harness connections and similarities between any of the vocabulary words or between similar reviews. So it should be possible to improve on the results achieved previously by using a more sophisticated model and by feeding data into the model in a way that allows it to take advantage of this information.

This work investigates using Support Vector Machines (SVMs) for this task. SVMs aim to learn a hyperplane to separate the data such that the smallest distance from any point to the hyperplane is maximized. SVMs also allow the use of a kernel so that the model can learn to separate data that is not linearly separable. In this work, the choice of kernels was investigated in order to find which performed the best for each type of model tested.

Additionally, `doc2vec` is used to learn embeddings of the documents. These are then used as input to the SVM model. This work investigates whether this can improve the performance of the model and which parameters are model choices result in the most useful embeddings.

1.1 Previous Work

In the previous L90 assignment, this task was attempted using a Naive Bayes model. The highest accuracy that was achieved by the best form of the Naive Bayes model was 84.15%. This was achieved with a model that combined both unigrams and bigrams, took as input a vector indicating presence of a unigram or bigram in a document and used Laplace smoothing. This work builds on the work from the previous assignment and attempts to improve this accuracy by using SVMs and using doc2vec to create document embeddings to use as input to the model.

2 Data

2.1 Movie Reviews

The data used for training and testing of the SVM consisted of 2,000 IMDB Movie Reviews, of which 1,000 were positive, and 1,000 were negative. These reviews were available both as raw text and with additional Part of Speech tags. I chose to use the untagged reviews as Pang et al (2002) [4] found that using tags did not result in a statistically significant improvement for this task.

Of this data, 10% was reserved to be used as an unseen test set. 9-fold cross-validation was performed using the remaining data. The classifier that performed best in cross-validation was then tested using the blind test set.

2.2 Data for Training doc2vec

In order to train doc2vec to produce suitable document embeddings, a corpus of 100,000 IMDB movie reviews was used [2]. This dataset was available as raw text. It contained some HTML tags which were stripped before use.

3 Method

The code used for this work can be found on my user area on the MPhil machines (user: `jw2088`) in the file `file`[FILE NAME][1]. The models produced by `doc2vec` that were used to obtain the results discussed are also available on the MPhil machines [WHERE?][2].

3.1 Support Vector Machines

Support Vector Machines are a machine learning technique that aims to learn a separating hyperplane such that the smallest distance from a data point to the hyperplane is maximised. They do not require extremely large amounts of data to train, and they are able to learn to separate data that is not linearly separable through the use of kernels, which makes[TO FINISH] them a useful technique for classification tasks.

The `scikit-learn` [5] SVC implementation of SVMs was used in this work.

The initial, non-`doc2vec` SVM model was tested in various configurations. It was tested using unigrams as vocabulary items and with bigrams as vocabulary items. It was tested with Bag of Word (BoW) vectors. Tests were performed both with BoW vectors that contained information about the frequency of the i^{th} vocabulary item in position i and with vectors that contained a 1 or 0 in position i in order to indicate the presence or absence of the i^{th} vocabulary item.

A frequency cutoff was used meaning that vocabulary items that appeared fewer than 5 times were not included in the BoW vector. Words that were outside of this cutoff were encoded as unknown vocabulary items.

For the models using `doc2vec` vectors as input, the different models were tested against one another, and the choice of kernel was evaluated.

3.1.1 Kernels

SVMs can learn to separate data that is not linearly separable by mapping the data into a higher dimensional space where it becomes linearly separable. To do this, we must define a kernel function that serves as the inner product in the higher dimensional space, since this is what is needed to learn the separating hyperplane and to classify a new vector.

[1] Alternatively, it can be viewed at `URL`

[2] Or they can be downloaded from `URL`

SVMs allow for a choice of kernel that allows the model to learn to sucessfully separate data that is not linearly separable. The implementation of SVMs used in this work allowed for a choice of 4 kernels: `rbf` (radial basis function), `linear`, `poly` and `sigmoid`, which are as follows, where x and x' are two vectors taken as input:

- `rbf`: kernel is of the form $e^{-\gamma||x-x'||^2}$ where γ is a parameter that may be specified but in this work was determined automatically by the model based on features of the training data;

- `linear`: kernel is a linear function of the form $\langle x, x' \rangle$;

- `poly`: kernel is a polynomial function of the form $\gamma(\langle x, x' \rangle + r)^d$. d, the degree, can be specified but in this work the default value of 3 was used. As in the case of γ in the `rbf` case, γ here was also determined automatically. The default value of $r = 0.0$ was used;

- `sigmoid`: kernel is of the form $tanh(\gamma\langle x, x' \rangle + r)$, where the default value $r = 0.0$ was used, and γ was determined automatically.

In this work, all of the models were tested with all of the available kernels. Depending on the model used, the best kernel for separating the data may change.

3.2 doc2vec

When words are used as input for some task, they are often given in the format of one-hot vectors. For a vocabulary of size N, these vectors will be of N dimensions and the vector for word i will contain a 1 at index i. While these are very simple to create, they have some undesirable properties. Using this method produces vectors with very large dimension, which can make computation troublesome. Additionally, they do not allow us to exploit any relations that may exist between words. In the vector space of word vectors, the vector for 'good' will be no closer to the vector for 'great' than to the vector for 'terrible' or 'bad'. One way to improve this is to use a word embedding, such as `word2vec` [3]. This trains an embedding by examining common context for a given word. The vectors that are obtained from this are both much smaller in size than one-hot encoded vectors and also encode relationships between words, such as related words being similar to one another in the embedding space. This should both improve computational efficiency and allow us to exploit similarities between words.

This principle was extended to deal with documents, creating a vector to represent each document and resulting in similar vectors for similar documents [1]. This has clear implications for any task that involves the classification of documents. Documents on a similar topic or containing a similar sentiment should be mapped to vectors that show high similarity in the embedding space.

When training a `doc2vec` model, it is possible to choose between two different approaches: Distributed Memory (DM) and Distributed Bag of Words (DBOW). They differ in the task used in training to obtain the embeddings. In the DM approach, a document vector is concatenated with word vectors from the document and used as input to predict the next word. In the DBOW approach, the document vector is taken as input and asked to predict a word from the document. Less data is needed to be stored in order to use this approach due to the absence of word vectors in the input.

In this work, `doc2vec` was used to create document embeddings to be used as input to the SVM. Positive reviews should be more similar than negative reviews, so this should help us to classify reviews more accurately than using vectors indicating presence or frequency of words. Additionally, these vectors will much smaller than the Bag of Word vectors, so training of the SVM should be considerably faster.

In training a `doc2vec` model, multiple parameters can be changed. As described above, a DBOW or DM model can be chosen, but additionally the size of the embedding can be selected, the frequency cutoff for words to be considered can be changed and the number of

Table 1: Table of parameters used for `doc2vec` models

	Training epochs	DM/ DBOW	Embedding Dimension	Frequency Cutoff
dbow	50	DBOW	100	3
dm	50	DM	100	3
dbow100	100	DBOW	100	3
dm100	100	DM	100	3
dbowlarge	50	DBOW	200	3
dmlarge	50	DM	200	3
dbow5cutoff	50	DBOW	100	5
dm5cutoff	50	DM	100	5
dbow1cutoff	50	DBOW	100	1
dm1cutoff	50	DM	100	1

training epochs can also be chosen. In this work, 10 models were trained and tested on the sentiment classifcation task. Their names and the parameters used are shown in Table 1.

4 Investigation and Evaluation of doc2vec models

Some investigation was performed as to how each `doc2vec` embedding performs when classifying documents. Three particular questions were asked for each trained embedding: (1) How similar are similar reviews in the embedding space? (2) Which word in a review has the vector most similar to that review in the embedding space? (3) How similar are related or synonymous words in the embedding space? Similarity was measured using cosine distance between vectors. These investigations were performed using a test set of 36 new reviews, half positive and half negative, describing 6 movies.

4.0.1 Review Similarity

In these experiments reviews were compared to one another to see whether "similar" reviews exhibited higher similarity in the vector space than reviews that are not "similar". Here, reviews were considered to be "similar" if they expressed the same sentiment or if they described the same movie, and especially so if both of these were the case.

Reviews were compared to reviews within their similarity group and outside these groups. If the embedding is performing well we would expect to see a higher similarity between "similar" reviews than with the full review set in general.

The performance on this task was varied. Some models actually exhibited lower similarity between "similar" reviews than with any 2 reviews from the whole set. All of the embeddings that exhibited this were models that trained using the DM method rather than DBOW. This would be expected to result in worse performance on the classification task for the DM embeddings, which we will observe later is in fact the case.

Most of the DBOW embeddings showed an increase of on average between 1.8% and 1.9% in similarity when looking at reviews in "similar" groupings. The `dbowlarge` embedding showed the greatest increase with an average of 2.5%, but the values for similarity were lower overall with this embedding. The DBOW embeddings did all show increases in similarity within "similar" groups, however these were generally small.

4.0.2 Most Similar Word in Each Review

In these experiments each word in a review was compared with the review as a whole in the embedding space and their similarity calculated.

Table 2: Table of accuracy achieved on the blind test set for non-`doc2vec` models

	rbf	**linear**	**poly**	**sigmoid**
unigram, presence	76.0%	83.5%	72.0%	44.0%
unigram, frequency	84.0%	84.5%	73.5%	83.5%
bigram, presence	56.5%	81.0%	55.3%	53.0%
bigram, frequency	81.0%	80.0%	55.0%	84.0%

Table 3: Table of accuracy achieved on the blind test set for `doc2vec` models

	rbf	**linear**	**poly**	**sigmoid**
dbow	90.0%	89.5%	89.5%	90.5%
dm	81.5%	80.5%	81.5%	79.5%
dbow100	88.0%	88.5%	88.5%	88.0%
dm100	90.0%	89.5%	89.0%	87.5%
dbowlarge	89.0%	88.0%	89.5%	91.0%
dmlarge	81.5%	82.0%	81.5%	83.0%
dbow5cutoff	89.5%	90.5%	91.0%	91.5%
dm5cutoff	84.0%	83.0%	82.0%	83.5%
dbow1cutoff	90.0%	90.5%	87.5%	91.0%
dm1cutoff	81.5%	82.0%	79.5%	74.0%

A list of the 10 words most similar to the review was then returned. In a high quality embedding we would hope to see that the most similar words are key words from the document.

Generally, the embeddings performed poorly on this task, with the majority of the most similar words being common words conveying no information regarding sentiment.

Occasionally a key word such as 'disaster', 'masterpiece' or 'great' appeared in the top 10, but this was infrequent and no model particularly stood out on this task.

4.0.3 Word Similarity

In these experiments, some sets of similar or synonymous words was created (for example, 'good', 'great', 'excellent' and 'amazing'). These words were then compared in the embedding space both to words within their set and words outside their set and the average similarity found for each. In a successful embedding we would expect similarity to be higher when comparing within these groups than outside them.

There were again embeddings that exhibited lower similarity within some of the groups than in general, but this was limited to one group for each embedding.

The embedding exhibiting the greatest increase in similarity when comparing within a group was `dbowlarge` with an average increase of 21.2%. The worst performing on average was `dm100` with an average increase of only 4.1%. One could assume from this that they will exhibit a large performance difference on the sentiment classification task, but we will actually find that they perform comparably on both the cross-validation and blind test sets. However, `dbowlarge` does perform significantly better on the test set of new reviews.

5 Results

The accuracy obtained on the blind test set for the instance of each classifier that performed best in cross-validation can be seen in Tables 2 and 3.

The classifiers using BoW vectors were greatly affected by the choice of kernel. For example, the bigram presence model performed well only with the choice of a linear kernel. The unigram frequency model performed the most consistently with different kernel choices. The best overall performance on the blind test set was achieved by the unigram frequency model with a linear kernel, achieving 84.5% accuracy. This is comparable to the best-performing Naive Bayes models from the previous L90 assignment.

The classifiers using `doc2vec` were generally less significantly affected by the choice of kernel, with the majority of the models showing less than 2% difference between their best- and worst-performing kernel choices. The best performance achieved on the blind test set by these models was achieved by the `dbow5cutoff` model with a sigmoid kernel, achieving 91.5% accuracy. However, many other models performed similarly with 14 other models achieving at least 89.5% accuracy. The performance of the best models using `doc2vec` shows a significant improvement over both the Naive Bayes models used in the previous L90 assignment, and the SVMs without `doc2vec` vectors.

5.1 Permutation Test

The Permutation Test was the test used to test for significance. The permutation test takes a set of paired results from two systems and examines how swapping a random selection of these would affect their means. In particular, it examines whether these permutations would change which mean is the larger one and uses this to determine whether the difference is significant or if it is likely to have occurred by chance. For the results presented here, a difference in means is considered significant if $p < 0.05$.

The significance tests were performed using the paired results of the two systems being compared from the 9-fold cross-validation tests. The average accuracies across cross-validation sets, along with the variance of these accuracy values, is presented in Tables 4 and 5.

In some cases, a system that performed bet-

Table 4: Table of accuracy achieved on the cross-validation test set for non-`doc2vec` models, with variance shown in brackets

	rbf	**linear**	**polynomial**	**sigmoid**
unigram, presence	69.3% (0.083)	81.6% (0.055)	65.2% (0.084)	46.6% (0.159)
unigram, frequency	84.0% (0.050)	83.1% (0.067)	72.8% (0.073)	84.3% (0.061)
bigram, presence	55.4% (0.071)	77.9% (0.089)	53.7% (0.052)	49.4% (0.238)
bigram, frequency	76.7% (0.087)	77.1% (0.145)	54.1% (0.037)	78.9% (0.069)

Table 5: Table of accuracy achieved on the cross-validation test set for `doc2vec` models, with variance shown in brackets

	rbf	**linear**	**polynomial**	**sigmoid**
dbow	88.4% (0.039)	87.8% (0.048)	88.1% (0.038)	87.7% (0.062)
dm	82.6% (0.042)	82.5% (0.036)	81.0% (0.049)	81.0% (0.036)
dbow100	88.0% (0.061)	87.7% (0.072)	87.3% (0.061)	86.8% (0.042)
dm100	87.6% (0.071)	87.3% (0.090)	87.4% (0.058)	86.3% (0.050)
dbowlarge	88.0% (0.048)	87.7% (0.032)	88.5% (0.036)	88.0% (0.042)
dmlarge	82.0% (0.056)	81.7% (0.077)	80.5% (0.035)	81.4% (0.051)
dbow5cutoff	88.4% (0.041)	88.1% (0.038)	88.2% (0.032)	88.2% (0.048)
dm5cutoff	82.1% (0.035)	82.4% (0.033)	81.0% (0.056)	80.8% (0.043)
dbow1cutoff	88.3% (0.071)	88.0% (0.088)	88.1% (0.115)	87.7% (0.060)
dm1cutoff	81.0% (0.031)	81.4% (0.061)	79.4% (0.105)	73.1% (0.088)

ter than another system on the blind test set actually performed worse on average in average of the cross-validation accuracies, so these differences were not significant.

5.2 Statistically Significant Differences

Permutation tests, as described above, were performed on the results to investigate which differences were statistically significant and determine which classifiers were giving the best performance overall.

Among the BoW classifiers, the unigram frequency model performed best with a linear kernel rather than the other kernels and this was found to be a statistically significant difference. Likewise, the bigram frequency model and bigram presence model showed statistically significantly better performance with linear and sigmoid kernels respectively. The bigram presence model showed less variation in performance with the use of different kernels. The sigmoid kernel, which performed best on the cross-validation tests, was statistically significantly better than the polynomial kernel, but the difference with the rbf and linear kernels was not significant.

Once it was established which kernel performed best for each model, these best models were then compared with one another. The unigram presence model with sigmoid kernel was the best-performing, and this showed a statistically significant improvement over the next highest-performing of the 4 models, which was unigram frequency with a linear kernel.

Among the `doc2vec` models, each model showed less variation in accuracy with different kernels and generally the differences between kernel types for an individual model were not found to be statistically significant.

Among the best performing `doc2vec` models performance was very similar and no statistically significant difference was found.

The `doc2vec` models using Distributed Bag of Words (DBOW) were found to be a statistically significant improvement over their Distributed Memory (DM) counterparts.

5.3 Deployment Test

In order to test the generalisability of the systems, a deployment test was performed. For this, 36 new reviews were collected from IMDB, with half being positive (with a rating of 8 out of 10 or higher) and half negative (with a rating of 3 out of 10 or lower). These reviews were primarily for films that have been released in the last year. The files containing these reviews are available alongside the code for this project.

This testing allowed for an examination of how the models generalise to data taken from a different dataset, for entirely different films, written recently.

The results of this testing are given in Tables 6 and 7.

The performance of the models overall was worse on this new test set. The BoW models were unable to perform as well, with the most sucessful model reaching only 69.4% accuracy.

The `doc2vec` models overall still performed better than the BoW models on this test set, but performance still tended to be worse than on the blind test set or on the cross-validation sets. Some models saw a particularly bad drop in accuracy on the new test set. The best performing model on this test set (dm5cutoff `doc2vec` model with a linear kernel) achieved an accuracy of 88.8% on this test set, which is close to the best performance achieved by any model on the blind test set, and which is actually an improvement on this particular model's performance on the cross-validation sets and the blind test set.

6 Discussion

6.1 Error Analysis

In order to better understand the failures of these model, an examination was conducted of the errors that were made on the blind test set.

Examining the errors made by the best models allows for an understanding of the errors that are likely to be the hardest to fix. In order to clearly discuss these errors, a few specific types of errors will be discussed.

Table 6: Table of accuracy achieved on the new review test set for non-`doc2vec` models

	rbf	**linear**	**polynomial**	**sigmoid**
unigram, presence	50.0%	69.4%	50.0%	47.2%
unigram, frequency	61.1%	66.7%	52.8%	63.9%
bigram, presence	50.0%	61.1%	47.2%	55.6%
bigram, frequency	63.9%	66.7%	50.0%	66.7%

Table 7: Table of accuracy achieved on the new review test set for `doc2vec` models

	rbf	**linear**	**polynomial**	**sigmoid**
dbow	77.8%	80.6%	69.4%	80.6%
dm	66.7%	63.9%	58.3%	61.1%
dbow100	80.6%	75.0%	69.4%	77.8%
dm100	72.2%	72.2%	72.2%	72.2%
dbowlarge	86.1%	86.1%	75.0%	83.3%
dmlarge	69.4%	75.0%	58.3%	72.2%
dbow5cutoff	77.8%	77.8%	75.0%	77.8%
dm5cutoff	77.8%	83.3%	63.9%	75.0%
dbow1cutoff	80.6%	77.8%	72.2%	75.0%
dm1cutoff	66.7%	77.8%	61.1%	55.6%

6.1.1 Unclear Reviews

Some reviews that were classified incorrectly are less clear or less emphatic about their sentiment. Some of these reviews were difficult to classify confidently even for a human. These include reviews that state that the film was fantastic overall, but could have been improved in a few areas and focus mostly on speaking about these improvements. These are incorrectly classified as negative. This also includes reviews that are given a score of 3 out of 5 stars which are technically positive but tend to read as quite lukewarm in sentiment and are much harder to classify. These may be the hardest errors to fix as they can be difficult even for humans when they are not explicit in their sentiment.

6.1.2 Reviews with Negations

Some reviews that were classified incorrectly include sentiments that may indicate positive or negative sentiment but where this has been negated in some way, and the classifier fails to pick up on this. For example, a reviewer may state that they detest the films of a particular director, but that this film was an exception, or a positive adjective may follow the word 'not'. These types of misclassifications could possibly be improved by performing pre-processing on the reviews that allowed for detection of negation and appending some token to a word as an indication of negation. Alternatively, this may be improved by use of an architecture such as LSTMs that would allow for better understanding of a sentence as a whole and the interaction between words rather than them being viewed individually.

6.1.3 Reviews Without Focus on Sentiment

Some of the reviews that were misclassified were reviews that focus largely on describing the film, its plot and the the actors and directors who were involved with it. They usually offer a short overview of their opinion at the end of the review. Since the majority of the review gives no indication of the sentiment, the classifier can fail to confidently recognise it. These errors could be difficult to fix as they require the model to recognise what is relevant information for the task of sentence classification. Perhaps this could be improved upon with an LSTM architecture which may be able to learn what information is necessary to remember over time with respect to sentiment.

6.2 Possible Improvements

Further improvements for this task could be made by changing the model used to perform sentiment classification, by perhaps using a neural network architecture, such as an LSTM which would allow for longer term dependencies to be understood and could help fix misclassifications caused by negations, for example. However, this would require much more than 2,000 examples as a testing and training set. Though, as movie review sentiment analysis is a well-researched task, much larger datasets, such as the one used to train `doc2vec` in this work, are available.

Without the need for additional data, the result could potential also be improved by using an ensemble of classifiers.

Further investigation could also be performed into the ideal parameters for training the `doc2vec` model, and whether a better model could be created than those used here. The performance of the `doc2vec` model could also potentially be improved by performing additional preprocessing on the training data and movie reviews. For example, removing stopwords or stemming the words in the review, and thus removing additional data that is unlikely to provide additional information as to the sentiment of the review.

Additionally, some approaches to sentiment analysis have involved preprocessing such as changing a word in a review to demonstrate that it has been negated if preceded by 'not', so that, for example, the word 'good' is not considered to be positive in the phrase 'the film was not good' [4].

Implementing some or all of these techniques could allow for the accuracy achieved in this work to be improved.

Word count: 3,883 words

References

[1] Quoc Le and Tomas Mikolov. Distributed representations of sentences and documents. In *Proceedings of the 31st International Conference on International Conference on Machine Learning - Volume 32*, ICML'14, pages II–1188–II–1196. JMLR.org, 2014.

[2] Andrew L. Maas, Raymond E. Daly, Peter T. Pham, Dan Huang, Andrew Y. Ng, and Christopher Potts. Learning word vectors for sentiment analysis. In *Proceedings of the 49th Annual Meeting of the Association for Computational Linguistics: Human Language Technologies*, pages 142–150, Portland, Oregon, USA, June 2011. Association for Computational Linguistics.

[3] Tomas Mikolov, Ilya Sutskever, Kai Chen, Greg S Corrado, and Jeff Dean. Distributed representations of words and phrases and their compositionality. In C. J. C. Burges, L. Bottou, M. Welling, Z. Ghahramani, and K. Q. Weinberger, editors, *Advances in Neural Information Processing Systems 26*, pages 3111–3119. Curran Associates, Inc., 2013.

[4] Bo Pang, Lillian Lee, and Shivakumar Vaithyanathan. Thumbs up?: sentiment classification using machine learning techniques. In *Proceedings of the ACL-02 conference on Empirical methods in natural language processing-Volume 10*, pages 79–86. Association for Computational Linguistics, 2002.

[5] F. Pedregosa, G. Varoquaux, A. Gramfort, V. Michel, B. Thirion, O. Grisel, M. Blondel, P. Prettenhofer, R. Weiss, V. Dubourg, J. Vanderplas, A. Passos, D. Cournapeau, M. Brucher, M. Perrot, and E. Duchesnay. Scikit-learn: Machine learning in Python. *Journal of Machine Learning Research*, 12:2825–2830, 2011.