A Brief Analysis of the Application of Different Pre-processing methods and Neural Networks to the Sentiment Classification of Movie Reviews

s1712247 **s1867849** **s1910176**

Abstract

This report applies different pre-processing and neural network learning methods to analyse which is best for the sentiment prediction of movie reviews. We focus on the optimum encoding method for accurate prediction. The highest performing encoding was found to be a count vector, which displays the best overall performance when paired with a CNN. We analyse and critically discuss our findings.

1 Introduction

Sentiment analysis of movie reviews is a hugely effective tool for gauging the opinion of the audience. [21] In turn, the general opinion of the populous is often a decisive factor in a person's decision to watch a movie. More generally, in our digital, social-media-run world, the knowledge of people's opinions is an exceedingly valuable currency. A less obvious example than movie reviews is sentiment analysis used in the financial industry - where it is employed to analyze news and other media to determine whether it is positive or negative, faster than any individual could, allowing for a competitive advantage.

This report explores and analyses the use of multiple prepossessing and neural network methods to discern sentiment using only text from the review. There are numerous studies with similar questions, for instance [21] [4] [18], which each use the same or similar data sets.

Our data is from the Stanford Sentiment Treebank (SST) [15]. It contains 10605 original snippets from Rotten Tomatoes movie reviews; a dictionary file, containing 239232 phrases parsed by the Stanford Parser from the original snippets; and a sentiment labels file, giving the human-allocated continuous sentiment number (floats between 0. and 1.) corresponding to each phrase in the dictionary. We convert the sentiment scores into classes: Very Negative, Negative, Neutral, Positive and Very Positive; and hence convert the problem into one of ordinal classification.

2 Cleaning the Corpus

Before implementing any pre-processing or learning methods, we converted the entire original snippets and dictionary corpora to lowercase and removed all special characters, punctuation and numbers. Additionally, using the Natural Language Toolkit library [12], we removed all 'stop words' (the, and, I, etc) and lemmatized each individual word in the corpus. Lemmatizing in this case means to convert a word to its grammatical root (for instance, the root of "builds" and "built" is "build".) By cleaning in this way, we aimed to remove from the data as much unhelpful "noise" as possible, to capture the sentiment of a review more easily. By removing vocabulary such as stop words, however, we do risk losing contextual information that could help encode semantic meaning.

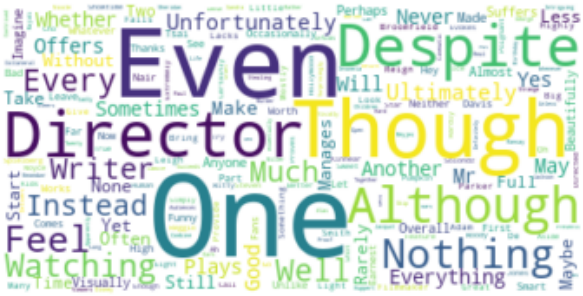


Figure 1: A word cloud showing the most used words inside the movie reviews data set.

3 Exploratory Data Analysis

The clean dictionary is now a data frame with 149902 rows and 2 columns: phrase, and sentiment category. Some of the most common words can be spotted in figure 1. It is apparent that although the most common words contain little intrinsic sentiment, some words that cause a semantic shift are still extremely common. Not only that, Table 1 suggests the even most common words among different sentiment classes have a small but distinct sentiment correlation.

	1st	**2nd**	**3rd**	**4th**	**5th**
Very negative:	Like	Bad	Character	Time	Minute
Negative:	Like	Character	Story	Much	Time
Neutral:	Like	Story	Character	Time	Make
Positive:	Good	Make	Character	Story	Like
Very Positive	Performance	Good	Best	Well	Funny

Table 1: The 5 most commonly used words (after "movie, "film" and "one" were removed) for each sentiment class.

Plotting the frequency distribution of words in both the dictionary corpus and the original snippets. Figure 2 shows the word frequency distributions of both (clean) corpora. As one might expect, the original snippets follow the general rule for word, frequency in large bodies of text - most of the corpus is made up of highly infrequent words, those appearing between 1 and 10 times, with a few highly common ones. The parsed dictionary file contains a lot of repeats of subphrases, and thus has a large majority of words appearing between 10 and 100 times.

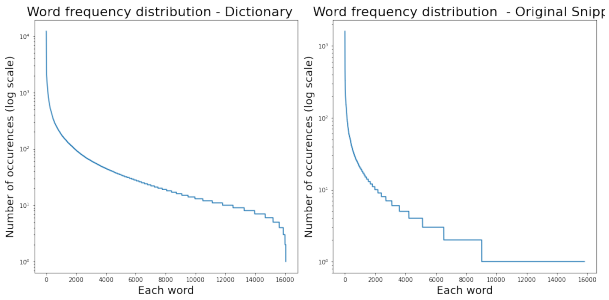


Figure 2: Frequency distributions of words in dictionary corpus (right) and the original review corpus (left).

4 Data Pre-processing - Encoding Phrases

Count Vector Encoding

One of the most basic bag-of-words encoding methods is count vector encoding. This means converting each phrase in the dictionary to a vector with a dimension equal to the number of different words in the entire dictionary. In each phrase vector, an entry is the number of times the corresponding word appears in the given phrase. The matrix made up of these count vectors is often termed a "sparse matrix", as by nature it contains mostly zeros. The resultant matrix encoding has dimensions (152018 x 17719).

TF-IDF

A second bag-of-words method we used to encode the data was by term frequency inverse document frequency (TF-IDF) scores. The score for each word is calculated as

$$\text{TF}(\text{word}, \text{phrase}) * \text{IDF}(\text{word})$$

where

$$\text{TF}(\text{word}, \text{phrase}) = \frac{(\text{word's frequency in the phrase})}{(\text{number of words in the phrase})} \text{[ref 23]}$$

$$\text{IDF}(\text{word}) = \ln\left(1 + \frac{(\text{total number of phrases})}{(\text{number of phrases containing word})}\right)$$

This technique embeds the **relative** frequency of a word in a phrase - thus giving less weight to higher frequency words and more weight to those of lower frequency in the embedding. The embedding uses the top 8000 most frequent words in the corpus to encode phrases - thus the shape of the resultant encoded array has dimensions (149901 x 8000).

Word2Vec

It is a known fact in the field of computational linguistics that a word's contextual information gives a decent approximation of its semantic meaning. This is because semantically similar words normally appearing in similar contexts[2]. Word2vec takes advantage of this fact by using a words contextual "window" to encode it as a vector. This vector is made up of weights tuned in a training process such that word vectors with greater semantic similarity have a lesser distance between them in hyperspace. Using the package Gensim[**?**], we trained a Word2vec model on the text file of original, un-parsed review snippets in order to maximise contextual information given to the algorithm. According to [7], greater accuracy is achieved for short input texts when the contextual window for each word appraised by the algorithm is small. We used the Continuous Bag of Words (CBOW) Word2vec method over Skip-gram, as its faster and more reliable for large data sets[9]. It was noticed that the presence of a few very high frequency words (such as "film" and "movie") were appearing in context of so many words that many were being encoding being encoded as similar regardless of semantic meaning. To combat this, the top 50 most frequent words in the corpus were also removed before training. Additionally, our model only contains words that appear in the corpus at least 5 times - this is because words appearing too few times simply become noise in the data. A shortfall of removing these words from the model is that a lot of information is lost. As seen in figure 2, words appearing under 5 times make up around two thirds of the vocab in the original snippets corpus, and removing high frequency words such as "good" and "well" gets rid of potentially important semantic information from many individual reviews. To get phrase encodings, we sum over all the word vectors in said phrase. This study [11] shows that summing word vectors gives better performance than any other basic operation. We encode 500 features for each word (and thus each phrase), and include all phrases containing at least one word from the model. We ran the model for 4 iterations, as we found that for any more than that we saw overtraining - in the form of all similarity scores being exceedingly high. The resulting matrix is a (207188 x 500).

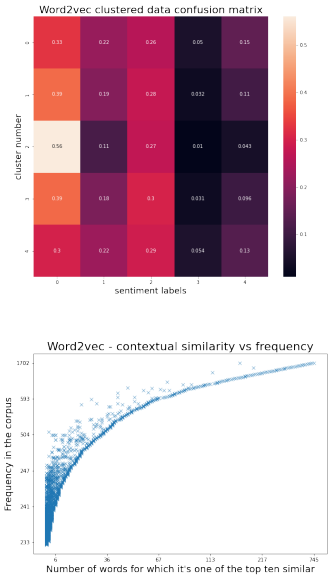
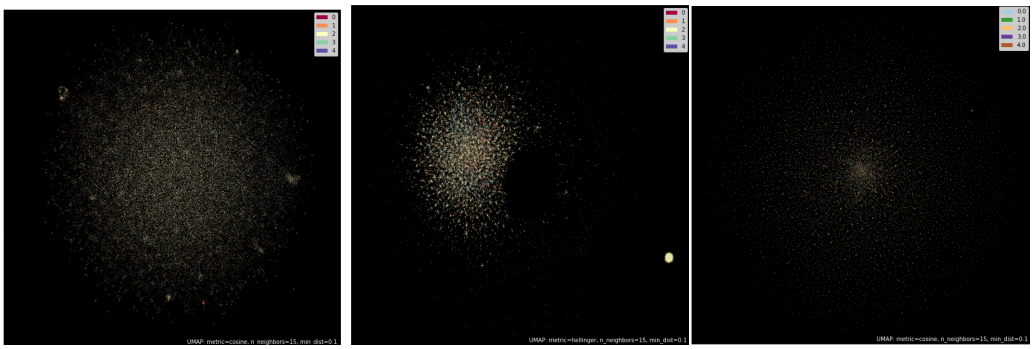


Figure 3: Top, clusterng by Kmeans of the word2vec encoded data. Bottom, number of words for which it's one of the top ten similar, plotted against that word's frequency in the corpus.

As semantic word embedding is an unsupervised process, it is difficult to evaluate its performance for a specific data set - however, we can examining the output of the resultant model. At first we see some promising results, for instance, among the closest vectors to "kiss" was "romantic". However, despite clearly a small amount of semantic information being stored, and despite removing high frequency words, we still see a clear correlation between a words frequency in the overall corpus and its similarity score to other words (see Figure 3.)

We clustered the data into five groups using Kmeans from scikitlearn [10], see Figure 3. We see very little correlation between how the Kmeans model clusters the phrases, and their sentiment labels, with the exception of most of "cluster 2" belonging to the "very negative" category. This could suggest that



(a) Count Array (b) TF-IDF (c) Word2vec

Figure 4: UMAP projections of the encoding created by each preprocessing method.

very little semantic information has been encoded, and warns against the use of clustering classifying methods like K-Nearest-Neighbours (KNN), used for text sentiment classification in literature [18][4].

Visualisation

Having projected the three encodings down into two dimensions using UMAP [8], see figure 4, we see very little to no evidence of class separation for any pre-processing method. Some small micro clusters of same sentiment appear in the projection of TF-IDF, even fewer for the count array and none for Word2vec. All the encodings seem to be quite normally distributed - Word2vec the most and TF-IDF the least. Based on this observation, and our earlier analysis, one might reasonably suggest that distance from the center in the Word2vec plot correlates strongly with frequency in the text. The TF-IDF encoding seems to be the most tightly grouped in general, and forms the most distinct shape - followed by the count encoding. We might expect that these encodings contain more information than Word2vec going into the training phase. What we see here suggests a method like KNN would be ill-suited to our task - or certainly these particular encodings.

5 Sentiment Classification - Neural Network Prediction

Methods

All models mentioned below were trained via Keras[5] and Tensorflow[1] packages, using 20% testing-training split and a 30% validation split. All models were tuned to the same hyperparameters for each encoding, to ensure a consistent and fair comparison between preprocessing methods, even if some of the models could have been tailored further to perform better. For each method an "adam" optimiser was used.

Dense Neural Network

A dense neural network is feed-forward perceptron, where every node is fully (densely) interconnected. Input is passed through a series of hidden layers of densely connected nodes to compute an output[14]. These models are highly prone to overtraining, and thus regularisers and dropouts often have to be used to reduced. We trained the model for 20 epochs.

1D Convolution Neural Network

Convolutional Neural Networks (CNNs) are highly effective for computer vision and speech recognition[3], due to their ability to learn generalised features of data and recognise long term dependencies[4] lack of sensitivity to small perturbations in input. They have also been widely used in the field on text sentiment analysis [17][3][20][19]. The network structure is as follows. Features of phrases are learnt by applying a "filter" vector to every possible contextual window of the phrase, producing a *feature map*; then applying a "max pooling layer", which combines close values in the feature map by representing them as their common maximum. The model aims to capture the most "influential" feature for each phrase's feature map [3], with the pooling layer in place to create a

robustness to small input perturbations. Due to its very large computational cost, our CNN was only trained for 10 epochs, less than that required for full optimisation.

Other Neural Networks

Although found later to be sub-optimal for our specific task in terms of achieved accuracy and processing time, we have implemented both a Reccurent Neural Network (RNN) and a Long Short Term Memory network (LSTM) on the data. An RNN captures sequential information[4] by computing an output, not only using the corresponding input phrase, but also the output of all previous phrases. An LSTM network works similarly, but at each iteration updates long and short term memory values based on previous outputs and the current input. Using these values combined with the current input, the LSTM computes the corresponding output. Both RNNs and LSTMs are used for text sentiment analysis in literature[21][16][20][13].

Results

We have used categorical crossentropy to evaluate all our models, as we have categorical data with multiple classes[6].

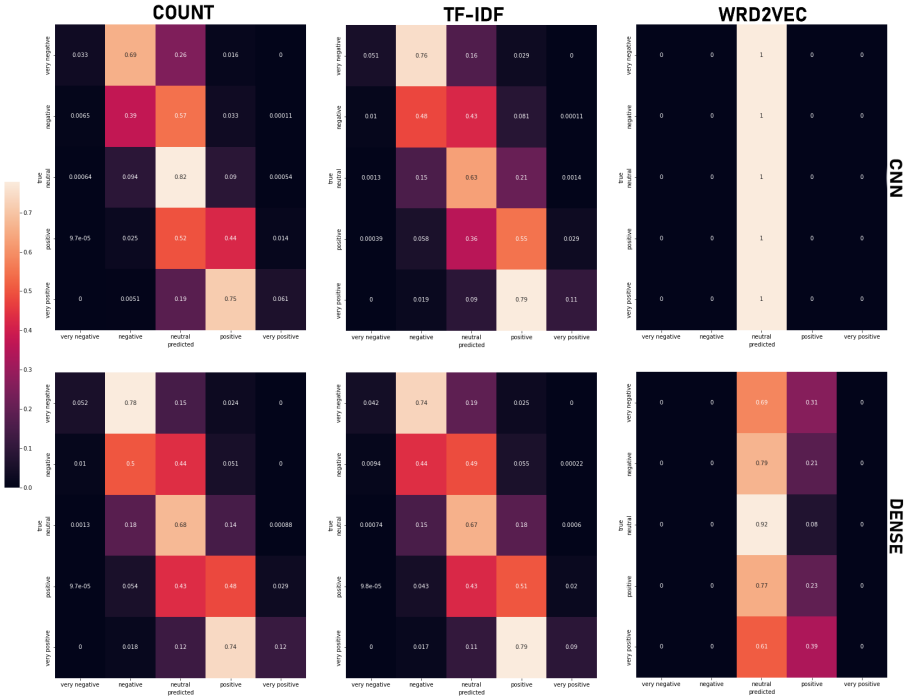


Figure 5: all of the confusion matrices

Method	Training accuracy	Training accuracy
Count CNN	60.39%	54.75%
Count Dense	64.69%	52.21%
TF-IDF CNN	59.397%	53.41%
TF-IDF Dense	63.85%	51.08%
Word2Vec CNN	46.12%	46.18%
Word2Vec Dense	47.56%	47.66%

Table 2: The training accuracy, and evaluation accuracy after hyperparameter tuning, for each preproccessing and learning method.

The results of our trained models can be seen in figure 5 amd Table 2. The count array combined with the CNN was the most effective, with a confusion matrix the closest to diagonal and an evaluation

accuracy of 54.75%. Along with higher accuracy the CNN confusion matrix shows that it was able to categorize neutral especially well, at 82%. The count array with the dense neural network and TF-IDF with the CNN also performed admirably. The TF-IDF encoding was clearly also an effective way to encode the phrases, as it performed similarly to the count vector.

The Word2Vec model consistently had the lowest accuracy of all encoding methods. Both of the confusion matrices were vertical with the CNN classifying everything as neutral, and accuracy scores were significantly the lowest.

Throughout, every method combination finds negative sentiment harder to classifying then postive - and 'extreme' sentiment far harder than that more neutral.

To successfully evaluate a model's performance we must also consider the model's overtraining and not just the testing accuracy. This is important to consider when choosing a model as a model that has overfitted on training data will struggle to classify new data correctly. A measure of overtraining can be visualised by the amount of diversion in the loss curves in 9 but is also apparent in the difference in the testing and training accuracy values. The least overtrained models were those which were performed on the Word2Vec encoding. This is not surprising as we see this model failed to learn sentiment. The most overtrained models were the dense models applied on the count vector-encoding and TF-IDF. This is expected considering the nature of a densely connected network.

6 Discussion and Conclusion

Word2vec's poor performance has a few potential explanations. It has been shown to perform well in literature undertaking similar tasks, so what is going wrong? The corpora given, even the original snippets corpus, is majority made up short, standalone texts, each with a similar format. Thus, the model may not have been given enough, **varied** contextual data to be able to separate words by sentiment effectively. Future methods might entail training the Word2vec model on the dictionary data, instead of the original snippets - as its far greater population of words occurring 10 to 100 times might be improve the quality of the semantic embedding. In this case you would have markedly less contextual variation however. Finally, it is potentially during the very crude summing of the word vectors to create each phrase vector that a lot, if not all, semantic information gets lost. Potentially this operation is acting as a We suggest future research into more complex and effective ways to semantically embed sentences.

Best method: CNN on TFIDF encoding As the accuracies of CNNs on the TF-IDF encoding and count vector encoding are very similar, their loss curves have been plotted on the same graph (7) to allow for comparison in overtraining to determine the best-performing method. Overall the count vector encoding has the highest testing accuracy (1.34% higher) and least overtraining, as shown by the reduction in divergence of he loss curves when compared to the TF-IDF. However, both performances are comparable so in practical terms it may be more suitable to use the TF-IDF CNN as it's smaller encoded size is much quicker to run and yeilds a comparible result to the TF-IDF

The most confident classifications lie inside the neutral and positive sentiment values and correspond to the trends in almost all confusion matrices. The lowest confidence results were found in the neutral and negative classifications. These models significantly struggled to map almost anything correctly into the very positive or very negative regions. The fact that both, the highest confidence and lowest confidence classifications, exist in the same class is not unusual; in fact, this indicates that the classes within the dataset are not well-defined, as can clearly be seen in figure 4. This leads to other classes being mislabeled as neutral.

Some network structures are prone to overfitting, specifically dense networks. This can be optimized through a process where random of network nodes are cut, and regularisers are put in place to reduce the complexity and depth. This simplification prevents the overfitting of the networks allowing for more accurate results. Below is this optimization applied to a dense network on a count vector encoding. The effectiveness of this practice is seen in figure 8 where we see a significant reduction in overtraining through the reduced amount of diversion in the loss curves.

Besides editing the layers of the neural network, callbacks are an effective but to improve Keras models. These document the parameter values every epoch and return values which lead to the best fit of a model (ie find which epoch to stop at or which learning rate works best) **??**.

References

[1] Martín Abadi, Ashish Agarwal, Paul Barham, Eugene Brevdo, Zhifeng Chen, Craig Citro, Greg S. Corrado, Andy Davis, Jeffrey Dean, Matthieu Devin, Sanjay Ghemawat, Ian Goodfellow, Andrew Harp, Geoffrey Irving, Michael Isard, Yangqing Jia, Rafal Jozefowicz, Lukasz Kaiser, Manjunath Kudlur, Josh Levenberg, Dandelion Mané, Rajat Monga, Sherry Moore, Derek Murray, Chris Olah, Mike Schuster, Jonathon Shlens, Benoit Steiner, Ilya Sutskever, Kunal Talwar, Paul Tucker, Vincent Vanhoucke, Vijay Vasudevan, Fernanda Viégas, Oriol Vinyals, Pete Warden, Martin Wattenberg, Martin Wicke, Yuan Yu, and Xiaoqiang Zheng. TensorFlow: Large-scale machine learning on heterogeneous systems, 2015. Software available from tensorflow.org.

[2] Marco Baroni, Georgiana Dinu, and Germán Kruszewski. Don't count, predict! a systematic comparison of context-counting vs. context-predicting semantic vectors. In *Proceedings of the 52nd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 238–247, Baltimore, Maryland, June 2014. Association for Computational Linguistics.

[3] Michael Cai. Analysis of tweets using deep neural architectures. 2019.

[4] Kartik Chaudhary. Sentiment classification with deep learning: Rnn, lstm, and cnn, Oct 2020.

[5] François Chollet et al. Keras. `https://keras.io`, 2015.

[6] Raúl Gómez. Understanding categorical cross-entropy loss, binary cross-entropy loss, softmax loss, logistic loss, focal loss and all those confusing names.

[7] Pierre Lison and Andrei Kutuzov. Redefining context windows for word embedding models: An experimental study. *CoRR*, abs/1704.05781, 2017.

[8] Leland McInnes, John Healy, and James Melville. Umap: Uniform manifold approximation and projection for dimension reduction, 2018.

[9] Tomás Mikolov, Quoc V. Le, and Ilya Sutskever. Exploiting similarities among languages for machine translation. *CoRR*, abs/1309.4168, 2013.

[10] 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.

[11] Ruggero Petrolito and Felice Dell'Orletta. *Word Embeddings in Sentiment Analysis*, pages 330–334. 01 2018.

[12] Radim Řehůřek and Petr Sojka. Software Framework for Topic Modelling with Large Corpora. In *Proceedings of the LREC 2010 Workshop on New Challenges for NLP Frameworks*, pages 45–50, Valletta, Malta, May 2010. ELRA. `http://is.muni.cz/publication/884893/en`.

[13] Sebastian Ruder, Parsa Ghaffari, and John G. Breslin. A hierarchical model of reviews for aspect-based sentiment analysis. *CoRR*, abs/1609.02745, 2016.

[14] Emil Hvitfeldt Silge and Julia. Supervised machine learning for text analysis in r, May 2022.

[15] Richard Socher, Alex Perelygin, Jean Wu, Jason Chuang, Christopher D. Manning, Andrew Ng, and Christopher Potts. Recursive deep models for semantic compositionality over a sentiment treebank. In *Proceedings of the 2013 Conference on Empirical Methods in Natural Language Processing*, pages 1631–1642, Seattle, Washington, USA, October 2013. Association for Computational Linguistics.

[16] Yixuan Tong, Yangsen Zhang, and Yuru Jiang. Study of sentiment classification for chinese microblog based on recurrent neural network. *Chinese Journal of Electronics*, 25:601–607, 07 2016.

[17] Xingyou Wang, Weijie Jiang, and Zhiyong Luo. Combination of convolutional and recurrent neural network for sentiment analysis of short texts. In *Proceedings of COLING 2016, the 26th International Conference on Computational Linguistics: Technical Papers*, pages 2428–2437, Osaka, Japan, December 2016. The COLING 2016 Organizing Committee.

[18] Asiri Wijesinghe. Sentiment analysis on movie reviews, 10 2015.

[19] Wei Xue and Tao Li. Aspect based sentiment analysis with gated convolutional networks. In *Proceedings of the 56th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 2514–2523, Melbourne, Australia, July 2018. Association for Computational Linguistics.

[20] Xindong You, Xueqiang Lv, Shangqian Zhang, Dawei Sun, and Shang Gao. Sentiment analysis of film reviews based on deep learning model collaborated with content credibility filtering. In Honghao Gao, Xinheng Wang, Muddesar Iqbal, Yuyu Yin, Jianwei Yin, and Ning Gu, editors, *Collaborative Computing: Networking, Applications and Worksharing*, pages 305–319, Cham, 2021. Springer International Publishing.

[21] Shangqian Zhang, Xueqiang Lv, Yunzhong Tang, and Zhian Dong. Movie short-text reviews sentiment analysis based on multi-feature fusion. In *Proceedings of the 2018 International Conference on Algorithms, Computing and Artificial Intelligence*, ACAI 2018, New York, NY, USA, 2018. Association for Computing Machinery.

7 appendix

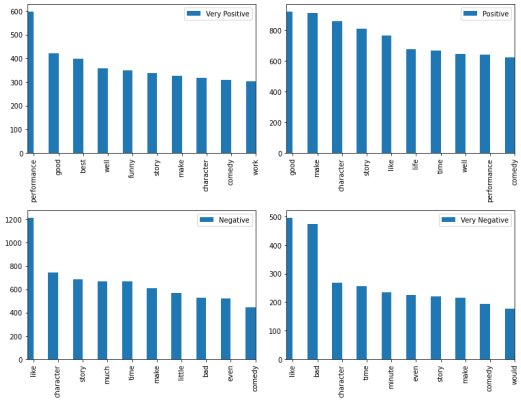


Figure 6: barcharts for phrases sentiment

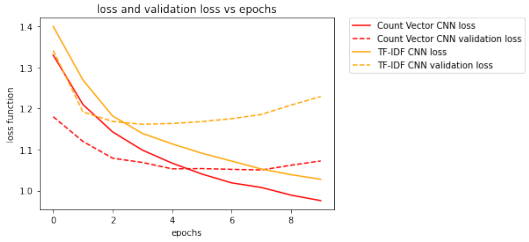


Figure 7: TF-IDF CNN vs Count Vector CNN

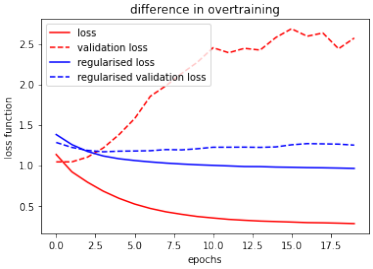


Figure 8: Difference in loss over epoch when overtraining is reduced by utilising regularisers, dropouts and limiting the depth of the neural network. This is evaluated by applying a Dense Neural Network to a count vector encoding

The phrase-trained model, specifically the CNN on the count vector encoding, is scaled so it can classify sentence sentiments. To achieve this sentence data was preprocessed identically to the phrase data. The sentence was tokenized and fit to the original count vector matrix as it must have the same features as the encoded phrase data to be used by the previously trained CNN. Predictions were made by the model using this new matrix as input data. There are no sentiment labels to compare the CNN-predicted classification to, thus the sentence sentiment can only be quantified by the confidence of classification (probability) and by manually assessing the classification.

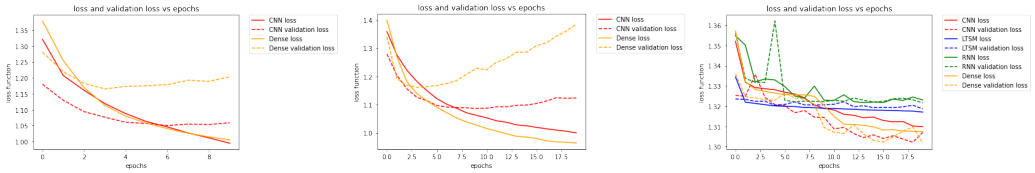
Highest confidence sentence classification

confidence (%)	sentiment	review
99.73537	Positive	Paid In Full is so stale, in fact, that its most... vibrant scene is one that uses clips from Bri...
99.70266	neutral	Before it collapses into exactly the kind of buddy cop... comedy it set out to lampoon , anyw
98.31885	neutral	Full of witless jokes, dealing in broad stereotypes and... outrageously unbelievable scenarios
98.04558	neutral	Rarely has a film's title served such dire warning.
97.99145	neutral	an 83 minute document of a project which started in a muddle , seesawed back...

Lowest confidence sentence classification

confidence (%)	sentiment	review
27.0120	neutral	Baran is shockingly devoid of your typical Majid Majidi shoe-loving, crippled children.
27.06285	neutral	Instead of building to a laugh riot we are left with a handful of disparate funny moments of n
27.26628	negative	Don't let your festive spirit go this far.
27.37083	neutral	A true-blue delight.
27.59124	negative	Lucy's a dull girl, that 's all.

Loss curves: assessing overtraining



(a) Count Array (b) TF-IDF (c) Word2vec

Figure 9: Loss curves of the model performance on each encoding.