UNIVERSITY OF
CAMBRIDGE

Sentiment Analysis of Movie Reviews

December 2024

Word Count: 1996

Contents

1 Introduction

Text classification is a common task within Natural Language Processing. This report examines the sentiment classification of movie reviews, exploring the performance impact of symbolic approaches and machine learning models.

2 Baseline and Essentials

2.1 Sentiment Lexicons

Before diving into machine learning approaches, we consider a simple symbolic method using a sentiment lexicon - a list of words annotated with their corresponding sentiment. The method is defined as follows:

$$S_{\text{binary}}(w_1 w_2 \dots w_n) = \sum_{i=1}^{n} sgn(SLex[w_i]) \tag{1}$$

For each word w_i in a review, add **1** if the label is **positive** and **-1** if **negative**. If $S_{\text{binary}}(w_1 w_2 \dots w_n) \geq t \in \mathbb{R}$, where t is the polarity threshold, the review is classified as positive, else negative.

The magnitude is also present so we can incorporate the strength of the word sentiment:

$$S_{\text{weighted}}(w_1 w_2 \dots w_n) = \sum_{i=1}^{n} SLex[w_i] \tag{2}$$

with weights as follows:

Word Magnitude	**SLex[$\mathbf{w_i}$]**
Strong Positive	2
Strong Negative	-2
Weak Positive	1
Weak Negative	-1

Table 1: Sentiment Lexicon Magnitudes and Weights

The results were similar: **67.4% accuracy** and **67.9% accuracy** for the **binary method** and the **weighted method** respectively. The sign test [10, pg. 80] showed the weighted results were not statistically significant with respect to the binary results.

2.2 Naive Bayes Classifier

Next we will explore a simple probabilistic machine learning approach, Naive Bayes implemented as a Bag-of-Words model. This is defined as follows:

$$\hat{c} = \arg\max_{c \in C} P(c \mid \boldsymbol{f}) = \arg\max_{c \in C} P(c) \prod_{i=1}^{n} P(f_i \mid c) \tag{3}$$

where

- $c \in C = \{POS, NEG\}$: the class
- $\hat{c}$: the most probable class
- $\boldsymbol{f}$: the feature vector of a review
- f_i: the number of times the i-th word appears in the review

To perform our held-out test we split the reviews into 10 folds, taking the first 9 as the train set and the last as the test set. Results in **Table 2** show an **accuracy of 51.5%**, slightly better than random guessing. This poor result is due to the presence of words in the test data that were not seen during training, causing probabilities, leading to the model's inability to generalise - see **Equation (4)**.

$$P(w_i \mid c) = \frac{\text{count}(w_i, c)}{\sum_{w \in V} \text{count}(w, c)} \tag{4}$$

where word w, is in vocabulary V in the training set.

Smoothing adds a non-negative constant so a small probability is assigned to unseen words. We will use Laplace smoothing with $\alpha = 1$.

$$P(w_i \mid c) = \frac{\text{count}(w_i, c) + \alpha}{\sum_{w \in V} \text{count}(w, c) + \alpha|V|} \quad \text{where } \alpha \in \mathbb{R} > 0 \tag{5}$$

Smoothing results in **82.5%** accuracy; this is a statistically significant performance increase compared to the results in **Section 2.1**. However, relying on the same test set risks **Type III** errors, leading to poor generalisation on new data [12]. To combat this, we will use cross-validation.

N-fold cross-validation divides the data into N chunks. The model is trained on all chunks but one and tested on the remaining chunk. This is repeated N times with a different test chunk. The Round-Robin N-fold cross-validation results in **81.6% ± 1.8% accuracy**. The standard deviation is small, signifying the performance is similar across chunks thus,

showing the model's reliability.

Method	**Accuracy ± Std. Dev. (%)**
NB: Held-Out Test	51.5 ± 0
Smooth NB: Held-Out Test	82.5 ± 0
Smooth NB: Cross-Validation	81.6 ± 1.8

Table 2: Impact of Smoothing on Naive Bayes Classification Performance[1]

We will use the cross validation result in **Table 2** as our baseline model to draw comparisons throughout the report. Only cross validation results for future models will be reported.

2.3 Processing, Features and Over-fitting

In our model **(Equation 3)** the count of each unique token corresponds to a dimension in the feature vector. However, there is often noise in text, resulting in higher dimensions than needed. This can lead to over-fitting and computational inefficiency. Consider the following tokens, **'run', 'RUN!', 'Running'**. We can map each to **'run'** since they all have the same meaning. Stop-words such as **'and'** and **'the'** occur frequently in text yet may be uninformative [11]. These pre-processing techniques can reduce noise and unnecessary model complexity.

Model	**Accuracy ± Std. Dev. (%)**	**Features**
Smooth NB (Baseline)	81.6 ± 1.8	52,555
Smooth NB + Stemming	82.0 ± 2.2	32,404
Smooth NB + Light Clean	81.7 ± 1.8	45,342
Smooth NB + Remove Punctuation	81.8 ± 2.1	52,496
Smooth NB + Remove Stop-Words	81.3 ± 2.4	52,408

Table 3: Impact of Pre-processing Methods on Naive Bayes Classification Performance

In **Table 3** we can see that each pre-processing technique causes a reduction in the number of features as expected. The impact on accuracy is not significant compared to the baseline; however, notice that the accuracy of all techniques improved slightly except the removal of stop-words. English stop-words in the NLTK package include words such as **'no' and 'not'**. Negations are important indicators of sentiment; without negations the sentiment is altered. Hence, their removal can cause poorer model performance.

[1]The standard deviation is 0 for held-out tests since they are based on a single evaluation.

Please note that **'light clean'** refers to converting text to lowercase and the removal of part-of-speech tags that appear as tokens in the tag file, which contribute to noise. Refer to **Figure 1**.

```
The     DT
story   NN
revolves        VBZ
around  IN
the     DT
adventures      NNS
of      IN
free-spirited   JJ
Kayley  NNP
-LRB-   -LRB-
voiced  VBN
by      IN
Jessalyn        NNP
Gilsig  NNP
-RRB-   -RRB-
,       ,
the     DT
early-teen      JJ
daughter        NN
of      IN
```

Figure 1: Misplacement of Part-of-Speech Tags in the Token Section of a Tag File

2.4 Analysis of Bi-grams and Tri-grams

Increased model complexity is not always bad; it can allow us to detect meaningful patterns in data. N-grams are sequences of n-words; they help provide the local context of a word by retaining some order [5, chp. 3]. For example, bi-grams help account for negation in the following bi-grams: **('not', 'bad'), ('n't', 'the', 'worst')**. Both n-grams hold positive/neutral sentiment whereas individually, **'bad'** and **'worst'** are deemed negative.

Table 4 shows bi-grams and trigrams improve accuracy slightly over the baseline (uni-grams), but the improvement is not statistically significant. Notice the tri-gram model and the uni-gram, bi-gram and tri-gram model, show lower performance. This is due to data sparsity - this where there is a large feature space but many of the trigrams do not appear in the test data, leading to less reliable probabilities [9, pg. 201].

The growth function for the number of possible features is also present in **Table 4**. Despite this, the growth for the number of features observed is linear. This is because not every combination of bi-grams and trigrams is present in the data. So although theoretically the trigram model could have $\mathbf{52,555^3}$ trigrams, not all trigrams occur in the dataset.

Model	Accuracy ± Std. Dev. (%)	Observed Features	Growth Function
Smooth NB (Baseline)	81.6 ± 1.8	52,555	x
Smooth NB with B only	84.4 ± 1.7	447,531	x^2
Smooth NB with T only	84.2 ± 2.3	962,519	x^3
Smooth NB with U + B + T	83.0 ± 1.7	1,462,605	$x + x^2 + x^3$

Table 4: Comparison of Naive Bayes Models with Different N-grams

A sample of bi-grams can be seen in **Figure 2**. Notice **('amusingly', 'contemptuous')**, together can show positive sentiment. However, when considered separately, as uni-grams, **'contemptuous'** is strongly negative and could lead to a misinterpretation of sentiment.

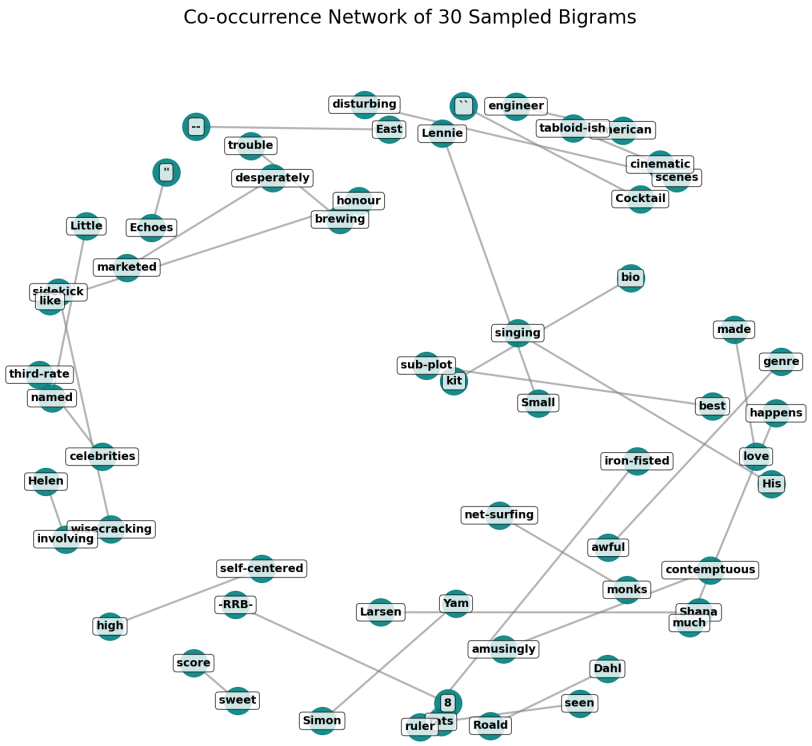

Figure 2: Sample of 30 Bi-grams Corresponding to the Bi-gram Model present in Table 4

Additionally, there is potential noise in the data such as POS tags and punctuation. We can examine the n-gram models with pre-processed data as described in **Section 2.3**. The results in **Table 5** show general but very slight improvement with light cleaning compared to **Table 4**. Removing punctuation alongside this reduces the accuracy slightly as it can alter bi-gram structures, leading to misleading sentiments. Suppose we have **('loved', '!'),('!', 'not'), ('not', 'bad')**. Without punctuation, this becomes **('loved', 'not'), ('not', 'bad')**, where the bigram **('loved', 'not')** could incorrectly imply negative

sentiment.

Model[2]	**Light Clean (%)**	**Light Clean + Remove Punctuation (%)**
Smooth NB with B only	84.3 ± 1.5	83.7 ± 1.7
Smooth NB with T only	84.8 ± 1.8	83.1 ± 2.3
Smooth NB with U + B + T	83.6 ± 2.1	82.9 ± 1.9

Table 5: Impact of Pre-processing on Naive Bayes Classification Performance with Different N-grams

2.5 Support Vector Machine (SVM) Classifier

Naive Bayes assumes feature independence however, words are often correlated. For example, **'not'** and **'good'** together hold negative sentiment. SVMs do not presume feature independence and posterior probabilities but instead, use a decision boundary which can be defined as follows [1, pg. 326]:

$$\mathbf{y}(\mathbf{x}) = \mathbf{w} \cdot \phi(\mathbf{x}) + b \tag{6}$$

where

- $\mathbf{w}$: weight vector
- $\phi(\mathbf{x})$: feature-space transformation of input vectors x
- b: bias term

The optimal decision boundary maximises the distance between the decision boundary and the support vectors - the points closest to the boundary from each class.

Hence, SVM Bag-of-Words based models are expected to perform better than Naive Bayes. Note a linear was kernel used; most text classification problems are linearly separable [3]. Although the results in Table 6 are not statistically significant compared to our Naive Bayes baseline model, the initial SVM model shows a higher accuracy of $\mathbf{83.8\% \pm 2.1\%}$.

[2]The notations are defined as follows: B for Bi-grams, T for Tri-grams and U for Uni-grams

Model	Accuracy ± Std. Dev. (%)	Features
SVM	83.8 ± 2.1	52555
SVM - POS Tagging	84.1 ± 2.7	59990
SVM - Discarding Closed-Class Words	84.7 ± 1.4	52097

Table 6: Performance of SVM Classification Models

2.6 Part-of-Speech (POS) Features

Part-of-Speech tags can provide linguistic context on the syntactic structures of words. **Table 6** shows POS tagging introduces **14.1%** more features, increasing model complexity. Yet, the accuracy only increases by **0.3%**, with increased variability, compared to the initial SVM model. Since order is ignored in Bag-of-Words models, adding POS tags provides limited value. Closed-class words are often uninformative and contribute to noise. Discarding closed-class words (i.e. keeping only nouns, adjectives, verbs and adverbs) shows slightly higher performance with less features. This suggests that pruning irrelevant POS tags may be a more efficient method in comparison.

3 Extension

3.1 Doc2Vec Embedding Models

Up until now we have used Bag-of-Words representations where the semantics of words are ignored and we rely on exact word matches, unlike Doc2Vec embeddings [7]. Semantics are important for example, **'the movie was brilliant'**, is expected to be closer to **'amazing film'** than **'the bear escaped'** in the vector space. Doc2Vec uses paragraph vectors [13], i.e. a vector for each review, that acts as context. In this section we will analyse various Doc2Vec embeddings, under the **Distributed Memory (DM)** and the **Distributed Bag-of-Words (DBoW)** methods, trained on IMDB reviews [8] and classify our initial movie review data using the SVM model defined in **Section 2.5**.

In DM, shown in **Figure 3**, paragraphs form a matrix **(D)** where each column is a paragraph vector; acting as memory. The corresponding n word vectors **(W)**, based on the window size, are provided [7]. These vectors are averaged (alternatively concatenated) to predict the next word. The prediction error is used to update vectors iteratively via stochastic gradient decent.

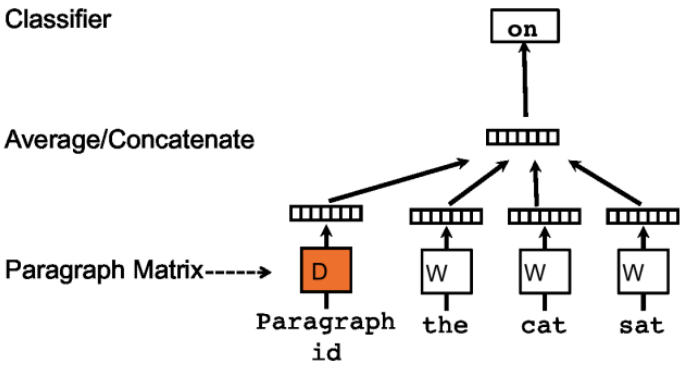

Figure 3: Doc2Vec Distributed Memory Algorithm [7]

In (DBoW), as shown in **Figure 4**, words within the window size are sampled from the paragraph [7]. The model attempts to predict these words using the paragraph vector. Stochastic gradient descent is applied to update the vectors iteratively based on the prediction error.

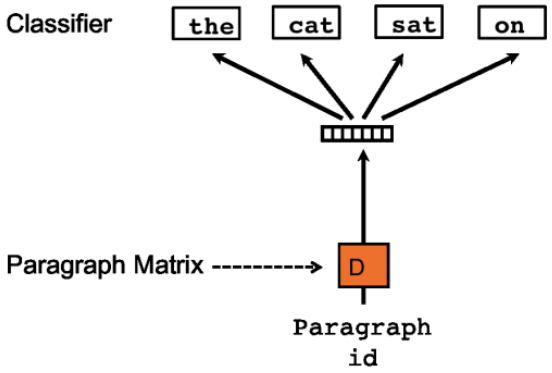

Figure 4: Doc2Vec Distributed Bag-of-Words Algorithm [7]

Tables 7 and 8 compare DBoW and D models by **vector size (v)** and **epochs (e)**. As discussed in **Section 2.3**, a light clean has been implemented in both the embedding and classification data, as well as the removal of html tags present within the embedding data, to remove noise.

Clearly the 100D models perform better than the 50D models. This because there are more dimensions to effectively encode semantic relationships. **The best model is DBoW with 100D and 20 epochs**, in fact all DBoW models are significant compared to our baseline model, whereas the DM models are not. Notice we consistently achieve higher performance even in 50D and lower epochs with DBoW compared to DM. This is consistent with Lau and Baldwins findings [6]. DBoW does not use word order which may provide

better generalisation for sentiment classification.

Model Name	**Accuracy ± Std. Dev. (%)**
doc2vec_dm_v100_e10	82.8 ± 2.1
doc2vec_dm_v100_e15	83.1 ± 2.4
doc2vec_dm_v100_e20	83.3 ± 1.6
doc2vec_dm_v100_e25	83.1 ± 2.4
doc2vec_dm_v50_e10	79.2 ± 1.6
doc2vec_dm_v50_e15	80.0 ± 2.3
doc2vec_dm_v50_e20	80.1 ± 1.7
doc2vec_dm_v50_e25	80.8 ± 2.4

Table 7: Performance of Various Doc2Vec Distributed Memory Embeddings with SVM Classification

Model Name	**Accuracy ± Std. Dev. (%)**
doc2vec_dbow_v100_e10	86.2 ± 1.9
doc2vec_dbow_v100_e15	87.9 ± 1.8
doc2vec_dbow_v100_e20	88.0 ± 2.1
doc2vec_dbow_v100_e25	87.5 ± 1.9
doc2vec_dbow_v50_e10	87.0 ± 1.7
doc2vec_dbow_v50_e15	87.3 ± 2.4
doc2vec_dbow_v50_e20	87.3 ± 2.1
doc2vec_dbow_v50_e25	87.4 ± 1.8

Table 8: Performance of Various Doc2Vec Distributed Bag-of-Words Embeddings with SVM Classification

Figure 5 provides a closer look at both methods with 100 dimensions. Accuracies **plateau at 20 epochs** suggesting the model has converged. The accuracy begins to **dip at 25 epochs** potentially due to excessive training leading to over-fitting. Even at 10 epochs, DBoW outperforms DM, suggesting that DBoW is less computationally intensive. DBoW's simplicity also means it tends to be faster, compared to DM which jointly infers word and paragraph vectors [6].

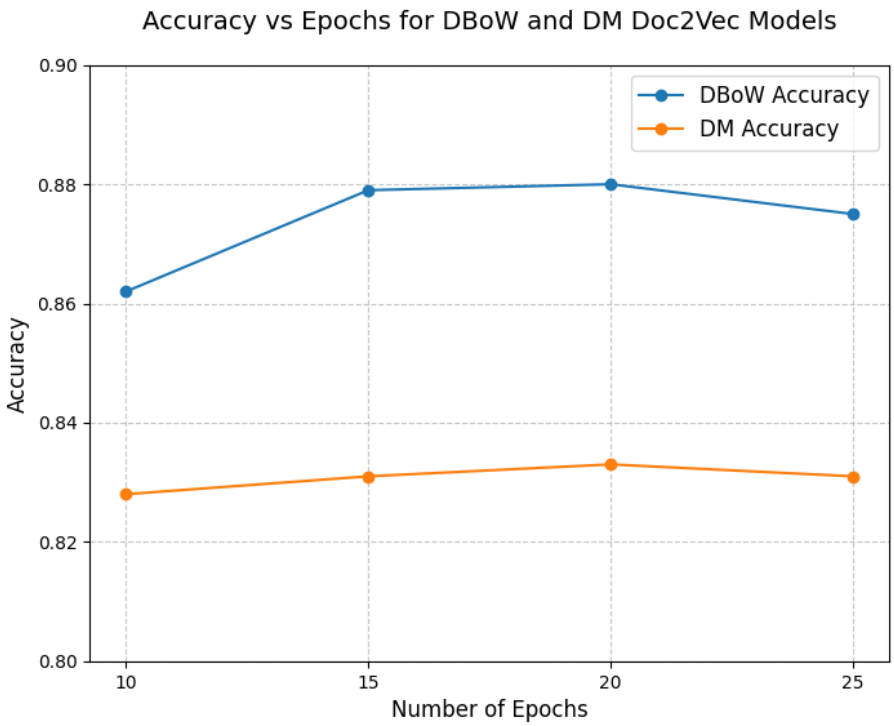

Figure 5: Accuracy Trends Across Epochs for 100-Dimensional DBoW and DM Doc2Vec Models

3.2 Analysis of the Doc2Vec Space

Let us delve deeper into the vector space. **Figure 6** shows there is significant overlap between the positive and negative reviews, suggesting our highest performing model perceives negative and positive reviews as similar. However the high accuracy seen in **Table 8** implies the overlap is largely due to projecting 100D vectors to 3D, resulting in information loss. Despite this some separation can be seen with negative reviews towards the left and positive, to the right.

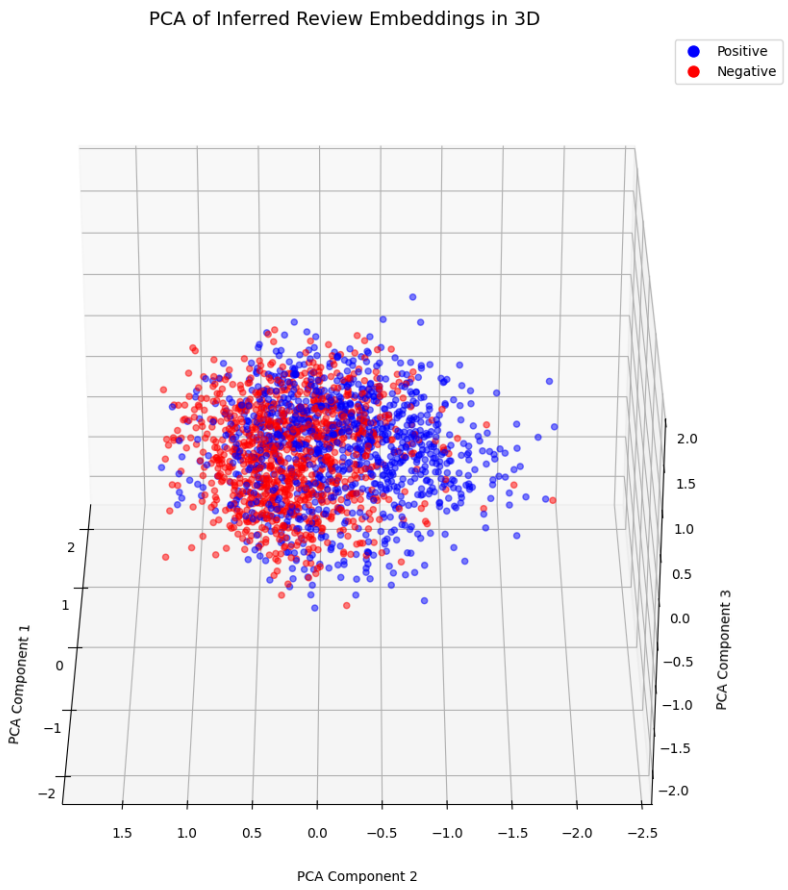

Figure 6: Classification of Reviews with `doc2vec_dbow_v100_e20` Vector Representations Reduced to 3D using PCA Dimensionality Reduction [4]

Despite `doc2vec_dbow_v100_e20` performing well with review embeddings, results in **Table 9** indicate the models poor ability to identify semantic relationships in word vectors. In contrast, in **Table 10**, the `doc2vec_dm_v100_e20` model shows strong semantic coherence. This is expected as DM trains both the paragraph and word vectors, whereas DBoW does not rely on word vectors - see **Section 3.1**. The `doc2vec_dm_v100_e20` model also effectively captures minor variations in punctuation. However, it incorrectly considers **'good'** and **'bad'** as similar, possibly because they often appear in similar contexts.

Word in doc2vec_dbow_v100_e20 Model	Similar Words	Cosine Similarity (%)
good	steps".	45.1%
	lemp	44.8%
	dujardin	44.6%
	coach,	42.0%
	dreyer's	41.3%
bad	provokes.	41.9%
	(motel	41.7%
	danning,	41.0%
	"movie."	40.4%
	sweeter.	39.6%

Table 9: Top 5 Words Most Similar to 'good' and 'bad' in the `doc2vec_dbow_v100_e20` Model

Word in doc2vec_dm_v100_e20 Model	Similar Words	Cosine Similarity (%)
good	great	82.9%
	decent	82.0%
	bad	77.5%
	fine	74.7%
	good,	73.3%
bad	terrible	79.0%
	good	77.5%
	horrible	75.7%
	bad,	74.6%
	lousy	73.7%

Table 10: Top 5 Words Most Similar to 'good' and 'bad' in the `doc2vec_dm_v100_e20` Model

Given `doc2vec_dm_v100_e20` captures better semantic relationships within words, we will examine the similarity between reviews and their critical words in this vector space. In **Figure 7**, **review 1342** primarily discusses disappointment but mentions some **"performances are top notch"**, aligning with the **0.28** and **0.19** cosine similarities for **'disappointment'** and **'amazing'**. **Review 20** refers to a horror movie the reviewer enjoyed which is consistent with the high similarity to **'horror'** and slight similarity to **'amazing'**. As expected, **review 16** shows high similarity with **'amazing'**, as it states the reviewer's amazement alongside other synonyms such as **'marvellous'** and **'astonishing'**.

However, its similarity to **'disappointment'** suggests an unintended proximity in the vector space, demonstrating the model struggles to differentiate antonyms as seen in **Table 10**.

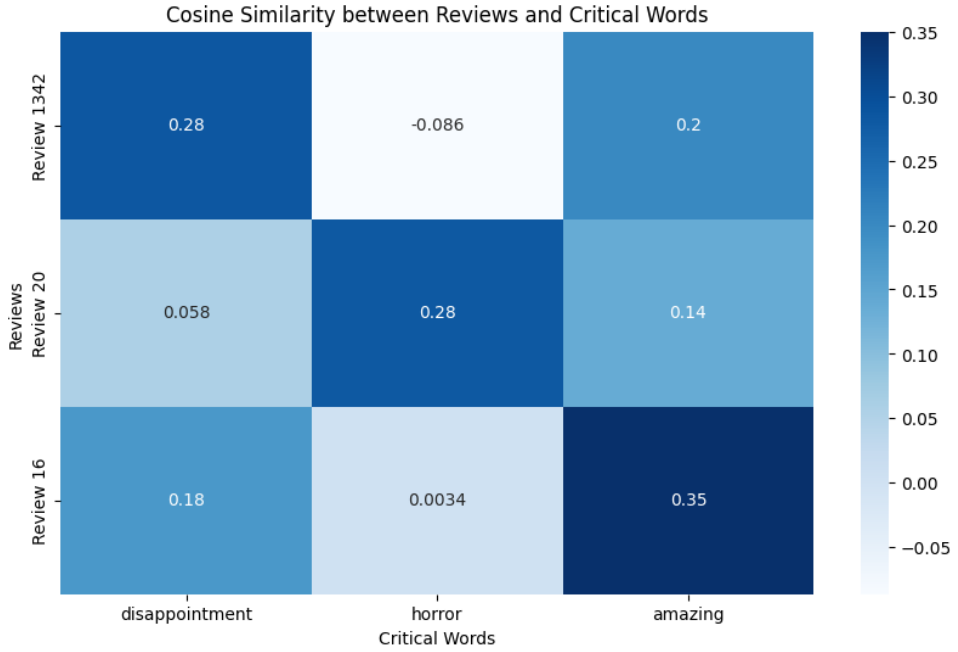

Figure 7: Cosine Similarity Between Reviews and Critical Words in the `doc2vec_dm_v100_e20` Model

4 Conclusions and Further Improvements

We have seen the progression of sentiment classification, from sentiment lexicons to data manipulation techniques to SVMs and embedding models. More specifically, DBoW embedding models performed statistically significantly better our Naive Bayes baseline model, consistently outperformed DM on sentiment classification and is more computationally efficient thus, more scalable. However, DM showed stronger semantic coherence on a word level and is more suitable for text similarity and prediction tasks. Although it struggles to differentiate antonyms; a common problem with this embedding approach [2].

Studies show that models combining both methods performs consistently well across many tasks [7]. The combination of the top performing DM and DBoW models, `doc2vec_combined`, supports this. It showed both high accuracy, **87.5% ± 1.6%**, and strong semantic coherence - see **Table 11**.

Word in doc2vec_combined Model	**Similar Words**	**Cosine Similarity (%)**
good	great	82.9%
	decent	82.0%
	bad	77.5%
	fine	74.7%
	good,	73.3%
bad	terrible	79.0%
	good	77.5%
	horrible	75.7%
	bad,	74.6%
	lousy	73.7%

Table 11: Top 5 Words Most Similar to 'good' and 'bad' in the `doc2vec_combined` Model

Bibliography

[1] Christopher M. Bishop. *Pattern Recognition and Machine Learning*. Springer, 2006.

[2] Nikola Mrkšić et al. Counter-fitting word vectors to linguistic constraints. 2016. URL `https://arxiv.org/abs/1603.00892`. Accessed: 10 December 2024.

[3] Thorsten Joachims. Text categorization with support vector machines: Learning with many relevant features. In *Machine Learning: ECML-98*, volume 1398 of *Lecture Notes in Computer Science*, pages 137–142, 1998.

[4] Ian T. Jolliffe and Jorge Cadima. Principal component analysis: a review and recent developments. *Philosophical Transactions of the Royal Society A: Mathematical, Physical and Engineering Sciences*, 2016. URL `https://royalsocietypublishing.org/doi/10.1098/rsta.2015.0202`. Accessed: 10 December 2024.

[5] Daniel Jurafsky and James H. Martin. *Speech and Language Processing*. Stanford University, 3rd edition, 2024.

[6] Jey Han Lau and Timothy Baldwin. An empirical evaluation of doc2vec with practical insights into document embedding generation. In *Proceedings of the 1st Workshop on Representation Learning for NLP*, pages 78–86. Association for Computational Linguistics, 2016. URL `https://aclanthology.org/W16-1609`.

[7] Quoc V Le and Tomas Mikolov. Distributed representations of sentences and documents. 2014. URL `https://arxiv.org/pdf/1405.4053`. Accessed: 10 December 2024.

[8] 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. URL `http://www.aclweb.org/anthology/P11-1015`.

[9] Christopher Manning and Hinrich Schutze. *Foundations of Statistical Natural Language Processing*. MIT Press, 1999.

[10] S. Siegel and Jr. Castellan, N. J. *Nonparametric Statistics for the Behavioral Sciences*. McGraw-Hill Book Company, 2nd edition, 1988.

[11] Ravjot Singh. Exploring nlp preprocessing techniques: Stopwords, bag of words, and word cloud, 2024. URL https://becominghuman.ai/exploring-nlp-preprocessing-techniques-stopwords-bag-of-words-and-word-cloud-c2cd67ce3fc8. Accessed: 10 December 2024.

[12] Simone Teufel. Overtraining and cross-validation - machine learning and real-world data, lecture 5, 2023-2024. URL https://www.cl.cam.ac.uk/teaching/2324/MLRD/slides/MLRD_5.pdf. Accessed: 10 December 2024.

[13] Radim Řehůřek. Doc2vec paragraph embeddings, 2014. URL https://radimrehurek.com/gensim/models/doc2vec.html. Accessed: 10 December 2024.