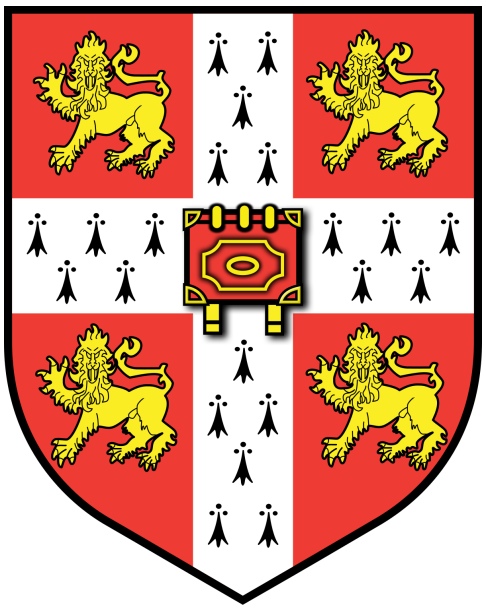

Exploring Techniques for Text Sentiment Classification

Student ID: J902G
Word Count: 1965[a]
Submitted: December 4, 2021

[a]The word count excludes the Bibliography, the Appendix, and quoted reviews.

Contents

1 Introduction

The purpose of this report was to analyse different methods for classifying text sentiment. Part 1 discusses Bag-of-Features approaches, where the features can be individual words (unigrams) or combinations of N words (N-grams), and investigates Naive Bayes and SVM algorithms. Subsequently, Part 2 discusses more advanced approaches using the Doc2Vec algorithm, which gives more complex feature representations for the reviews, and therefore explores more advanced sentiment relationships. Throughout the report, binomial siginificance testing is used to gauge whether accuracy improvements or deterioration between algorithms is statistically significant.

2 Part 1: Methods with Naive Word Embeddings

2.1 Using a Sentiment Lexicon (Question 0)

Binomial Classes (Question 0.0)

If our lexicon has words that are pre-tagged with positive and negative sentiment then a simple count based approach can be taken, with positive words contributing +1 to each review's score, and negative words -1. The review is then classed as positive if its score is ≥ 0, or negative otherwise. Using this approach a **classification accuracy of 67**% was achieved. This result is lower than perhaps expected, for example because the approach does not account for negation of positive words in a negative document[1].

Including Semantic Subjectivity (Questions 0.1/0.2)

The semantic lexicon model can be more fine-tuned by additionally including word subjectivity[2] as described in [1]. Now a strongly positive (negative) word contributes +2 (-2) to the score, and a weakly positive (negative) one +1 (-1). With this adaptation an **accuracy of 68**% was achieved, which was not statistically significant, with $p = 0.82. > 0.05$.

2.2 Naive Bayes Approaches (Questions 1-5)

The short-comings of Semantic Lexicon are that it does not account for word context and it requires a pre-labelled corpus, which restricts vocabulary size. To use an unlabeled corpus, we turn to Bayesian inference, and use training data to predict a test-document's sentiment probabilities given its word/feature content. The mathematics for the Naive Bayes' (NB) approach can be seen in Appendix A.1.

[1] e.g. a review reading 'compared to good films, this one was not great' could be calculated as positive as the phrase 'not great' could have net sentiment $-1 + 1 = 0$.

[2] The subjectivity is an indicator of the reliability of a word having a positive or negative sentiment (like 'abhorrent' is reliably negative), and hence is called a 'reliability tag' in [1].

Without Smoothing (Question 1.0)

To use Bayesian inference we need sentiment likelihoods for each feature, $P(\vec{f}_i^{(j)}|C_i)$, and the naive method is to build this likelihood using training data and a frequentist approach. E.g. for positive sentiment:

$$P(\vec{f}_i^{(j)}|C_i = +) = \frac{\text{Count}(\vec{f}_i^{(j)}, C_i = +)}{\sum_{f \in V} \text{Count}(\vec{f}, C_i = +)} \tag{1}$$

where V is the *vocabulary*, the entire set of features in the training-set. Post-training, the classification of a test-document can be inferred via[3]:

$$\hat{C}_{test} = \arg\max_{k \in \{+,-\}} \sum_j \ln P(\vec{f}_{test}^{(j)}|C_{test} = k) \tag{2}$$

In cases where the test feature has not be seen in training, $P(\vec{f}_{test}^{(j)}|C_{test}) = 0$, and $\ln(0)$ issues arises[4]. Therefore in these situations the contribution to the log-posterior from that test-feature is ignored: $\ln P(\vec{f}_{test}^{(j)}|C_{test}) := 0$.

With this approach an **accuracy of 45%** was achieved on the held-out test-set; not much of an improvement over classifying all reviews as one sentiment. This was due to many words in the test-set not being in the training data, and hence many test-features were ignored in the classification. To combat this issue a smoothing parameter can be introduced so that if a test-feature was not present at training, it is assigned a small, non-zero, probability and *not* ignored.

With Smoothing (Questions 2.0/2.1)

Equation (1) is thus adapted to include a smoothing parameter, κ, such that:

$$P(\vec{f}_i^{(j)}|C_i = +) = \frac{\text{Count}(\vec{f}_i^{(j)}, C_i = +) + \kappa}{\sum_{f \in V} \Big(\text{Count}(\vec{f}, C_i = +) + \kappa\Big)} \tag{3}$$

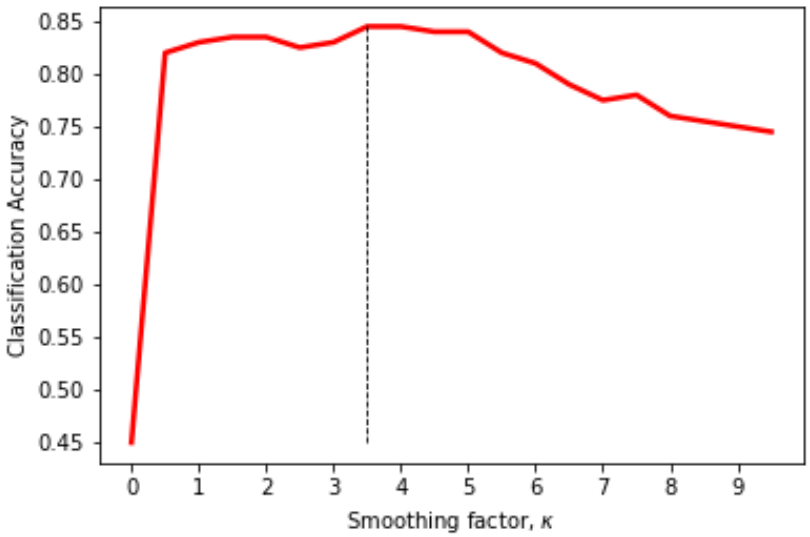

Figure 1: Variation of classification accuracy with κ.

[3] see Appendix A.1 for derivation

[4] This is the same as saying that the probability of the test document given each sentiment is 0, which cannot be true as the sum over positive and negative sentiments must be 1.

As seen in Figure 1, including a well-chosen smoothing parameter drastically improves classification accuracy. After $\kappa \approx 5$ the accuracy begins to fall again, eventually getting to 50%, as κ begins to dominate the word counts:

$$\lim_{\kappa\to\infty} P(\vec{f}_i^{(j)}|C_i = +) = \frac{\kappa}{\sum_{f\in V}\kappa} = \frac{1}{\text{length}(V)} = \lim_{\kappa\to\infty} P(\vec{f}_i^{(j)}|C_i = -) \tag{4}$$

Choosing $\kappa = 4$ gives an **accuracy of 84**% however, which is a statistically significant improvement to not using smoothing, $p = 0.00013 << 0.05$.

Using Cross-Validation (Questions 3.0/3.1)

To avoid biases from the test-set (for example if one element of the test set is much more similar to the training set than another), 10-fold round-robin cross-validation was used for the test accuracies, giving a mean and standard deviation. With this, a **mean accuracy of 81.0**% (standard deviation 2.1%) was obtained for the smoothed Naive Bayes classifier ($\kappa = 1$). The small standard deviation shows there is little spread across the 10 folds, and hence there is not likely to be bias in the test-set. For the rest of Part 1, 10-fold round-robin cross-validation is used for reported accuracies unless stated otherwise.

Using Stemming (Questions 4.0/4.1/4.2)

Stemming can be used to re-introduce language information back into the model. Stemming pools different inflectional forms of the same base word, so that words such as 'great' and 'greatness' are counted as the same rather than separately, which simplifies the lexicon vocabulary. In the case of our dataset's vocabulary, using stemming reduced the number of unique words from 52,556 to 32,562 (38 percent reduction). This greater simplicity works to reduce overfitting. Despite this, stemming did not significantly improve the results from smoothing, with an **accuracy of 81.3**% (standard deviation 2.6%), with p-value $p = 0.42 > 0.05$.

N-gram Naive Bayes Models (Questions 5.0/5.1)

Thus far, only unigrams, which treat the review contents as a bag of words, have been considered. Therefore N-gram models can be used to give some word context, as in addition to the unigram features, they include features for all N-word combinations found in the vocabulary.

Note that whilst this method now provides word context, it also rapidly increases the number of features, and hence can quickly lead to overfitting due to the sparsity of the feature space if the training-set is small or the vocabulary diverse. This is demonstrated using unigram+bigram, unigram+trigram, and unigram+bigram+trigram models, with the results shown below in Table 1.

It appears as though the word context models overfit to the training data as the test-set accuracies progressively decrease with N, suggesting that the reviews rarely use the same words in the same orders (i.e. in the same contexts). Whilst this was not statistically significant compared to the unigram case for the most part, including both unigrams, bigrams and trigrams gave so many features that the model starts overfitting to the point of statistically significant deterioration. The number of features increases with N approximately according to the linear relationship:

$$M_N = 500,000N - 550,000 \tag{5}$$

for $N > 1$, where M_N is the vocabulary size and M_N the number of features for the N-gram-only model[5].

[5]i.e. not including the 52,556 words from the unigram model for $N > 1$

N-gram Model	Accuracy (std. deviation)	Number of Features	Significant vs smoothed NB?
Unigram Only	81.0% (2.1%)	52,556	N/A
Unigram + Bigram	76.7% (2.0%)	499,062	False ($p = 0.26$)
Unigram + Trigram	70.4% (2.7%)	990,616	False ($p = 0.056$)
Unigram + Bigram + Trigram	66.4% (2.3%)	1,437,122	True ($p = 0.0072$)
Unigram Only (LC)	**81.7**% (**2.4**%)	45,349	False ($p = 0.78$)
Unigram + Bigram (LC)	76.1% (2.1%)	470,129	False ($p = 0.26$)
Unigram + Trigram (LC)	68.9% (3.1%)	965,835	True ($p = 0.028$)
Unigram + Bigram + Trigram (LC)	65.3% (3.1%)	1,390,615	True ($p = 0.0056$)

Table 1: Accuracies for different N-gram models, all using smoothing ($\kappa = 1$) but no stemming. LC stands for lower-case, and all the review contents were made lower-case before both training and testing so that the capitalised words at the start of a sentence are not treated separately. As expected, this reduced the vocabulary size slightly.

Naive Bayes Section Summary

In summary, the performances of the NB methods were (ignoring some tests using lower-case, stemming, and smoothing parameters $\kappa \neq 1$ for brevity, if they did not give competitive results):

Model	Accuracy (std. deviation)	Significant vs smoothed NB?
Unigram (no smoothing)	44.6% (2.6%)	True ($p = 0.00013$)
Unigram (smoothing, $\kappa = 1$)	81.0% (2.1%)	N/A
Unigram (smoothing, $\kappa = 4$)	81.5% (2.7%)	False ($p = 0.84$)
Unigram (LC) (smoothing, $\kappa = 1$)	81.7% (2.4%)	False ($p = 0.78$)
Unigram (LC) (smoothing, $\kappa = 4$)	**82.3**% (**2.0**%)	False ($p = 0.44$)
Unigram (stemming, smoothing, $\kappa = 1$)	81.3% (2.6%)	False ($p = 0.94$)
Unigram+Bigram (smoothing, $\kappa = 1$)	76.7% (2.0%)	False ($p = 0.26$)
Unigram+Trigram (smoothing, $\kappa = 1$)	70.4% (2.7%)	False ($p = 0.056$)
Unigram+Bigram+Trigram (smoothing, $\kappa = 1$)	66.4% (2.3%)	True ($p = 0.0072$)

Table 2: Summary of results using 10-fold cross-validation on the Naive Bayes approaches. The best model was the Unigram model with smoothing and using a lower-case corpus and a smoothing parameter of 4. However, this choice required more extensive parameter tuning than the original non-lower-case corpus model with a smoothing parameter of 1, and therefore from here-on, this model is taken as the baseline when using SVMs in Section 2.3.

2.3 Support Vector Machines (SVMs) (Questions 6-7)

Aside: Less Naive Features

When using a SVM, the inputs must be equal length vectors, which means the review texts have to be pre-processed to convert them to such. The simplest way to do this is to use count vectors - each element in a review vector is the count of how frequently a feature occurs in that review (with 0 if it was not present). However, this means uninformative features' counts are weighted equally to important features' counts. If there are many more uninformative than informative features in a review, their axes will dominate the feature vector and reduce the sentiment information given by the useful features.

Therefore, the count vectors were converted to an importance-weighted vector via the Term Frequency-Inverse Document Frequency (TF-IDF) transform[6]. With TF-IDF, unhelpful features like 'the'/'and' are ignored in each review's vector, whilst the axes of features such as 'good'/'bad' are boosted. The result is better distinction between positive and negative sentiment vectors in the feature space, and hence better classification accuracies.

A Unigram SVM (Question 6.0)

Using a SVM with an RBF kernel and unigram TF-IDF transformed input features, an **accuracy of 83.1**% was achieved (standard deviation 0.7%). Compared to the smoothed NB results, this was an improvement, but not a statistically significant one.

Using Part-of-Speech (POS) Tags (Questions 7.0/7.1)

Part-of-Speech tags can be incorporated into the input texts to give more context to the features, for example whether a particular word is being used as a verb or a noun. To do this, the reviews' words were adapted to be 'word_POS', which increased the feature space dimension. When using POS tags the **accuracy decreased slightly, to 81.7**% (standard deviation 0.9%), possibly due to the higher input dimension leading to overfitting.

It could be caused by too many POS tags due to the inclusion of 'closed-class words', which mainly serve grammatical purposes, and contribute little to semantic meaning. Discarding these closed-class words led to accuracy improvements, giving an **accuracy of 83.8**% (standard deviation 1.2%). The reason for this is similar to the justification for using TF-IDF - by discarding closed-class words we discard many uninformative features such as grammatical words, and thus the remaining features hold more information about a review's sentiment, thus allowing for greater separation between its feature vectors.

If we were to go in the opposite direction however, and not only disclude POS tags completely, but also use stemming to *reduce* the number of features, then an even better **accuracy of 84.1**% is achieved (standard deviation 0.6%). This demonstrates the trade-off between word context and input dimension. Whilst POS tags give linguistic context to the words which can improve performance, the higher dimensions of inputs it leads to can lead to overfitting. Therefore, for the relatively small training-set of 1800 reviews, the simpler models often outperform the more complex ones which are prone to overfitting.

If there were a way to include linguistic and word context without greatly increasing the input dimensions, then an optimal middle-ground could be reached. This is what the Doc2Vec algorithm discussed in Part 2 aims to do.

[6]For details on how this is derived see Appendix A.2

SVM Section Summary

The SVM results can be summarised as:

Model	Accuracy (std. deviation)	Significant vs smoothed NB?
SVM	83.1% (0.7%)	False ($p = 0.26$)
SVM (POS tags)	81.7% (0.9%)	False ($p = 0.46$)
SVM (POS tags, discard closed-classes)	83.8% (1.2%)	False ($p = 0.21$)
SVM (stemming)	**84.1**% (**0.6**%)	False ($p = 0.14$)

Table 3: Accuracies for a SVM with an RBF kernel, all using unigram features but with different linguistic adaptations. As a reminder, the accuracy of the smoothed Naive Bayes algorithm under the original review conditions (smoothing factor 1 without stemming or lower-case) was 81.0%.

3 Part 2: The Doc2Vec Algorithm

The best-performing algorithms in Part 1 used a form of word embedding that was less naive than the basic word counts. However, even the most complex of the previously considered embeddings, the TF-IDF feature (see Section 2.3) still uses a BOW based approach, and gives little allowance for word context. At the same time, the bigram and trigram models that do account for word context gave large and sparse feature vectors that were prone to overfitting. The Doc2Vec model proposes a solution for this: it accounts for word context in a way that does not increase feature dimensions by using (trained) linear transforms from the count vectors to an output feature vector for each document [4]. The Doc2Vec model architectures and descriptions of how they work are in the Appendix A.3.

In all of this section, after a paragraph vector was found for each review using a Doc2Vec model, the sentiment classification was done using a SVM with an RBF kernel.

Model Specifications (Question 8.0)

1. DM-C: Distributed memory model with concatenation of word and paragraph vectors, with a context window size of 10 and a feature vector length of 100
2. DM-M: Distributed memory model with mean of word and paragraph vectors, with a context window size of 10 and a feature vector length of 100
3. DBOW: Distributed Bag of Words model with feature vector length of 100
4. DBOW + DM-M: Concatenation of the DBOW and DM-M models, giving overall feature vector length of 200
5. DBOW + DM-C: Concatenation of the DBOW and DM-C models, giving overall feature vector length of 200

Each of the models trained above has a minimum count of 2, meaning any words in the vocabulary that appears less than twice are ignored. With a dynamic learning rate starting at 0.027 and finishing at 0.005 after 12 epochs, the performances of the 5 models were the following:

Model	Validation Accuracy	Test Accuracy	Statistically Significant vs Smoothed NB Accuracy?
DM-C	79.3%	78.7%	False ($p = 0.36$)
DM-M	84.8%	81.6%	False ($p = 0.32$)
DBOW	**90.2**%	**90.0**%	True ($p = 0.013$)
DBOW + DM-C	90.0%	89.6%	True ($p = 0.013$)
DBOW + DM-M	90.0%	89.0%	True ($p = 0.013$)

Table 4: Accuracies of the tested models after 12 epochs. The validation set is used for the Doc2Vec model's (unsupervised) training (i.e. when all parameters are being trained). The test set is used only post-training, but requires SGD of the paragraph weights due to the architecture of Doc2Vec (see Appendix A.3), whilst all other parameters are held constant. Note that in order to use the binomial significance test, the new IMDB corpus was passed through the smoothed Naive Bayes model (Section 2.2), for training and testing, which gave an accuracy of 81.6%.

These results are surprising in that the simplest model, DBOW, which does not account for word context, performs best across both the validation and the test accuracies. This suggests that additional model complexity slightly causes overfitting to the training data, and also explains the marginally poorer generalisation to the test cases seen in the concatenated Doc2Vecs. This is a similar story to the N-gram models in Section 2 that account for word context, which also performed more poorly than the monogram NB model. This indicates that the IMDB review data-set contains a large vocabulary and/or word contexts vary a lot throughout the dataset, meaning that when words are put into context windows greater than 1, the models are prone to overfitting.

It is also worth noting that during training, after early epochs, the more complex models (DM-M and DM-C and their concatenations) converged a lot faster to their 12 epoch validation accuracy than DBOW, supporting the idea that added model complexity helps models fit to the training cases faster, but makes them less able to generalise.

Investigating Paragraph Representations (Question 8.1)

Similarly to its predecessor Word2Vec [3], the training of paragraph vectors to best represent their word content makes Doc2vec capable of producing semantic vector relations between reviews. The use of vectors also allows for similarity scores to be defined [4], such that for each document we can return the most similar neighbours. An example of this is shown in Table 5 using the most accurate model, DBOW. As this example uses a document from the training data that the Doc2Vec model has previously, we would expect the most similar document to be itself (this also acts as a sanity check of the model's predictions).

Original:	Document ID: 92975	
Most similar	92975	75.1%
2^{nd} most similar	67773	73.2%
Least Similar	18048	23.3 %

Table 5: A table showing the most similar documents to the review shown above. The rightmost column is the similarity score (between 0 and 1) for the documents.

```
" the purple rose of cairo " was the first allen's movie i saw back in moscow in the end
   of the 80s and it started my eternal love for his films . " the purple rose . . . "
   is wonderful , is one if allen's greats - a perfect combination of poignancy and
   humor , romance and drama , reality and fiction . it is the movie-within-the-movie
   that blends sophisticated romance from the lives of rich and beautiful where the
   dashing main hero with bravery and chivalry written into his character always gets a
   girl and depression era new york city where a poor waitress tries to escape the
   realities of her joyless life in the movie theater . the story focuses on cecilia (
   mia farrow ) , a waitress and a battered wife of an unemployed abusive man ( danny
   aiello ) . cecilia only feels alive when she watches her favorite movies that take
   her away from her dreary realities . one day , as she watches " purple rose of cairo
   " for the 10th or maybe 15th time , the leading man tom baxter ( jeff daniels )
   decides to leave the movie and be with cecilia in real life . his screen partners are
   left confused and " trapped " in a scene they can't get out of . the live actor who
   plays baxter is blamed by the film's producer for his character's rebellion and tries
   to get him back on the screen . cecilia's husband finds out that his wife was seen
   with a good looking man instead of working as a babysitter in the evening . on the
   top of all , tom baxters in other theaters try to leave " purple rose of cairo " ,
   too . . . it is not the first or last time allen has played with the concept of the
```

```
thin line ( in this film , the silver screen ) that divides film's world and reality
but rarely has he created the film as sweet , gentle , sad , technically realized and
simply terrific as " purple rose of cairo " . 9/10
```

Listing 1: The original review used for the results of Table 5 above. Note that here, and in the rest of the review examples taken from Doc2Vec, the Doc2Vec method automatically transforms all of the reviews to lowercase prior to finding paragraph vectors.

```
the best movie i saw in my all life great actors great story just great . .
```

Listing 2: The second most similar film as listed in Table 5 above.

```
this film is an attempt to present jared diamonds theory of " guns , germs and steel " ,
    explaining how europeans have dominated much of the globe . the version i saw of this
    documentary came on 2 discs covering 3 hours . i think the information could have
    been presented in 20 minutes . there are completely useless scenes of : professor
    jared diamond watching birds through binoculars , professor jared diamond failing to
    use a bow and arrow properly , professor jared diamond firing a muzzle-loader badly .
    was this documentary supposed to make a hero out of " professor jared diamond ? " .
    this part of the documentary was so bad , it could have been a spoof . the worst was
    when diamond is shown breaking down and weeping when touring the malaria ward in an
    african hospital . none of this helps me understand his theory of " guns , germs and
    steel . " btw , " guns , germs and steel " is said about 100 times . " can the
    europeans guns , germs and steel get them out of this dire situation ? stay tuned and
    find out ! " when he finally gets down to business , his theory is equal parts
    interesting and utterly boring . europeans conquered the natives peoples of other
    lands , because they had guns and fine blades against stone and wooden weapons . do i
    really need a professor to convince me of this ? the parts of his theory that explain
    how the europeans came to have the advantages that allow the conquest are interesting
    , but the coverage is paper-thin . in the end , i think the documentary was only
    trying to convince me that non-europeans are as capable as europeans . if i'm not a
    racist , i already know this . if i'm a racist , jared diamond is not going to
    convince me with his bumbling use of native implements . i don't think adults are the
    intended audience for this documentary . kids may enjoy this more than i , though . i
    have read that the book from which this documentary is much better than the
    documentary .
```

Listing 3: The least similar film with the greatest distance from the paragraph vector of the original review.

When looking at reviews that are classed as similar and dissimilar above, we can see firstly that the similarity is independent of review length, unlike the sparse document vectors in SVM classification[7]. In addition, they correctly pick-up sentiment, with the most similar being positive (like the original) and the least similar being negative, as expected. We can also go one step further and look at how the models represent individual words within reviews. Table 6 shows this for the DBOW and DM-M models. The nonsensical order of the DBOW model compared to DM-M is because, unlike DM-M, DBOW does not account for word context, and therefore does not train word representations in parallel to paragraph representations (see Appendix A.3). This results in the word order and assigned score being random and completely arbitrary.

[7]In SVMs, even if two reviews contain the same words at the same relative frequency, they will have different magnitude vectors if the review lengths are different and vectors un-normalised.

Original word: *'excellent'*			
DBOW		DMM	
'blindfolding'	0.457	'outstanding'	0.909
'swordsmanship'	0.411	'exceptional'	0.813
'jolt'	0.407	'superb'	0.776
'shellacked'	0.405	'awesome'	0.775
'doran'	0.384	'amazing'	0.762
'1/9'	0.382	'terrific'	0.724
'conveyor-belted'	0.378	'incredible'	0.714
"yuen's"	0.377	'fantastic'	0.697
'swordfighting'	0.377	'excellent'	0.685
"'sophisticated'"	0.370	'above-average'	0.675

Table 6: Word similarities showing the most similar 10 words to *'excellent'* using the trained DBOW and DMM models, both with 100 dimensional paragraph vectors. The right hand column for each model is the similarity score, normalised between 0 and 1.

Finally, we are able to visualise the paragraph vectors to see similarity clusters. As the paragraph vectors have been made 100 dimensional, they will have to be mapped first to a 2-dimensional space (or 3, but that is still hard to visualise) first. There are many methods for dimensionality reduction, but perhaps the most flexible[8] is T-distributed Stochastic Neighbour Embedding (t-SNE) [2], which is often used for visualising NLP word embeddings [5]. A plot of a t-SNE plot using 1000 documents is shown in Figure 2, which shows how even in two-dimensions, there exists some mapping that allows for separation of paragraph vectors based on their sentiment. Having said this, the overlap between sentiment classes and occasional incorrectly mapped vector show that there are some reviews containing words that make it hard to distinguish which sentiment they belong to. An example of this is the review with ID 3314, which is classed as positive, but sits in the overlap (see Figure 2). This is understandable however, when reading the review text:

```
although flawed in it's view of homosexuals , this movie will shed light for the viewer
    about the myths and inaccuracies concerning aids . despite the depressing subject
    matter , this film depicts a warm friendship between two boys , and will make you
    laugh as well as cry . very well-acted by all , especially joseph mazzello and brad
    renfro . the language is a little strong , though appropriate , and it's an
    entertaining and intelligent film for the whole family . but remember to have the
    kleenex ready !
```

Listing 4: Although classed as positive, the sentiment of this review is hard to distinguish, and thus its paragraph vector is mapped to the centre of the t-SNE plot (see Figure 2).

[8] In that it is non-linear and can extract very complex high-dimensional cluster relationships, including when clusters are embedded within each other. This is unlike other methods such as PCA, whose linearity hinders its decomposition ability, only explaining 10.6% of the paragraph vectors' variance in the top 2 dimensions (6.3% and 4.3% respectively).

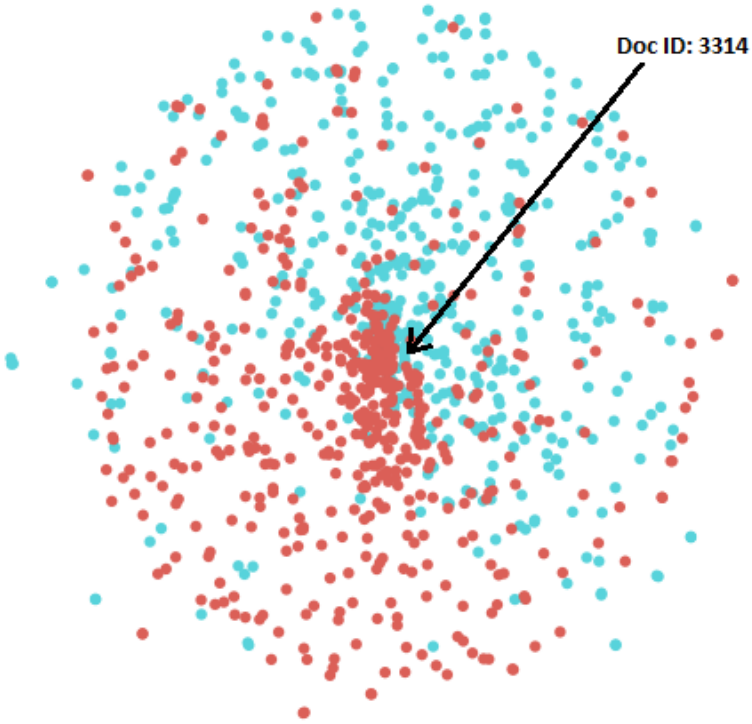

Figure 2: A plot of 1000 reviews (500 positive sentiment in cyan, 500 negative sentiment in red) using TSNE with a perplexity of 500, ran for 10,000 iterations.

4 Conclusion

To conclude, this report has demonstrated that there are several algorithms over a range of complexities that are able to classify our review datasets' sentiment with similar accuracies. Whilst it is true that the accuracy improved with model complexity, with SVMs outperforming Naive Bayes, and Doc2Vec outperforming SVMs, it has also been apparent that this additional complexity is prone to overfitting, and thus the simplest variants of each of the the three algorithms have shown to perform best (i.e. unigrams with Naive Bayes, unigrams and TF-IDF for SVMs, DBOW for Doc2Vec). However, if our training sets were a lot larger or across a less diverse vocabulary, then we would expect this to change, as the more compound features (in N-grams for example) would have higher frequency of occurrence, and thus would likely outperform the simple Bag-of-words models.

Finally, we illustrated how Doc2Vec and t-SNE can be used to visualise semantic relations and give similarity scores between reviews, which is something that the previous simpler algorithms are not well equipped to do. However, if this is not required, it could be better to choose a more simple model at the expense of a few percent accuracy, as whilst the simple models took a matter of seconds to train and run, multiple epochs of training the most complex Doc2Vec models could take up to an hour.

References

[1] Theresa Wilson, Janyce Wiebe, and Paul Hoffmann. "Recognizing Contextual Polarity in Phrase-Level Sentiment Analysis". In: *Proceedings of the Conference on Human Language Technology and Empirical Methods in Natural Language Processing*. 2005. DOI: 10.3115/1220575.1220619. URL: https://doi.org/10.3115/1220575.1220619.

[2] Laurens van der Maaten and Geoffrey Hinton. "Visualizing Data using t-SNE". In: *Journal of Machine Learning Research* 9.86 (2008), pp. 2579–2605. URL: http://jmlr.org/papers/v9/vandermaaten08a.html.

[3] Tomas Mikolov et al. *Efficient Estimation of Word Representations in Vector Space*. 2013. arXiv: 1301.3781 [cs.CL].

[4] Quoc Le and Tomas Mikolov. "Distributed Representations of Sentences and Documents". In: *Proceedings of the 31st International Conference on Machine Learning*. Proceedings of Machine Learning Research. PMLR, 2014, pp. 1188–1196.

[5] Jiwei Li et al. "Visualizing and Understanding Neural Models in NLP". In: *Proceedings of the 2016 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies*. Association for Computational Linguistics, 2016. DOI: 10.18653/v1/N16-1082. URL: https://aclanthology.org/N16-1082.

A Appendix

A.1 Naive Bayes Algorithms

Using Bayes' equation, the probability of a document d_i belonging to class $C_i \in \{+, -\}$ is given by:

$$\begin{aligned} P(C_i = +|d_i) &= \frac{P(d_i|C_i = +)P(C_i = +)}{p(d_i)} \\ \hat{C}_i &= \arg\max_{k \in \{+,-\}} P(d_i|C_i = k)P(C_i = k) \end{aligned} \tag{6}$$

for i^{th} document d_i and classification C_i. In our case we have 1000 positive 1000 negative reviews, so $P(C_i = +) = P(C_i = -) = 0.5$. By expanding the likelihood term in the above we can then express this in terms of the vectorised features $\vec{f}_i^{(j)}$ that make up document d_i:

$$\begin{aligned} \hat{C}_i &= \arg\max_k P(C_i = k) \prod_j P(\vec{f}_i^{(j)}|C_i = k) \\ &= \arg\max_k \left[\sum_j \ln P(\vec{f}_i^{(j)}|C_i = k) + \ln P(C_i = k) \right] \\ &= \arg\max_k \sum_j \ln P(\vec{f}_i^{(j)}|C_i = k) \end{aligned} \tag{7}$$

In the unigram case, the features are simply the words, as unigram models assume that the context of the words does not matter such that:

$$\begin{aligned} P(\vec{w}_i^{(1)}, \vec{w}_i^{(2)}, ..., \vec{w}_i^{(N)}|C_i) &= P(\vec{w}_i^{(1)}|C_i)P(\vec{w}_i^{(2)}|C_i)...P(\vec{w}_i^{(N)}|C_i) \\ &= \prod_j P(\vec{w}_i^{(j)}|C_i) = \prod_j P(\vec{f}_i^{(j)}|C_i) \end{aligned} \tag{8}$$

However, for bigram and trigram cases, the features become more complex. For example, for the bigram case:

$$\begin{aligned} P(\vec{w}_i^{(1)}, \vec{w}_i^{(2)}, ..., \vec{w}_i^{(N)}|C_i) &= P(\vec{w}_i^{(N)}|\vec{w}_i^{(N-1)}, C_i)P(\vec{w}_i^{(N-1)}|\vec{w}_i^{(N-2)}, C_i)...P(\vec{w}_i^{(2)}|\vec{w}_i^{(1)}, C_i)P(\vec{w}_i^{(1)}|C_i) \\ &= P(\vec{w}_i^{(1)}|C_i) \prod_{j=2}^{N} P(\vec{w}_i^{(j)}|\vec{w}_i^{(j-1)}, C_i) := \prod_{j=1}^{N} P(\vec{w}_i^{(j)}|\vec{w}_i^{(j-1)}, C_i) \end{aligned} \tag{9}$$

so now the features have become $\vec{f}_i^{(j)} = (\vec{w}_i^{(j-1)} \rightarrow \vec{w}_i^{(j)})$. (We also include the previous single word features in both bigram and trigram models, such that the bigram and trigram features are *additional* features to the previous unigram feature set.)

A.2 TF-IDF for Feature Importance Measurement

The purpose of TF-IDF is to assign an 'importance' to each feature in the feature space, so that the least important features can be dropped. To do this, TF-IDF multiplies two terms, the Term Frequency (the TF part) and the Inverse Document Frequency (the IDF part). The term frequency (TF) is review dependent, and is define for each feature m in review d as:

$$\mathrm{TF}_d(m) = \frac{\mathrm{count}(m \in d)}{\sum_{m \in d} \mathrm{count}(m)} \tag{10}$$

There is an obvious drawback to using solely TF for an importance measure - features like 'a' and 'the' occur very frequently, and thus have high TF score, despite not giving any information on the semantic content of the texts. Therefore we wish to remove these aspects, and only keep the frequent features that are semantic content related, such as 'good' and 'bad'. In order to do this, the second part, IDF, is used. IDF is defined as:

$$\mathrm{IDF}(m) = \log_{10}\left(\frac{\text{Total Number of Reviews}}{\text{Number of Reviews including } m}\right) \tag{11}$$

In contrast to the term frequency, uninformative features like 'a' are now down-weighted, as they will be in almost every review, and thus their IDF will be effectively 0. However, the shortcoming of this term is that very rare words (such as those found in diverse corpora) are scaled very highly, despite being incomparable to other words in a corpus (and thus not helpful for classification) as they only occur once or twice.

The best of both worlds can be found by multiplying to two, to give the final TF-IDF transform for each review. Consequently the highest TF-IDF scores are allocated to features which are in common enough to have high TF values, whilst not so abundant that they have negligible IDF values - such as words conveying semantic meaning in reviews.

Note that for TF-IDF in SciKit-Learn (as used in this report), divde by zero errors are avoided by redefining TF-IDF to a similar function:

$$\text{TF-IDF}_{\text{SK-Learn}} = \frac{1}{Z}\frac{\mathrm{count}(m \in d)}{\sum_{m \in d} \mathrm{count}(m)}\left[\log_{10}\left(\frac{1 + \text{Total Number of Reviews}}{1 + \text{Number of Reviews including } m}\right) + 1\right] \tag{12}$$

where Z is a normalising constant. Note that the normalisation is of particular importance for SVMs - if it were not for normalisation, then vectors with the same words and same sentiment could still be far apart due to one having many more words and so a larger magnitude. Thus normalisation normalises all magnitudes to 1, so that two reviews with the the same ratio to the same words would be mapped to the same place, regardless of review length.

A.3 The Doc2Vec Model

There are several variants of Doc2Vec model, but all take a similar methodology. The more complex form of model is the Distributed Memory (DM) model, which uses the same paragraph vector as a context (referred to as memory) for all the words in that paragraph (document in our case). The simpler form does not account for the word contexts in a document, and is called the Distributed Bag of Words (DBOW) model.

DM Architecture and Training

The algorithm itself is a self-classification algorithm, in that it uses a window of words from the document (set by the window parameter) to predict the next word in the sequence, using one-hot encoded labels for the output word. The input words and paragraphs are also one-hot encoded, before being embedded as word vectors and a paragraph vector respectively by linear transforms using a word embedding matrix W and paragraph embedding matrix V. By defining a loss function and comparing the predicted one-hot encoded outputs with the true next-in-sequence words, W and V can be learned by stochastic gradient descent.

The paragraph vectors that are outputted by the model are the embedded paragraph vectors created by the matrix product of the learned matrix, V, with the one-hot-encoded paragraph ID vector. In this way the model assumes that a document can be classed based on how the words within it are arranged, and documents with similar word contexts will get similar paragraph vector outputs.

There is a further distinction within DM models, namely DM using a mean (DM-M) or DM using concatenation (DM-C) (see Figure 3). This refers to how the paragraph and word embeddings are combined to make a prediction on the most probable next-in-sequence word- they can be averaged (DM-M) or concatenated (DM-C) to form the input layer of the next-word classifier. The concatenation has higher complexity as it gives the next-word classifier higher dimensional inputs.

DBOW Architecture and Training

In contrast to the DM model, the DBOW model now does not include word contexts, only the document context. It aims to predict all the words in the document solely from the paragraph embedding, and in this way makes the paragraph vector a representation of the words in the document. This model therefore assumes that similar class documents contain similar words, and is similar to the Naive Bayes method, except for the fact that the paragraph embedding vector is of fixed dimension (often a lot denser than Naive Bayes' sparse vectors) and is trained using SGD.

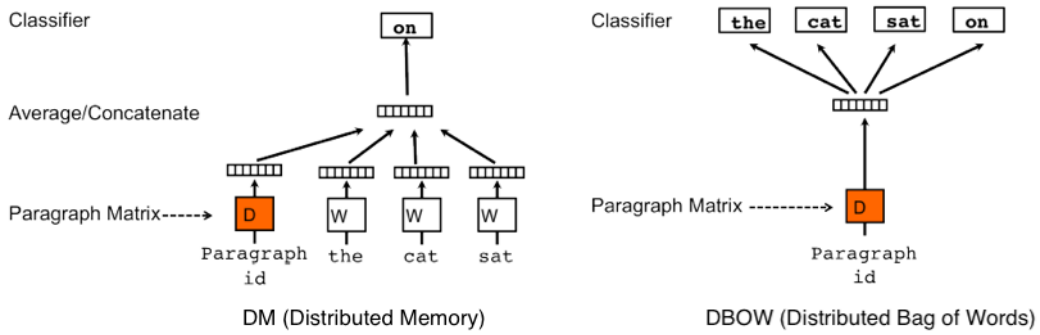

Figure 3: The architecture of two type of Doc2Vec models. (Left) Distributed memory, which uses the paragraph vectors to give a context/memory for the word model. (Right) Distributed Bag of Words, which is a simpler model that does not include word contexts, only the paragraph ID. Figure taken from [4].

Paragraph Vector Predictions

After training, both Doc2Vec models can be used to infer new paragraph vectors from test documents. In a normal neural network this would entail simply inputting the test data into the pre-trained network with fixed parameters and collecting the outputs, however, for Doc2Vec models it is slightly more complicated because the output vector is part of the internal model architecture. In this case, the new test documents' IDs are appended to the matrix containing the one-hot-encoded paragraph IDs used in training, and the paragraph embedding matrix, V, is extended to account for this. SGD is then run again at test-time as before during training, except that at test-time all other parameters are held constant (namely the word embedding matrix, W, and the activation function parameters). We are therefore able to specify testing epochs that dictate how many iterations of SGD are used to train the new V matrix, with more epochs providing better predictions, but at the cost of time complexity (as the size of V is large due to now including all training cases plus test cases).