UNIVERSITY OF CAMBRIDGE

MPhil in Machine Learning and Machine Intelligence 2022-2023

Module Coursework Feedback

Module Title: Natural Language Processing

Module Code: MLMI13

Candidate Number: K5002

Coursework Number: 1

I confirm that this piece of work is my own unaided effort and adheres to the Department of Engineering's guidelines on plagiarism. √

Date Marked: Click here to enter a date. Marker's Name(s): Click here to enter text.

Marker's Comments:

This piece of work has been completed to the following standard *(Please circle as appropriate)*:

	Distinction			Pass			Fail (C+ - marginal fail)		
Overall assessment (circle grade)	**Outstanding**	**A+**	**A**	**A-**	**B+**	**B**	**C+**	**C**	**Unsatisfactory**
Guideline mark (%)	90-100	80-89	75-79	70-74	65-69	60-64	**55-59**	**50-54**	**0-49**
Penalties	**10% of mark for each day, or part day, late (Sunday excluded).**								

The assignment grades are given **for information only**; results are provisional and are subject to confirmation at the Final Examiners Meeting and by the Department of Engineering Degree Committee.

MLMI13 - Sentiment Analysis of Movie Reviews

K5002 - 1999 words

December 6, 2022

1 Introduction

Sentiment analysis is one of the most prominent topics within the field of natural language processing. It aims to extract the emotion (sentiment) behind a body of text. In this paper we will investigate multiple approaches to solving this task, such as lexicographical, ngram and SVM models. As well as the utility of document embeddings provided by Doc2Vec, Bert and DistilBert.

2 0.1 Lexicon Based approach

One of the simplest approaches to sentiment analysis is through lexicons, first devised by Stone[10] they consist of a dictionary of polarised words which are matched against a corpus of text to obtain a document-level sentiment score. Wilson[12] dubs these scores as a *"prior polarity"* `{positive negative, both, neutral}` since the words are not contextualised. Lexicons can be extended through *"reliability classes"* denoting the strength `{strongsubj, weaksubj}`. This extension is constructed from a corpus of contextualised polarity scores, indirectly considering context in the lexicon.

Solely using prior polarity lexicons achieves 63% accuracy over a corpus of news articles, and 65.7% when using reliability classes. It was found that the news articles contain a bias towards being positive accordingly a constant bias of -8 was used to compensate. A linear correlation between bias and the number of tokens in a review was also found (Figure 1). With a -0.05946 compensation for each token, and -6.9 bias (Figure 2).

Model -	Reference [12]	Reference +Reliability [12]	constant bias	constant bias + Reliability	linear bias	linear bias + Reliability
Accuracy	63.0%	65.7%	0.57%	61.0%	67.3%	69.1%
F1	-	-	0.7229	0.7547	0.8045	0.8169

Table 1: Lexicographical model performances compared to the reference implementation

Using the two-tailed sign test[11] the performance improvement achieved by adding *reliability classes* is significant at the $\rho = 4.676E-5 < \alpha = 5\%$ level. Similarly adding the linear bias significantly improves performance $\rho = 1.714752E-4 < \alpha = 5\%$

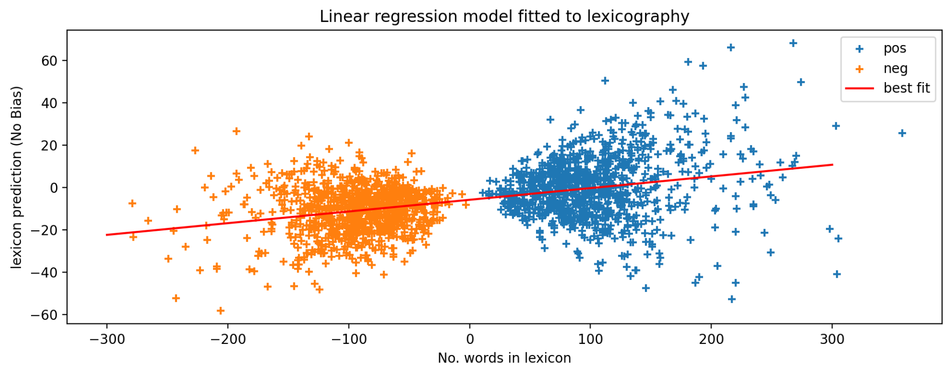

Figure 1: Linear correlation between bias and the number of tokens verified via Spearman statistic $\rho = 0.4472$

$$S_{weighted}(w_1, w_2, ..., w_n) = -6.9 + \sum_{i=1}^{n} SLex[w_i] - 0.05946 \tag{1}$$

Figure 2: Lexicon using linear bias and magnitude (Reliability)

3 1.0 - Bag of words

Although lexicographical approaches are very simple and generalise well, their performance is limited since they fail to model the structure of a specific domain. Naive Bayes is a probabilistic classifier that builds a model for the likelihood of a class generating some input data (generative model) [5]. Providing a more analytical solution compared to lexicons.

$$\begin{aligned}
\hat{c} &= arg \max_{c \in C} P(c|d) = arg \max_{c \in C} P(d|c)P(c) \\
&= arg \max_{c \in C} P(f_1, f_2, ..., f_n|c)P(c) \qquad \text{(representing document with features)} \\
\Rightarrow \hat{c}_{NB} &= arg \max_{c \in C} P(c) \prod_{f \in F} P(f|c) \qquad \text{(naive Bayes assumption)} \\
\hat{c}_{NB} &= arg \max_{c \in C} \log P(c) \sum_{f \in F} \log P(f|c) \qquad \text{(naive Bayes assumption)}
\end{aligned} \tag{2}$$

Where each f_i is a word w_i from a document d. To build this model we must extract $P(c)$ and $P(w_i|c)$ from our corpus, $P(c)$ is simply the number of documents with that class divided by the total number of documents.

$$\hat{P}(c) = \frac{N_c}{N_{doc}} \tag{3}$$

However $P(w_i|c)$ is more nuanced, we can crudely compute it in the same way as $\hat{P}(c)$. (V is the vocabulary of words in the training and test data)

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

However there is no guarantee that $count(w_i, c)$ is non-zero, occurring when a word in the test corpus of a given class does not appear in the training corpus. This causes the 50% accuracy seen in the first model (equivalent to random guessing), since the majority of products $\prod_{f \in F} P(f|c)$ become zero thanks to a single zero $P(f|c)$.

This can be easily resolved with (Laplace) *smoothing*[5] by introducing a prior count, preventing 0 probabilities.

$$\hat{P}(w_i|c) = \frac{count(w_i, c) + 1}{|V| + \sum_{w \in V} count(w, c)} \tag{5}$$

Implementing this results in a significant performance improvement, correctly classifying 78% of the documents, on par with the reference implementation's 78.7% (All results use 10-fold cross validation to reduce type III errors). However we are also using an unnecessarily large vocabulary of words where many have the same meaning thanks to the various possible conjugations of a given word. By *stemming* words down to their root, removing morphological and inflexional endings [9] (such as `-ED, -ING, -ION, -IONS` [4]) we can reduce the number of features from 53,555 to 32,404 however this reduction in information does cost 0.8% accuracy although this difference is statistically insignificant.

Next we can relax the naive Bayes assumption to allow conditional dependency between n consecutive words, allowing us to capture context between words such as negations. However this comes at a high cost with the number of features being in the order of $O(V^n)$ (n=2 for bigrams, n=3 for trigrams). In reality not all these permutations are present in the corpus, combined with stemming, the number of bigrams is 500k and trigrams is 1.46M. Using a hash table this is tractable and significantly improves performance w.r.t. bigrams (similarly to the reference [7]) and considerably improves performance w.r.t. trigrams.

Model	Unigram			Uni + Bigram	Uni+Bi+Trigram
Smoothed	No	Yes	Yes	Yes	Yes
Stemmed	No	No	Yes	Yes	Yes
Accuracy	$50.1 \pm 1.1\%$	$78.0 \pm 1.8\%$	$77.2 \pm 1.8\%$	$82.5 \pm 2.0\%$	$84.2 \pm 1.7\%$
F1	0.6680	0.8764	0.8716	0.9038	0.9142
Significant	-	Yes	No	Yes	No
Features	52,555	52,555	32,404	500,086	1,462,605
Reference [7]	-	78.7%	-	80.6%	-

Table 2: Cross validated BoW performance

4 SVM

The assumption of conditional independence that naive bayes makes, introduces skew in most real life corpera. SVM's sacrifice interpretability to forgo this assumption. They function via the following mechanisms:

$$y_i = \underline{w}^\top \underline{x}_i + b \tag{6}$$

$$y_i(\underline{w^T x_i} + b) \geq 1 \quad i = 1, ..., m \tag{7}$$

Where $\underline{x}_i$ is a feature vector (words) and y_i is a class label for a given document and $\underline{w}$ are the weights. Basic SVM's (Eqn 6) assume that that a corpus is linearly separable by $\underline{w}$ in $\mathbb{R}^D$ (where D is the dimensionality of the input vectors $\underline{x}_i$). Achieved by satisfying the constraint in equation 7. We can turn this into a minimisation problem via the objective function in equation 8

$$\min_{\underline{w}, b} \frac{1}{2} |\underline{w}|^2 \tag{8}$$

The constraint 7 and objective function 8 can be combined into the Lagrangian function 9 and it's maximisation 10:

$$L(\underline{w}, b, \alpha) = \frac{1}{2}|\underline{w}|^2 - \sum_{i=1}^{m} \alpha_i (y_i(\underline{w^T x_i} + b) - 1) \tag{9}$$

$$arg \max_{\alpha} \sum_{i=1}^{m} \alpha_i - \frac{1}{2} \sum_{i,j=1}^{m} \alpha_i \alpha_j y_i y_j (\underline{x}_i \cdot \underline{x}_j) \tag{10}$$

Which has the solution 11:

$$\underline{w} = \sum_i \alpha_i y_i \underline{x}_i \tag{11}$$

The Lagrangian multipliers α_i are only non-zero for the *support vectors*, the data points that define the decision boundary. Accordingly only those points closest to the decision boundary define α_i and consequently the solution to $\underline{w}$. (Figure 3)

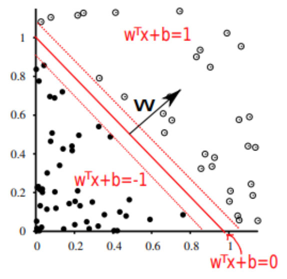

Figure 3: SVM Decision boundary and adjacent support vectors

SVM's remove the requirement for linearly separable data by introducing basis functions $\phi(\underline{x}) = \underline{x}$. Which non-linearly transform the input data, they need only be *continuously differentiable*, e.g.

Polynomial - $K(\underline{x}_i, \underline{x}_j) = (\underline{x}_i \cdot \underline{x}_j)^d$

Gaussian - $K(\underline{x}_i, \underline{x}_j) = exp\left(-\frac{|\underline{x}_j - \underline{x}_j|_2^2}{2\sigma^2}\right) \qquad \sigma \in \mathbb{R}$

hyperbolic tangent - $K(\underline{x}_i, \underline{x}_j) = tanh(k(\underline{x}_j \cdot \underline{x}_j) + \Theta) \qquad k, \Theta \in \mathbb{R}$

radial basis - $K(\underline{x}_i, \underline{x}_j) = exp\left(-\frac{|\underline{x}_j - \underline{x}_j|^2}{2s^2}\right)$

None of these kernels were found to significantly improve performance versus one another.

The final consideration when using SVM's is the prepossessing and normalisation of the inputs $\underline{x}_i$, Pang-et-al.[7] uses a count vectoriser which is normalised to length 1 (A $V \times 1$ vector where the i^{th} element denotes the frequency of the word in the document). Taking inspiration from Das-Chakraborty[1] we used TF-IDF as they report 7.01% greater accuracy versus the rerference implementation[7]. TF-IDF combines the count vectoriser with a scaling which suppresses common words.

$$\begin{aligned} \text{TF-IDF}(t,d,D) = tf(t,d) \times idf(t,D) &= \frac{f_d(t)}{\max\limits_{w \in d} f_d(w)} \times \ln\left(\frac{|D|}{|\{d \in D : t \in d\}|}\right) \\ f_d(t) &- \text{frequency of term in document d} \\ d &- document \\ D &- \text{corpus of documents} \\ score_T(s_i) &= \sum_{t \in s_i} \text{TF-IDF}(t,d,D) \end{aligned} \tag{12}$$

Pant-et-al have also annotated each word with a *part of speech* tag. There are 36 unique tags which represent local relationships between words for example "*strongsubj/weaksub features are true if the word and its parent share an adj, mod or vmod relationship*"[12] (Figure 4). These features allow linguistic modification rules to be encoded against each word in the corpus. Further they can be used to restrict the feature space to only those words that contribute to the sentiment of a document. We extracted nouns, adjectives, adverbs and *Verb - base form* NN, JJ, RB, VB[8]. Adding these POS tags does not significantly improve performance and increases the number of features. However filtering on these tags does not impact performance but almost halves the number of features.

Interestingly none of the SVM models developed in this chapter significantly improve upon the unigram +bigram+trigram naive Bayes model, however, do achieve a 135 fold reduction in complexity. Accordingly the SVM model is superior to the NB model, provided explainability is not required. Further as expected all models surpass the reference implementation, thanks to the use of the TF-IDF vectoriser.

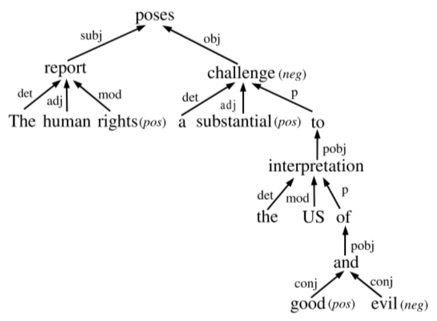

Figure 4: Pang-et-al[7] part of speech features

Model	SVM	SVM + POS	SVM + POS + POS Filter				Reference [7]
Kernel	RBF	RBF	RBF	Linear	Poly	Sigmoid	RBF
Accuracy	$86.9 \pm 1.4\%$	$87.3 \pm 1.4\%$	$87.4 \pm 2.0\%$	$87.4 \pm 2.1\%$	$85.4 \pm 1.4\%$	$86.5 \pm 2.0\%$	82.9%
F1	0.9296	0.9319	0.9325	0.9322	0.9213	0.9273	-
Features	17684	20795	10514	10514	10514	10514	16165

Table 3: SVM performance - none of which provide a significant improvement vs the Uni+Bi+Trigram NB model

5 Doc2Vec

To understand doc2vec we must first consider it's origins, word2vec. Word2vec[6] aims to provide a small embedded representation of a large vocabulary. Where *multiple degrees of similarity* are maintained between word embeddings, for instance similar words will have a similar embedding, but further than this, algebraic operations also hold: vector("King") - vector("Man") + vector("Woman") $\approx$ vector("Queen")[6]. This is achieved by either of two unsupervised models *Continuous Bag of Words* (CBOW) and *Continuous skip-gram* (Skip-gram).

CBOW is trained by repeatedly sampling 2N+1 contiguous words. The N *history* and N *future* (context) words are passed through a non-linear *projection* layer, and the outputs are summed forming a $D \times 1$ embedding. Then a hidden layer maps from the embedding to the target word. This forces the projection layer to have a *continuous distributed representation of the context* [6].

The second model, skip gram, reverses CBOW, reconstructing context words given a single word. This word is passed through the projection layer, then a single large hidden layer maps from the word embedding to a vector encoding the C context words. Using a single layer allows for the representation of word order.

Both models are then trained using stochastic gradient decent via backpropagation. CBOW is 3.6 faster to train but 17.2% less accurate [6].

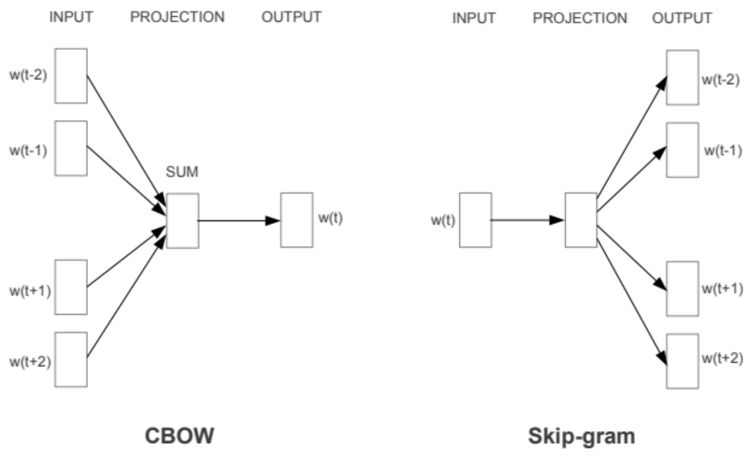

Figure 5: Mikolov et al. [6]

Doc2Vec operates similarly but describes entire documents/*paragraphs*. It also has 2 models *Paragraph vector: distributed memory* (PV-DM) which predicts multiple missing words, given a context vector and paragraph ID, similar to CBOW. And *distributed bag or words* (PV-DBOW) which predicts many context words from a document given a document ID, similar to skip gram.

In PV-DM each paragraph ID is passed through a non-linear paragraph-projection layer forming the paragraph embedding vector D. Similarly each word is embedded forming word embeddings W. These vectors are either averaged or concatenated together (we found concatenation performs best) then a hidden layer projects from the embedding to the missing word(s). The addition of a paragraph vector vs CBOW allows the semantics of words to be represented plus the ordering of words within the paragraph. Consequently PV-DM generalises well despite it's high dimensional representation.[3]

In PV-DBOW we only use the document embedding D, then predict a number of context words sampled from the document. This model is far simpler and requires fewer parameters since the word-projection layer parameters are not required.

Quoc Le & Mikolov[3] found the best results when PV-DM and PV-DBOW are combined. Similarly when combining PV-DM and PV-DBOW by concatenating their embeddings, we obtained a significant performance improvement of 4.8% achieving 88.1% accuracy!

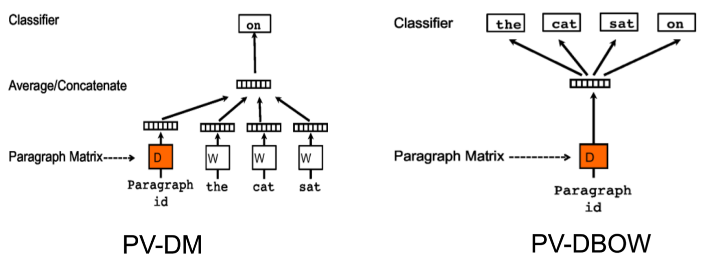

Figure 6: Quoc Le & Mikolov [3]

In both cases the dataset of IMDB movie reviews was preprocessed by removing stop words and stemming the vocabulary. Performance was measured by training an SVM to classify document sentiment from the document embeddings, providing a measure for how much information was encoded within the document embeddings. Hyper-parameter tuning follows exploration-exploitation theory varying a single parameter at a time then exploiting the best parameter. Training took approximately 40 minutes for each hyper-parameter setting running on a 16 core machine, accordingly only 14 models were fully trained and K-fold cross validation was not viable (Table 4).

Epochs	Model	Alpha	vector size	window size	F1	Accuracy
50	PV_PM	0.025	100	30	0.7699	62.5%
50	PV_PM	0.025	100	5	0.8289	70.8%
100	PV_DBOW	0.01	300	30	0.8328	74.2%
100	PV_DBOW	0.005	300	30	0.8861	80.1%
50	PV_DBOW	0.0025	300	30	0.9032	82.2%
50	PV_DBOW	0.001	300	30	0.8346	71.6%
50	PV_DBOW	0.0025	300	5	0.8495	73.8%
50	PV_DBOW	0.0025	50[3]	30	0.8710	77.2%
50	PV_DBOW	0.0025	100	30	0.8888	80.0%
50	PV_DBOW	0.0025	300	15	0.8933	80.7%
50	PV_DBOW	0.0025	300	60	0.9027	82.3%
500	PV_DBOW	0.0025	300	20	0.9069	83.0%
300	PV_DBOW	0.0025	300	30	0.9144	**83.3%**
50	BOTH	0.0025	300/50	30/5	0.9370	**88.1%**

Table 4: Doc2Vec performance

Where the vector size is the size of the document embeddings D and word embeddings W, the window size is the maximum distance between words in the context vector and alpha is the learning rate. The learning rate was *cooled down* linearly during training to $\alpha/10$, ideally a more reactive learning rate would be used based on the current losses first and second moments as well as using momentum, such as the Adam optimiser. Unfortunately the training loss is not stored in Gensim's doc2vec implementation preventing the use of an optimiser.

Overall only by combining both Doc2Vec models did we achieve a significant performance improvement compared to the naive Bayes Unigram+Bigram+Trigram model, being 88.1% accurate compared to the 84.2% achieved by naive Bayes.

BERT[2] is a transformer architecture using an attention mechanism intended to understand language, it's trained via a combination of Masked LM (predicting missing words in sentences) and NSP (Next

Sentence Prediction). The trained model produces an embedding that captures the meaning of a sentence and so transfer's well between a variety of tasks. We used a pre-trained model[2] trained on 3.3 billion words to produce document embeddings for each review, before using an SVM to classify the embedding's sentiment. We also used a newer variant of BERT, *DistilBert* which simplifies the architecture and executing ∼2x faster. We used a pre-trained version trained on 11,038 books and 2.5 billion words from Wikipedia.

Model	Training Time	F1	Accuracy
BERT	11:56	0.8400	72.4
DistilBert	6:21	0.8465	73.38

Table 5: BERT & DistilBert + SVM performance - time in minutes, embedding size=768

Overall both architectures perform significantly worse than our Doc2Vec implementation, we believe this is due to using an SVM to classify the embeddings. Had a multi-layer neural network been used more information could have been extracted from the embedded representation.

5.1 Embedded representations

Theoretically the document embeddings produced by Doc2Vec should be similar for similar reviews. To test this analytically we have computed the cosine similarity between the Doc2Vec embeddings of all the possible combinations of 1,000 reviews. For reference we have also done so for the TF-IDF representation as well as BERT and DistilBERT embeddings.

A	The acting was bad, script was bad and ending was just terrible...the only good comment i could make about this movie would be the special effects and make up...but apart from that...this movie would be one of the worst movies of 2001... why on earth did they have to remake such a perfect movie and ruin it...why!!!!			
B	This movie changes its way a third of the way in.its totally pointless boring and stupid. i hated this movie so much that i will never watch it again.some bad films can be really funny. this is just a British art house picture that should never of been made.1 out of 10			
	D2V	TF-IDF	BERT	DistilBERT
Cosine Similarity	0.9621	0.1005	0.7774	0.9819

Table 6: Two similar reviews according to Doc2Vec, both are negative

A	No offense to anyone who saw this and liked it, but I hated it! It dragged on and on and there was not a very good plot, also, too simple and the acting was so so... I would give this snorefest a 2 at the most			
B	I love this movie. I mean the story may not be the best, but the dancing most certainly makes up for it. You get to know a little bit about each character the way THEY want you to learn about them. I just think that you won't like this movie unless you're into Broadway..			
	D2V	TF-IDF	BERT	DistilBERT
Cosine Similarity	0.9550	0.0569	0.9396	0.9468

Table 7: Two similar reviews according to Doc2Vec, A is negative, B is positive

In Table6 review A talks about acting, scripts, endings and special effects while review B indirectly talks about the plot, doubts it's humour and complains about the studio house. Despite the lack of topic overlap the Doc2Vec embedding is almost identical. However, we can see that the Doc2Vec embedding is reasonable as both BERT and DistilBERT also have similar embeddings. The same holds to an even greater extent in Table7 given the reviews even have opposing sentiment. Accordingly Doc2Vec embeddings are not close in space to their most critical words exclusively, this can be verified by the large difference in TF-IDF scores (as this indicates the overlap of vocabulary of the two reviews). Further we can show that similar embeddings are almost never spatially close to one another via Table8:

D2V	**TF-IDF**	**BERT**	**DistilBERT**
0.9621	0.1005	0.7774	0.9819
0.9550	0.0569	0.9396	0.9468
0.9509	0.0634	0.8903	0.9774
0.9496	0.0427	0.9581	0.9918
0.9494	0.0754	0.9608	0.9403
0.9489	0.1181	0.9432	0.9959
0.9479	0.0450	0.9478	0.9427
0.9472	0.0645	-0.2965	0.8581
0.9471	0.0438	0.9763	0.9827
0.9466	0.1337	0.8022	0.9384
0.9466	0.4879	0.9695	0.9035

Table 8: Summary of top 10 cosine similarity scores

Table 8 shows that there is good consensus between embedding schemes with the exception of TF-IDF. This is significant as TF-IDF encodes the word counts in a review, therefore the embeddings are **not close in space according to the most critical words** or even any words for that matter.

6 Conclusion

Overall Doc2Vec provided the most significant results, correctly classifying 88.1% of the test examples. Naive Bayes provide the best results compared to their implementation's complexity but uses an extortionate number of features. SVM provides the most compact representation using just 10,514 features to obtain near leading performance. The BERT based architectures were the most surprising, given the large number of parameters better results would be expected, however the implementation could be better optimised for these architectures in future research.

Model	F1	Accuracy
Lexicon	0.8169	69.1%
BERT	0.8400	72.4%
DistilBert	0.8465	73.38%
Uni+Bi+Trigram	0.9142	84.2%
SVM	0.9319	87.4%
Doc2Vec	0.9370	88.1%

Table 9: Model performances

References

[1] Bijoyan Das and Sarit Chakraborty. *An Improved Text Sentiment Classification Model Using TF-IDF and Next Word Negation*. 2018. DOI: 10.48550/ARXIV.1806.06407. URL: https://arxiv.org/abs/1806.06407.

[2] Jacob Devlin et al. *BERT: Pre-training of Deep Bidirectional Transformers for Language Understanding*. 2018. DOI: 10.48550/ARXIV.1810.04805. URL: https://arxiv.org/abs/1810.04805.

[3] Quoc V. Le and Tomas Mikolov. *Distributed Representations of Sentences and Documents*. 2014. DOI: 10.48550/ARXIV.1405.4053. URL: https://arxiv.org/abs/1405.4053.

[4] M.F.Porter. *An algorithm for suffix stripping*. https://tartarus.org/martin/PorterStemmer/def.txt. (Accessed on 12/04/2022). July 1980.

[5] Daniel Jurafsky & James H. Martin. *Speech and Language Processing*. https://web.stanford.edu/~jurafsky/slp3/. (Accessed on 12/04/2022).

[6] Tomas Mikolov et al. *Efficient Estimation of Word Representations in Vector Space*. 2013. DOI: 10.48550/ARXIV.1301.3781. URL: https://arxiv.org/abs/1301.3781.

[7] Bo Pang, Lillian Lee, and Shivakumar Vaithyanathan. "Thumbs up? Sentiment Classification using Machine Learning Techniques". In: *Proceedings of the 2002 Conference on Empirical Methods in Natural Language Processing (EMNLP 2002)*. Association for Computational Linguistics, July 2002, pp. 79–86. DOI: 10.3115/1118693.1118704. URL: https://aclanthology.org/W02-1011.

[8] *Penn part-of-speech tags*. https://cs.nyu.edu/~grishman/jet/guide/PennPOS.html. (Accessed on 12/04/2022).

[9] Martin Porter. *Porter Stemming Algorithm*. https://tartarus.org/martin/PorterStemmer/. (Accessed on 12/04/2022). Jan. 2006.

[10] Philip J. Stone et al. *The General Inquirer: A Computer Approach to Content Analysis*. Cambridge, MA: MIT Press, 1966.

[11] Larry E. Toothaker. "Book Review : Nonparametric Statistics for the Behavioral Sciences (Second Edition): Sidney Siegel and N. John Castellan, Jr. New York: McGraw-Hill, 1988, 399 pp., approx. $47.95". In: *Applied Psychological Measurement* 13.2 (1989), pp. 217–219. DOI: 10.1177/014662168901300212. eprint: https://doi.org/10.1177/014662168901300212. URL: https://doi.org/10.1177/014662168901300212.

[12] Theresa Wilson, Janyce Wiebe, and Paul Hoffmann. "Recognizing Contextual Polarity in Phrase-Level Sentiment Analysis". In: *Proceedings of Human Language Technology Conference and Conference on Empirical Methods in Natural Language Processing*. Vancouver, British Columbia, Canada: Association for Computational Linguistics, Oct. 2005, pp. 347–354. URL: https://aclanthology.org/H05-1044.