Declaration on Plagiarism

Names	James Barry
Student Numbers	16212754
Programme	MCM
Module Code	CA683
Assignment Title	Practicum Submission
Submission date	22/08/2017
Module coordinator	Dr Donal Fitzpatrick

I declare that this material, which I now submit for assessment, is entirely my own work and has not been taken from the work of others, save and to the extent that such work has been cited and acknowledged within the text of my work. I understand that plagiarism, collusion, and copying are grave and serious offences in the university and accept the penalties that would be imposed should I engage in plagiarism, collusion or copying. I have read and understood the Assignment Regulations set out in the module documentation. I have identified and included the source of all facts, ideas, opinions, and viewpoints of others in the assignment references. Direct quotations from books, journal articles, internet sources, module text, or any other source whatsoever are acknowledged and the source cited are identified in the assignment references. This assignment, or any part of it, has not been previously submitted by me or any other person for assessment on this or any other course of study.

I have read and understood the referencing guidelines found recommended in the assignment guidelines.

Name: James Barry Date: 22/08/2017

Sentiment Analysis of Online Reviews Using LSTM and Bag-of-Words Approaches

James Barry
Dublin City University
School of Computing
james.barry26@mail.dcu.ie

***Abstract*—This paper implements a binary sentiment classification task on data sets of online reviews. The datasets include the Amazon Fine Food Reviews dataset and the Yelp Challenge Dataset. The paper approaches sentiment classification via two approaches: Firstly, a non-neural bag-of-words approach which uses Multinomial Naive Bayes and Support Vector Machine classifiers built on top. Secondly a Long Short-Term Memory Recurrent Neural Network is used. For the LSTM approach, we leverage the use of the pre-trained word embeddings Word2vec and GloVe as well as initialising our own weights from scratch. By doing so, we will be able to study the role of features which give us a greater semantic understanding of the text in the form of pre-trained weights. We also introduce a test where we use Word2vec embeddings which were initialised on our own corpora. The experiment is designed to test the role of word order in sentiment classification firstly by comparing bag-of-words approaches where word order is absent with an LSTM approach where sequential data is used as inputs. Our results show that the neural network approaches using pre-trained weights perform the best, though only by a slight amount. The tests are also carried out on a balanced dataset as well as a randomly distributed dataset to evaluate their effect on model performance.**

***Index Terms*—Sentiment analysis, Word2vec, GloVe, bag-of-words, Support Vector Machine, Naive Bayes, LSTM-RNN.**

I. Introduction

Sentiment Analysis is a fundamental task in Natural Language Processing. Its uses are many; from understanding behavioural trends on social media [1], gathering insight from user-generated product reviews [2] or even for financial purposes such as for developing trading strategies based on market sentiment. The goal of most sentiment classification tasks is to identify the overall sentiment polarity of the documents in question. For our case, we use online user-generated reviews from the Amazon Fine Food Reviews [3] and Yelp Challenge [4] datasets. In order to perform this sentiment classification task, we use a number of machine learning models to learn and predict binary reviews. This poses a supervised learning task.

The academic literature which approaches this task is rich and varied; pioneering approaches include Pang et al. [5] who use bag-of-words approaches with machine learning algorithms built on top to create a sentiment classifier. The popularity of such bag-of-words approaches is mainly due to their simplicity and efficiency, all the while having the ability to achieve very high success. Bag-of-words features are created by viewing the document as a collection of words, which are then used for classifying the document. In some cases, the words may be linked to a lexicon which contains pre-assigned sentiment values for the words.

Despite their overall high success rates, there exist some downsides to using bag-of-words or n-gram approaches. The main pitfall of such approaches is that that they ignore long-range word ordering such that modifiers and their objects may be separated by many unrelated words [6]. As word order is lost, sentences which use the same words will have the exact same representation despite the fact that they could have a very different semantic meaning in reality. Another key downside to using bag-of-word approaches is that they have very little understanding of the semantics of the words, or more specifically, the distances between words in an embedding space [7]. This is because words are treated as atomic units, resulting in sparse ”one-hot” vectors and therefore there is no notion of similarity between words [8].

The development of word embeddings has advanced many fields within Natural Language Processing. Such methods map words to a high-dimensional embedding space. The corresponding vectors in the embedding space can be used as inputs to enable non-linear learning algorithms such as neural networks. The idea of vector representations of words has been further advanced with the development of Word2vec [8] and GloVe [9] which are learned vector representations of words found by using unsupervised learning techniques on very large corpora. Such representations can encode fundamental relationships between words. As a result, simple algebraic operations can be performed on the word vectors to yield interesting syntactic relationships between words. For instance, the vectors [”King”] - [”Man”] + [”Woman”] results in the vector [”Queen”] [8].

Despite the fact that bag-of-words approaches used with machine learning classifiers achieve significant success, the development of word embeddings has allowed practitioners to use more advanced learning algorithms which can remember sequential data such as Recurrent Neural Networks (RNNs). Initially, standard RNNs were used for sequence-dependent tasks such as machine translation [10]. Despite being important tools for modeling sequential data, training RNNs by back-propagation through time can be difficult. More specifically, a problem arises when using RNNs for Natural Language Processing tasks because the gradients from the objective function vanish after a few time steps [11]. Indeed, multiple iterations of multiplying the weights of the network can lead to ’vanishing’ gradients if the weights are small. Conversely,

the gradients will 'explode' if the weight values are large. For such reasons, simple RNNs have rarely been used for NLP tasks such as text classification [6].

In such a scenario we can turn to another model in the RNN family such as the Long Short Term Memory (LSTM) RNN developed by Hochreiter and Schmidhuber [12]. LSTMs are better suited to this task due to the presence of input gates, forget gates, and output gates, which control the flow of information through the network. More recently, neural networks incorporating the LSTM structure have been used for sentiment classification. It is also shown that LSTMs achieve very good performance on a variety of document classification tasks once the hyperparameters are tuned correctly [6]. Such progress in machine learning techniques in recent years has enabled us to use more complex models on large datasets, which typically outperform more simple models. A very successful approach for many NLP tasks is to use distributed representations of words [13] with neural network models. Researchers using similar approaches such as Bengio et al. [14] and Mikolov [15] have shown us that using neural network based language models can yield better results than traditional n-gram models.

II. Related Literature

A. Recurrent Neural Networks

Recurrent neural networks became popular in the 1980s and 1990s with Jordan [16], Elman [17] and Rumelhart et al. [18]. They possess the ability to reuse parameters and model sequences over time. The models have a relatively simple architecture, for example, in Elman [17] there is an input layer for the current feature, a hidden layer which performs some nonlinear computation (which also sees its own state from the previous time step) and an output layer. Essentially, some form of memory is achieved by the state of the hidden layer.

While initially a promising model, it became known that there were some drawbacks with RNNs particularly in their ability to train, especially with stochastic gradient descent and backpropagation because of the vanishing/exploding gradient problem [11]; [19]. An important development which helped deal with these issues was the LSTM RNN architecture developed by Hochreiter and Schmidhuber [12] which added further architecture to the simple recurrent neural network which include neurons with input gates, forget gates and output gates which allowed the models to achieve greater memory by controlling the flow of information to hidden neurons. Their success was shown in NLP tasks such as hand writing recognition by Graves and Schmidhuber [20]. Today RNNs are used for many tasks such as speech recognition, machine translation, handwriting recognition and many other sequential problems.

B. Word Embeddings

Alongside the development of more sophisticated RNNs, a big contribution to tasks in the field of NLP was the development of word embeddings. Despite their recent fame, they are not necessarily a recent phenomena. Hinton et al. [13] and Elman [21] showed that simple neural networks can be used to obtain distributed representations of words. Collobert and Weston [22] achieved great success in using deep neural networks for the same purposes. Collobert et al. [23] induced useful word embeddings by applying a convolutional network architecture to a language modeling task trained using a large amount of unlabeled text data. These embeddings improve the generalisation performance on many of their NLP tasks, in particular, part-of-speech tagging and named entity recognition. The general idea of how their model works is that a word and its context is a positive training example, whereas when a random word is placed in that same context, this gives results in a negative training example. For example, a positive training example would be: "cat chills on a mat" and a negative example: "cat chills Jeju a mat". A neural network model gives a greater score for the real word in the sample than a random word placed in the same context. Other approaches for creating word embeddings using neural networks include Bengio et al. [24] and Turian et al. [25] who use deep learning models to represent words as a dense vector. Such word embeddings enable an understanding of the degree of similarity between words and also get rid of the problem where we have sparse "one-hot" vectors.

More recently, for learning word embeddings, Mikolov et al. [8] and Pennington et al. [9] developed the very popular word embeddings, Word2vec and GloVe respectively. Mikolov et al. [8] develop simple architectures such as Continuous-Bag-of-Words (CBOW) and Skip-Gram to find distributed representations of words. These models gain understanding of words in a corpora by analysing the co-occurrences of words over a large training sample. Word2vec is much faster and more accurate than previous neural network based approaches and speeds up training from previous methods significantly. Word embeddings have become highly popular as the features can be used easily in classifiers to boost the performance on many NLP tasks.

C. Sentiment Classification

Concerning sentiment classification, Pang et al. [5] use three standard algorithms: Naive Bayes classification, maximum entropy classification and a support vector machine to compare their performance in determining sentiment classification on document data (IMDb movie reviews). The authors incorporated a standard bag-of-features framework, which included good indicator words for positive and negative sentiment. Their results showed that machine learning techniques outperformed that of human-generated baselines. Despite the success of such approaches, one problem with bag-of-words methods is that they are unable to deal effectively with negation. For example, if they see words like "great" or "inspiring" in a review, it will classify a strong positive sentiment. However, if the actual sentence was "the cast was not great, nor was the movie inspiring" it has a completely different meaning which the models will fail to pick up. To overcome such difficulties as negation, researchers such as Turney [26], developed hand-written algorithms which can reverse the semantic orientation

of a word when it is preceded by a negative word. Turney develops a simple unsupervised learning algorithm for classifying reviews as recommended or not recommended. The algorithm uses a written review as the input and produces a classification (recommended or not recommended) as the output. Turney first uses a part-of-speech tagger to identify phrases in the text that contain adjectives or verbs and then estimates the semantic orientation of the phrases. While such algorithms are an important development to handle things like negation, it can be very time-consuming to develop heuristically designed rules which may not be able to deal with the multiple scenarios prevalent across human language.

D. Approaches using Neural Networks

Studies which use neural network architectures include Socher et al. [27] who use a semi-supervised approach using recursive autoencoders for predicting sentiment distributions. The model learns vector space representations for multi-word phrases and possesses the ability to understand the recursive nature of sentences. Socher et al. [28] highlight that understanding compositionality in tasks such as sentiment detection requires greater supervised training resources and more powerful models of composition. To fulfil such criteria, the authors introduce a Sentiment Treebank which includes sentiment labels for over 215,000 phrases. They also introduce a Recursive Neural Tensor Network, which when trained on the new Treebank, outperforms all previous methods on several metrics and forms a state of the art method for determining the positive/negative classifications of single sentences. Li et al. [29] compare recursive and recurrent neural networks on five NLP tasks including sentiment classification. Dai and Lei [6] perform a document classification task across a variety of datasets as well as a sentiment analysis task. They found that LSTMs pre-trained by recurrent language models or sequence autoencoders are better than LSTMs initialised from scratch. Le and Mikolov [7] introduce Paragraph Vector to learn document representation from semantics of words.

III. Data

Our datasets include the Amazon Fine Food Reviews dataset, the Yelp Challenge dataset and the IMDb dataset, all of which contain a series of reviews and labeled ratings. For this project, as it is a sentiment classification task, only the data containing raw text reviews and their equivalent rating were parsed. An example of a positive and a negative review is given below for the Amazon dataset:

> *Positive Review:* These bars are great! Great tasting and with quality wholesome ingredients. The company is great and has outstanding customer service and stand by their product 110% I highly recommend these bars in any flavor.
>
> *Negative Review:* These get worse with every bite. I even tried putting peanut butter on top to cover the taste. That didn't work. My five-year-old likes them. That is the only reason I didn't rate it lower.

A. Amazon Fine Food Reviews

The Amazon Fine Food Reviews Dataset contains 568,454 reviews. The dataset contains almost 46 million words and comprises 2.8 million sentences, with an average of 5 sentences per review. From Fig.1 we can see that there is a large number of 5-star reviews. We separate the data firstly into an evenly split distribution of 80,000 positive and negative reviews for one test and sample the distribution randomly for the other.

Fig. 1. Distribution of Ratings Amazon

B. Yelp Academic Reviews

We use the Yelp Academic Reviews dataset from the Yelp Challenge Dataset, which contains written reviews of listed businesses. The dataset contains over 4 million reviews and we parse data from two fields: "stars" and "text", where "stars" is the customer's rating from 1 to 5 and "text" is the customer's written review. The number of reviews in the dataset are 4,153,150. Out of a sample of 100,000 reviews: The number of sentences is: 829,165, while the number of words is 11.57 million. The average number of sentences in the reviews is 8.

C. Data Processing

The ratings in the Amazon and Yelp datasets were turned into binary positive and negative reviews where negative labels were assigned to ratings of 2 stars and below. Positive labels were assigned to ratings of 4 stars and above. Neutral, 3-star reviews were excluded so that our data would be highly polarised. In order to conduct our experiment, the datasets were split evenly between positive and negative reviews, with 82,000 reviews for each category and a second, randomly distributed dataset of 164,000 reviews was also created.

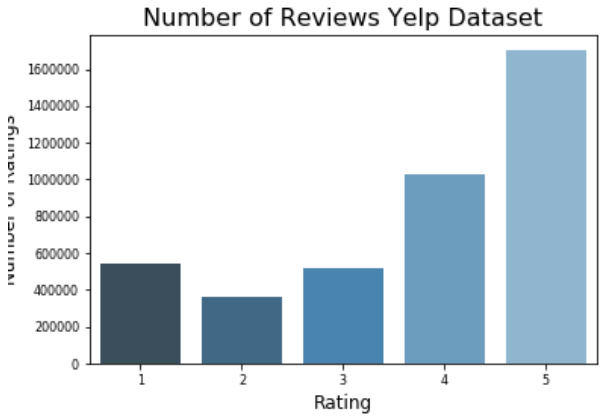

Fig. 2. Distribution of Ratings Yelp

D. Word Embeddings

For pre-trained word2vec word embeddings, we used the GoogleNews embeddings which were trained on 3 billion words from a Google News scrape. The data contains 3 million 300-dimensional word vectors for the English language. We also use GloVe embeddings which stands for Global Vectors for Word Representation. We use the dataset of a crawl on 42 billion tokens, with a vocabulary of 1.9 million words. Similarly, these vectors also have a dimension of 300.

IV. Approach

A. Problem Statement:

A long-term goal for NLP and Artificial Intelligence research involves developing machines which have the ability to understand language. If a human were asked to carry out a sentiment classification task, it would be presumed that they would carry out the task in a sequential, bottom-up fashion whereby they analyse the semantic relationship of the words and sentences overall rather than declaring the polarity of the review based on individual words. This motivates us to carry out this test: firstly to use traditional bag-of-words methods with an SVM classifier as a baseline, and secondly, to mimic a more human approach, using word order and features which have a more intelligent understanding of semantics such as pre-trained word embeddings.

As mentioned in the Introduction, traditional methods of sentiment analysis such as Bag of Words models, disregard the order of words, therefore these experiments will aim to compare various models which can incorporate sequential data, such as a recurrent neural network versus the bag-of-features model. Here the input sentence can be viewed as a sequence of tokens, rather than an unordered bag of tokens.

B. Baseline Approach I: Support Vector Machine

Support Vector Machines are a type of machine learning model introduced by Cortes and Vapnik [30]. An SVM is used in our experiment for text classification as they are shown to consistently achieve good performance on text categorisation tasks compared to other models [31]. A reason for this is that they possess the ability to generalise well in high dimensional feature spaces and eliminate the need for feature selection, making them a suitable choice of models for text categorisation tasks.

$$\overrightarrow{w} := \alpha_j c_j \overrightarrow{d_j}, \alpha_j \geq 0,$$

In our case, for binary sentiment classification, the idea behind the training method of the SVM is to find a maximum separating hyperplane $\overrightarrow{w}$, that separates the different document classes. The corresponding search is a constrained optimisation problem, letting $c_j \in$ -1:1 (where 1 refers to positive and -1 is negative) be the correct class of document d_j. From the above solution, the α_js are obtained by solving a dual optimisation problem. The documents $\overrightarrow{d_j}$ where α_j is greater than zero are called support vectors since only those document vectors are contributing to the hyperplane $\overrightarrow{w}$. We are able to classify test instances by determining which side of the of the hyperplane $\overrightarrow{w}$ they lie [5].

a) SVM Implementation: For the bag-of-words approach, the reviews were cleaned via a text-processing algorithm to remove any unwanted characters such as HTML links or numbers and retrieve only raw text. The next stage is to convert the words so that they have a numeric representation which can be used in machine learning models. The bag-of-words model builds a vocabulary from all of the words in the documents. It then models each document by counting the number of times each word appears. Considering the datasets contain a large volume of reviews, resulting in a large vocabulary, we limit the size of the feature vectors by choosing a maximum vocabulary size of 5000. Once the feature vectors were created, we performed GridSearch cross validation to find the optimal parameters for our support vector classifier. We found a linear kernel with a penalty of 1 to be the optimal model using cross-validation. We then fitted this tuned model to learn the relationship between the feature vectors and the labeled target variables. After doing so, we used our SVM to classify the text on the test dataset.

Initially the Support Vector Classifier with a linear kernel was successful for prediction. However, the model takes a very long time to perform GridSearch cross validation and to train on, especially when using bag-of-words features. This is a problem because GridSearch cross validation has to run a number of models to find the best penalty parameter. Also the test would have had to be run across multiple datasets, taking a period of time which was too much for this project. The difficulty arises as SVMs using kernels compute a distance function between each data point. When there is a large number of features, this creates a burden on memory and results in long training times. The problem is further exacerbated when the SVM tries to find the optimal separating hyperplane for data which is not scaled proportionally. As a result, there is a very large search space. One way of dealing with this issue is using a form of feature scaling where we convert our features into tf-idf features. Additionally, to speed

up training times, the number of K-Fold cross validation steps was reduced to three. Despite using tf-idf weights for feature scaling, it was still found to take a large period of time to run a linear SVC for this task.

It was found that another form of linear SVM, the SGD-Classifier significantly sped up cross validation and training times. The SGDClassifier is a form of linear SVM which uses stochastic gradient descent for learning. The model can also be a probabilistic classifier, if the loss function is set to 'log' which uses a logistic regression classifier but for the purpose of this project, only the linear SVM was used. GridSearch cross validation was used to find the optimal regularization and alpha or learning rate values. As the SGDClassifer linear SVM was able to train much faster than a traditional linear SVM with a C parameter, the model was able to handle bag-of-words features but we also evaluated tf-idf features for a comparison.

C. Baseline Approach II: Naive Bayes

As a second baseline classifier, we use the Multinomial Naive Bayes model. It is worth noting that Naive Bayes operates under the conditional independence assumption, that given the class, each of our words are conditionally independent of one another. This is not true in reality, as the order of words in a sentence plays an important role in the overall sentiment of a sentence. That said, Naive Bayes models using a bag-of-words features can still achieve impressive results, making it a valid baseline classifier.

For our context, we can state Bayes theorem as follows:

$$P(C^{(j)}|w_1^{(j)},...,w_n^{(j)}) = \frac{P(C^{(j)})P(w_1^{(j)},...,w_n^{(j)}|C^{(j)})}{P(w_1^{(j)},...,w_n^{(j)})} \quad (1)$$

To carry out this task we want to know $P(C^{(j)}|w_1^{(j)},...,w_n^{(j)})$, that is the probability of the class of the document $C^{(j)}$ given its words $w_1^{(j)},...,w_n^{(j)}$, where j is the document.

a) Naive Bayes Implementation: The Naive Bayes model was used as a baseline model in order to compare the results with the linear SVM. Parameters were mainly set to default given time constraint pressure for parameter tuning. As with the SVM approach, both tf-idf and regular features were evaluated.

D. Approach 2: Long Short Term Memory RNN

For the second approach, we use LSTM recurrent networks because they generally have a superior performance than traditional recurrent neural networks for learning relationships in sequential data. This is due to their unique architecture of having input gates, forget gates and output gates.

$$i_t = \sigma(W^{(i)}x_t) + U^{(i)}h_{t\_1} : \textit{Input gate}$$

$$f_t = \sigma(W^{(f)}x_t) + U^{(f)}h_{t\_1} : \textit{Forget gate}$$

$$\sigma_t = \sigma(W^{(o)}x_t) + U^{(o)}h_{t\_1} : \textit{Output gate}$$

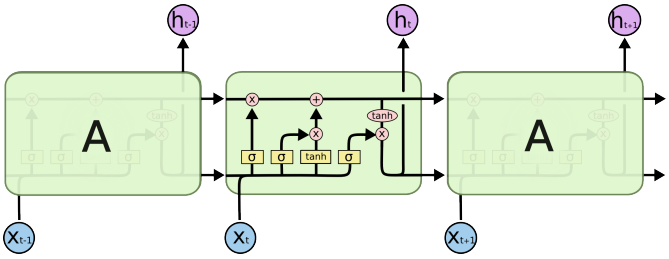

Fig. 3. The repeating module in an LSTM containing four interacting layers. [32].

$$\tilde{c}_t = tanh(W^{(c)}x_t) + U^{(c)}h_{t\_1} : \textit{New memory cell}$$

$$c_t = f_t * c_{t\_1} + i_t * \tilde{c}_t : \textit{Final memory cell}$$

$$h_t = o_t * tanh(c_t)$$

1) **New Memory Generation:** The input word x_t and the past hidden state h_{t_1} are used to generate a new memory $\tilde{c}_t$ which includes aspects of the new word xt.
2) **Input Gate:** The input gate's function is to ensure that new memory is generated only if the new word is important. The input gate achieves this by using the input word and the past hidden state to determine whether or not the input is worth preserving and thus controls the creation of new memory. It produces i_t as an indicator of this information.
3) **Forget Gate:** The forget gate is similar to the input gate but instead of determining the usefulness of the input word, it assesses whether the past memory cell is useful for the computation of the current memory cell. Here, the forget gate looks at the input word and the past hidden state and produces f_t.
4) **Final Memory Generation**: For this stage, the model takes the advice of the forget gate f_t and accordingly forgets the past memory c_{t_1}. It also takes the advice of the input gate i_t and gates the new memory $\tilde{c}_t$. The model sums these two results to produce the final memory c_t.
5) **Output/Exposure Gate:** This gate's purpose is to separate the final memory from the hidden state. Hidden states are present in every gate of an LSTM and consequently, this gate assesses what part of the memory c_t needs to be exposed/present in the hidden state h_t. The signal o_t is produced by i_t to indicate this and is used to gate the point-wise tanh of the memory [33].

E. LSTM Implementation

As with the baseline approach, we use a text processing algorithm to remove any unwanted characters and format our text data into sequences of sentences. We then convert all text samples in the dataset into sequences of word indices which involves converting each word to its integer ID. Essentially, we convert words from each text document into a sequence of vectors which can be used in our model. Each resulting sequence of vectors will be a separate data sample used an

input into the LSTM model. The LSTM will then process all vectors in that sequence and output a vector with a probability of the text belonging to a positive or negative category.

For this study, we permit the 200,000 most commonly occurring words in our vocabulary. We truncate the sequences to be a maximum length of 1000 words. Once the words are converted into their corresponding integers, we can prepare an embedding matrix which contains at index i, the embedding vector (e.g. Word2vec or Glove embedding) for the word at index i. The embedding matrix is then loaded into a Keras embedding layer and fed through the LSTM. The output layer of our LSTM model is a softmax function which is used to classify the review as positive or negative. Because our model has a deep architecture, Dropout is used as a form of regularization which essentially turns off certain nodes in the neural network architecture to prevent over-fitting.

For our study, this process is carried out with 3 variations: Firstly, we use Word2vec word embeddings and secondly, we use GloVe embeddings. Both methods allocate a dense numeric vector to every word in the dictionary. By doing so, the distance (e.g. the cosine distance) between the vectors will capture part of the semantic relationship between the words in our vocabulary. Lastly, we test how well we would have performed by not using pre-trained word embeddings and instead use our own indices in the embedding layer and allow the model to learn the weights. As a second experiment within the LSTM approach, we try using Word2vec weights which were initialised our own corpora and not pretrained on a larger corpora.

V. Results

The results of our various models are shown in this section. Here we compare our approaches from the baseline bag-of-words models as well as the results from the different LSTM and word embedding experiments. The study was carried out on two datasets, one where there is an even split between positive and negative reviews, and secondly, one which is sampled randomly and should have a proportionally higher amount of positive reviews as seen in our Data section.

In the even dataset, for the bag-of-words approaches, the SGDClassifier SVM using tf-idf features performed the best achieving an accuracy of 93.2%. The Naive Bayes classifier performed worse across the range of tests but still achieved somewhat satisfactory accuracy, rendering its use as a baseline.

Unfortunately, for the LSTM approach, model training and evaluation times were taking very long, particularly when accounting for validation. The models were allowed to run on the even dataset for a number of hours but unfortunately it was taking too long to apply the models to both dataset distributions and across two different data sets to finish in time for this project. Consequently, only the results of three models are shown: an LSTM with Word2vec embeddings, LSTM with self initialised weights and the LSTM using Word2vec embeddings trained on our own corpora (LSTM Word2vec*). The LSTM with Word2vec weights performs the best across the whole of the Amazon reviews category, indicating that the LSTM can handle sequential information, such as text in a review, very well. Meanwhile the weights help improve model performance as the model with pre-trained weights outperforms the LSTM which uses Word2vec embeddings which were learned on its own corpora as well as the model with self initialised weights.

Model Results Even Dataset		
Model Type	Amazon	Yelp
SVM BoW	88.4	90.8
SVM BoW tf-idf	89.3	**93.2**
NB	85.7	86.9
NB tf-idf	86.4	88.5
LSTM Word2Vec	**93.7**	-
LSTM GloVe	-	-
LSTM Self	92.4	-
LSTM Word2vec*	90.5	-

Moving on to the randomly distributed dataset: It is worth noting that in order to speed up model run times for the LSTM approaches in the Yelp dataset, smaller dataset sizes were used. The models still generalise well to the data and have good overall performance, however, reducing the dataset sizes for the sake of completion may harm the validity of the results. While time and compute power are large constraints here, it is hoped that by the time of the presentation, these tests will be carried out on the same size datasets, giving more conclusive results. As with the even dataset, the LSTM with pre-trained word embeddings performed best. The model using GloVe embeddings narrowly outperformed the model using Word2vec embeddings, which may be attributed to the larger token and vocabulary size in the GloVe crawl, which contained 42 billion tokens. We can also see that the model using self-initialised weights slightly under performs its counterparts which use pre-trained weights, though only by a small amount. This slight difference can highlight the role of the greater syntactic understanding of the text data when using pre-trained weights as compared with self-learned weights. These results, combined with the lower results from the Word2vec model trained on our own, smaller corpora, can give credibility to the use of features which have a degree of understanding about the words in the documents.

Likewise, with the evenly distributed dataset, SVM models perform the best among the bag-of-words models, where we have a very strongly performing SGDClassifier which has standard n-gram features which are not converted to tf-idf features.

Model Results Random Dataset		
Model Type	Amazon	Yelp
SVM BoW	91.9	93.8
SVM BoW tfidf	**92.9**	92.9
NB	90.3	89.4
NB tf-idf	84.6	89.8
LSTM Word2Vec	-	94.2
LSTM GloVe	-	**94.3**
LSTM Self	-	92.9
LSTM Word2vec*	-	

VI. Conclusion

In this study we have compared bag-of-words and neural network based approaches for sentiment classification. Firstly, we use features which have an unstructured order, e.g. bag-of-words models, and secondly, we use an LSTM model which can handle sequential data as well as leverage the use of pre-trained word embeddings. From our preliminary analysis, the LSTM models seem to slightly outperform the bag-of-words models, although only by a small margin. Bag-of-words models still perform very well, particularly with respect to their shorter training period. The best performing LSTM models incorporate pre-trained word embeddings which enable the model to have a greater syntactic understanding of the data. We also compared our results across two different data distributions, a balanced dataset and a randomly sampled one. There is very little difference between the results from the balanced dataset versus that of a randomly sampled one (which tends to be overly-represented by positive reviews). Under these circumstances, the model may overfit slightly on positive reviews but despite this, may perform well due to it being more likely to classify the majority.

Acknowledgment

I'd like to thank my supervisor Dr Jennifer Foster for her time and guidance throughout this project.

References

[1] A. Bakliwal, J. Foster, J. van der Puil, R. O'Brien, L. Tounsi, and M. Hughes, "Sentiment analysis of political tweets: Towards an accurate classifier." Association for Computational Linguistics, 2013.

[2] B. Pang, L. Lee *et al.*, "Opinion mining and sentiment analysis," *Foundations and Trends® in Information Retrieval*, vol. 2, no. 1–2, pp. 1–135, 2008.

[3] J. J. McAuley and J. Leskovec, "From amateurs to connoisseurs: modeling the evolution of user expertise through online reviews," in *Proceedings of the 22nd international conference on World Wide Web*. ACM, 2013, pp. 897–908.

[4] Yelp, "Yelp Dataset Challenge," $https://www.yelp.ie/dataset_challenge$, 2017, [Online; accessed 23-June-2017].

[5] B. Pang, L. Lee, and S. Vaithyanathan, "Thumbs up?: sentiment classification using machine learning techniques," in *Proceedings of the ACL-02 conference on Empirical methods in natural language processing-Volume 10*. Association for Computational Linguistics, 2002, pp. 79–86.

[6] A. M. Dai and Q. V. Le, "Semi-supervised sequence learning," in *Advances in Neural Information Processing Systems*, 2015, pp. 3079–3087.

[7] Q. Le and T. Mikolov, "Distributed representations of sentences and documents," in *Proceedings of the 31st International Conference on Machine Learning (ICML-14)*, 2014, pp. 1188–1196.

[8] T. Mikolov, K. Chen, G. Corrado, and J. Dean, "Efficient estimation of word representations in vector space," *arXiv preprint arXiv:1301.3781*, 2013.

[9] J. Pennington, R. Socher, and C. D. Manning, "Glove: Global vectors for word representation." in *EMNLP*, vol. 14, 2014, pp. 1532–1543.

[10] I. Sutskever, O. Vinyals, and Q. V. Le, "Sequence to sequence learning with neural networks," in *Advances in neural information processing systems*, 2014, pp. 3104–3112.

[11] S. Hochreiter, Y. Bengio, P. Frasconi, J. Schmidhuber *et al.*, "Gradient flow in recurrent nets: the difficulty of learning long-term dependencies," 2001.

[12] S. Hochreiter and J. Schmidhuber, "Long short-term memory," *Neural computation*, vol. 9, no. 8, pp. 1735–1780, 1997.

[13] G. E. Hinton, "Learning distributed representations of concepts," in *Proceedings of the eighth annual conference of the cognitive science society*, vol. 1. Amherst, MA, 1986, p. 12.

[14] Y. Bengio, "Deep learning of representations: Looking forward," in *International Conference on Statistical Language and Speech Processing*. Springer, 2013, pp. 1–37.

[15] T. Mikolov, A. Deoras, D. Povey, L. Burget, and J. Černockỳ, "Strategies for training large scale neural network language models," in *Automatic Speech Recognition and Understanding (ASRU), 2011 IEEE Workshop on*. IEEE, 2011, pp. 196–201.

[16] M. I. Jordan, "Attractor dynamics and parallellism in a connectionist sequential machine," 1986.

[17] J. L. Elman, "Finding structure in time," *Cognitive science*, vol. 14, no. 2, pp. 179–211, 1990.

[18] D. E. Rumelhart, G. E. Hinton, J. L. McClelland *et al.*, "A general framework for parallel distributed processing," *Parallel distributed processing: Explorations in the microstructure of cognition*, vol. 1, pp. 45–76, 1986.

[19] Y. Bengio, P. Simard, and P. Frasconi, "Learning long-term dependencies with gradient descent is difficult," *IEEE transactions on neural networks*, vol. 5, no. 2, pp. 157–166, 1994.

[20] A. Graves, M. Liwicki, S. Fernández, R. Bertolami, H. Bunke, and J. Schmidhuber, "A novel connectionist system for unconstrained handwriting recognition," *IEEE transactions on pattern analysis and machine intelligence*, vol. 31, no. 5, pp. 855–868, 2009.

[21] J. L. Elman, "Distributed representations, simple recurrent networks, and grammatical structure," *Machine learning*, vol. 7, no. 2-3, pp. 195–225, 1991.

[22] R. Collobert and J. Weston, "A unified architecture for natural language processing: Deep neural networks with multitask learning," in *Proceedings of the 25th international conference on Machine learning*. ACM, 2008, pp. 160–167.

[23] R. Collobert, J. Weston, L. Bottou, M. Karlen, K. Kavukcuoglu, and P. Kuksa, "Natural language processing (almost) from scratch," *Journal of Machine Learning Research*, vol. 12, no. Aug, pp. 2493–2537, 2011.

[24] Y. Bengio, R. Ducharme, P. Vincent, and C. Jauvin, "A neural probabilistic language model," *Journal of machine learning research*, vol. 3, no. Feb, pp. 1137–1155, 2003.

[25] J. Turian, L. Ratinov, and Y. Bengio, "Word representations: a simple and general method for semi-supervised learning," in *Proceedings of the 48th annual meeting of the association for computational linguistics*. Association for Computational Linguistics, 2010, pp. 384–394.

[26] P. D. Turney, "Thumbs up or thumbs down?: semantic orientation applied to unsupervised classification of reviews," in *Proceedings of the 40th annual meeting on association for computational linguistics*. Association for Computational Linguistics, 2002, pp. 417–424.

[27] R. Socher, J. Pennington, E. H. Huang, A. Y. Ng, and C. D. Manning, "Semi-supervised recursive autoencoders for predicting sentiment distributions," in *Proceedings of the conference on empirical methods in natural language processing*. Association for Computational Linguistics, 2011, pp. 151–161.

[28] R. Socher, A. Perelygin, J. Y. Wu, J. Chuang, C. D. Manning, A. Y. Ng, C. Potts *et al.*, "Recursive deep models for semantic compositionality over a sentiment treebank," in *Proceedings of the conference on empirical methods in natural language processing (EMNLP)*, vol. 1631, 2013, p. 1642.

[29] J. Li, M.-T. Luong, D. Jurafsky, and E. Hovy, "When are tree structures necessary for deep learning of representations?" *arXiv preprint arXiv:1503.00185*, 2015.

[30] C. Cortes and V. Vapnik, "Support-vector networks," *Machine learning*, vol. 20, no. 3, pp. 273–297, 1995.

[31] T. Joachims, "Transductive inference for text classification using support vector machines," in *ICML*, vol. 99, 1999, pp. 200–209.

[32] "Understanding lstm networks," (Date last accessed 22-August-2017). [Online]. Available: $http://colah.github.io/posts/2015-08-Understanding-LSTMs/$

[33] "Lecture notes: Part v2," (Date last accessed 22-August-2017). [Online]. Available: $http://web.stanford.edu/class/cs224n/lecture_notes/cs224n-2017-notes5.pdf$