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

James Barry

School of Computing, Dublin City University, Ireland
james.barry26@mail.dcu.ie

Abstract. This paper implements a binary sentiment classification task on datasets of online reviews. The datasets include the Amazon Fine Food Reviews Dataset and the Yelp Challenge Dataset. The paper performs sentiment classification via two approaches: firstly, a non-neural bag-of-words approach using Multinomial Naive Bayes and Support Vector Machine classifiers. Secondly, a Long Short-Term Memory (LSTM) Recurrent Neural Network is used. The experiment is designed to test the role of word order in sentiment classification by comparing bag-of-words approaches where word order is absent with an LSTM approach which can handle sequential data as inputs. For the LSTM approaches, we test the role of various features such as pre-trained Word2vec and Glove embeddings as well as Word2vec embeddings learned on domain specific corpora. We also test the effect of initialising our own weights from scratch. The tests are carried out on balanced datasets as well as on datasets which follow their original distribution. This measure enables us to evaluate the effect of ratings distribution on model performance. Our results show that the LSTM approaches using GloVe embeddings and self-learned Word2vec embeddings perform best, whilst the distribution of ratings in the data has a meaningful impact on model performance.

1 Introduction

Sentiment Analysis is a fundamental task in Natural Language Processing (NLP). Its uses are many: from analysing political sentiment on social media [1], gathering insight from user-generated product reviews [2] or even for financial purposes, such as developing trading strategies based on market sentiment [3]. The goal of most sentiment classification tasks is to identify the overall sentiment polarity of the documents in question, i.e. is the sentiment of the document positive or negative? For our case, we use online user-generated reviews from the Amazon Fine Food Reviews [4] and Yelp Challenge [5] datasets. In order to perform this sentiment classification task, we use a mixture of baseline machine learning models and deep learning models to learn and predict the sentiment of binary reviews. This poses a supervised learning task.

Pioneering approaches for sentiment classification include Pang et al. [6] who use bag-of-words features 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, whilst having the ability to achieve very high accuracy. Bag-of-words features are created by viewing the document as an unordered collection of words, which are then used to classify the document. 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 [7]. As word order is lost, sentences with different meanings which use the same words will have similar representations.

Another key downside to using bag-of-words approaches is that they are unable to deal effectively with negation. For example, if the model sees words like "great" or "inspiring" in a review, it will likely prompt a positive classification. However, if the actual sentence was, "The cast was not great, nor was the movie inspiring.", it has a completely different meaning which the model will fail to pick up. Additionally, bag-of-words features have very little understanding of the semantics of the words, which can be measured as the distances between words in an embedding space [8]. 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 [9].

The inclusion of word embeddings in NLP tasks enables us to overcome such problems. Recently, Mikolov et al. [9] and Pennington et al. [10] developed the very popular word embedding models, Word2vec and GloVe respectively which gain an understanding of words in a corpora by analysing the co-occurrences of words over a large training sample. Such representations can encode fundamental relationships between words, such that simple algebraic operations can yield meaningful semantic information between words.

Furthermore, the addition of word embeddings to the field of NLP has enabled practitioners to use more advanced learning algorithms which can handle sequential data as inputs such as Recurrent Neural Networks (RNNs). An important development in the field of RNNs was the introduction of the Long Short-Term Memory (henceforth LSTM) RNN by Hochreiter and Schmidhuber [11]. Their success has been shown in NLP tasks such as handwriting recognition by Graves and Schmidhuber [12]. Today, LSTMs are used for many tasks such as speech recognition, machine translation, handwriting recognition and many other sequential problems.

2 Related Literature

Concerning sentiment classification, Pang et al. [6] incorporated a standard bag-of-features framework to predict the sentiment class of movie reviews. Their results showed that machine learning techniques using bag-of-words features outperformed simple decision-making models which used hand-picked feature words for sentiment classification. To overcome difficulties with bag-of-words methods such as negation, Turney [13], developed hand-written algorithms which can reverse the semantic orientation of a word when it is preceded by a negative word. 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.

Studies which use neural network architectures include Socher et al. [14] who use a semi-supervised approach using recursive autoencoders for predicting sentiment distributions. Socher et al. [15] introduce a Sentiment Treebank and 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. [16] compare recursive and recurrent neural networks on five NLP tasks including sentiment classification. Dai and Lei [7] 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 [8] introduce Paragraph Vector to learn document representation from semantics of words.

3 Data

Our datasets include the Amazon Fine Food Reviews dataset[1] and the Yelp Challenge dataset [2], both of which contain a series of reviews and labeled ratings. For this project, as it is a sentiment classification task, only the data containing the raw text reviews and their equivalent rating were parsed. An example of a positive and a negative review from the Amazon dataset is given below:

> *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.*

3.1 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.

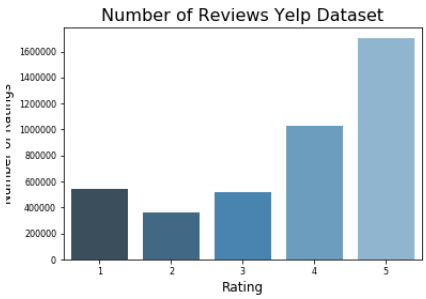
3.2 Yelp Academic Reviews

We use the Yelp Academic Reviews dataset from the Yelp Dataset Challenge, which contains written reviews of listed businesses. We parse data from two

[1] `https://snap.stanford.edu/data/web-FineFoods.html`
[2] `https://www.yelp.com/dataset/challenge`

fields: “stars” and “text”, where “stars” is the customer’s rating from 1 to 5 and “text” is the customer’s written review. There are 4,153,150 reviews in the dataset. 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.

(a) Distribution of Ratings Amazon



(b) Distribution of Ratings Yelp

3.3 Word Embeddings

For the pre-trained Word2vec word embeddings, we use the GoogleNews embeddings [3] 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 as a comparison. We use the GloVe embeddings which were trained on a crawl of 42 billion tokens, with a vocabulary of 1.9 million words. [4] Similarly, these vectors also have a dimension of 300.

3.4 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. From Figures (a) and (b) we can see that there is a large number of 5-star reviews in both datasets. In order to test the effect of using a balanced dataset, with an even number of positive and negative reviews and a dataset which follows the original distribution, we carry out two different sampling techniques: First, we separate the datasets into an evenly split distribution of 82,000 positive and

[3] The GoogleNews embeddings are available at: `https://github.com/mmihaltz/word2vec-GoogleNews-vectors`.

[4] The GloVe 42B embeddings are available at: `https://nlp.stanford.edu/projects/glove/`.

82,000 negative reviews (as there are only around 82,000 negative reviews in the Amazon dataset). For the second test, to analyse the effect of the original distribution, we randomly sample 164,000 reviews from the datasets, which should have a proportionally higher number of positive reviews reflecting the original distribution. We use 164,000 reviews in both datasets and distributions to ensure differing results are not attributed to varying dataset sizes. For our experiments, we partition each of our datasets by an 80:20 training/test split. As the split is made after the various dataset sampling measures, the distribution of ratings in the training/test sets should be representative of the specified sampling approach. By doing so, we avoid the situation whereby models trained on balanced data are used to predict on the original distribution and vice versa.

4 Approach

4.1 Baseline Approach I: Support Vector Machine

Support Vector Machines are a type of machine learning model introduced by Cortes and Vapnik [17]. 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 [18]. 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. Many classifier learning algorithms such as SVMs using a linear kernel assume that the training data is independently and identically distributed as part of their derivation [19]. This assumption is often violated in applications such as text classification as the order of words in a sentence will have a significant impact on the overall sentiment of the sentence or phrase. Nevertheless, such classifiers can achieve high accuracy, representing good baseline metrics for our study.

$$\overrightarrow{w} := \sum_j \alpha_j \mathsf{c}_j \overrightarrow{d_j}, \alpha_j \geq 0,$$

Looking at the above solution, the idea behind the training method of the SVM for this task is to find a maximum separating hyperplane $\overrightarrow{w}$, that separates the different document classes. The corresponding search is a constrained optimisation problem, letting $\mathsf{c}_j \in$ {-1:1} (where 1 refers to positive and -1 is negative) be the correct class of document d_j. 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 [6].

SVM Implementation For the bag-of-words approach, the reviews were cleaned via a text-processing algorithm to remove any unwanted characters, HTML links or numbers and retrieve only raw text. The next stage involves converting the

words in the reviews from text to integers so that they have a numeric representation which can be used in machine learning models.[5]. 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 a word in the vocabulary appears in the document. Considering the datasets contain a large number of reviews, resulting in a large vocabulary, we limit the size of the feature vectors by choosing a maximum vocabulary size of the 5000 top-occurring words. Bag-of-words features were created for both training and test sets. We used an 80:20 train/test split for our experiment. GridSearch cross-validation with 3 folds was used to find the optimal Cost parameter for our Linear Support Vector Classifier (SVC) on the training sets. A form of feature scaling used in text classification tasks includes converting the words to tf-idf features, which stands for term frequency - inverse document frequency, which is a value that corresponds to how distinctive a word is in a corpus [20]. We evaluate both standard bag-of-words and tf-idf features.

4.2 Baseline Approach II: Multinomial Naive Bayes

As a second baseline classifier, we use the Multinomial Naive Bayes (MNB) 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 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:

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

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.

Multinomial Naive Bayes Implementation The MNB model was used as a baseline model in order to compare the results with the linear SVM. GridSearch cross-validation was used to find the optimal α value for each MNB model. As with the SVM approach, both tf-idf and regular features were evaluated.

4.3 Long Short-Term Memory RNN

For our neural network approach, we use LSTM RNNs because they generally have a superior performance than traditional RNNs for learning relationships in

[5] Both the Multinomial Naive Bayes and Support Vector Machine Classifiers were implemented in Python using the Scikit-learn library

sequential data. A problem arises when using traditional RNNs for NLP tasks because the gradients from the objective function can vanish or explode after a few iterations of multiplying the weights of the network [21]. For such reasons, simple RNNs have rarely been used for NLP tasks such as text classification [7]. In such a scenario we can turn to another model in the RNN family such as the LSTM model. 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. The LSTM architecture is outlined below:

$$i_t = \sigma(W^{(i)}x_t) + U^{(i)}h_{t-1} : \textit{Input gate}$$
$$f_t = \sigma(W^{(f)}x_t) + U^{(f)}h_{t-1} : \textit{Forget gate}$$
$$\sigma_t = \sigma(W^{(o)}x_t) + U^{(o)}h_{t-1} : \textit{Output gate}$$
$$\tilde{c}_t = tanh(W^{(c)}x_t) + U^{(c)}h_{t-1} : \textit{New memory cell}$$
$$c_t = f_t * c_{t-1} + i_t * \tilde{c}_t : \textit{Final memory cell}$$
$$h_t = o_t * tanh(c_t)$$

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 [22].

LSTM Implementation For our study, the LSTM implementation is carried out with four variations: Firstly, we use pre-trained Word2vec embeddings. Secondly, we use pre-trained GloVe embeddings. Thirdly, we use Word2vec embeddings which were learned on domain-specific corpora. For this experiment, we run the Word2vec model to generate word embeddings on each of our datasets. These

embeddings are used as inputs to the LSTM to learn and predict on that particular dataset. Lastly, we test how well we would have performed by not using pre-trained word embeddings and instead keep the original word indices in the embedding layer and allow the model to learn the weights itself. In contrast to this approach, the Word2vec and GloVe 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.

LSTM Design As with the baseline approach, we use a text processing algorithm to remove any unwanted characters. We then convert all text samples in the dataset into vectors of word indices which involves converting each word to its integer ID. 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 in the reviews are converted into their corresponding integers, for the word embedding approaches, 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.[6]

Param	**Value**	**Param**	**Value**
Input length	1000	Embedding size	300
LSTM size	200	Hidden layer size	128
Dropout	0.25	Recurrent Dropout	0.25
Activation	ReLU	Optimizer	Nadam
Batch size	64	Output	Sigmoid

Table 1: LSTM Hyperparameter values.

As with the baseline approaches, we partition the training and test data by an 80:20 split. We perform GridSearch cross-validation with 3 folds to find the optimal model hyperparameters on the training data. We tested for several parameters including LSTM and hidden layer sizes, batch size and the dropout value. After conducting GridSearch cross-validation, the following hyper-parameters were chosen: The optimal LSTM layer size was found to be 200, while the hidden layer size was found to be 128 units. A Dropout value of 25% was selected, which helps our models to prevent over-fitting by randomly turning off nodes in the network. The activation function we use on the inner layer is ReLU (Rectified Linear Unit), which is a function that maps negative values to 0 and positive values linearly which helps transmit errors during back-propagation. The optimizer we use is Nadam, which is a variation of the popular Adam optimizer

[6] Keras was used as the deep learning library to build the LSTM network. The GPU version of Tensorflow was used to speed up training times significantly.

which incorporates Nesterov momentum [23]. The output layer of our LSTM model is a Sigmoid function which is used to condense the output value of our network into a probability of classifying the review as positive or negative. The number of epochs during model training was set to 10 as validation accuracy was improving whilst signs of over-fitting were not setting in at this value.

5 Results

The results of our various models and dataset distributions are shown in this section. The metrics we use are accuracy and AUC (referring to area under ROC curve), which gives a measure of the relative share of true positive and false positive rates depending on a threshold. The inclusion of this metric will help shed light on the role of ratings distributions and how they affect the model's classification ability. For example, a model trained on a dataset containing a higher proportion of positive reviews may perform better on a superficial level due to it being more likely to predict the majority class prevalent in the data.

From Table 2, with respect to the balanced dataset, from the bag-of-words approaches, the SVM using TF-IDF features performed the best achieving an accuracy 88.95% on the Amazon dataset. Similarly, the SVM TF-IDF model performed best on the Yelp dataset with an accuracy of 92.91%. In both cases, the AUC score is very similar to the accuracy score. The MNB classifiers performed worse across the range of tests but still achieved satisfactory accuracy and AUC scores, rendering the use of MNB models as a baseline.

For the LSTM models, we can see that they generally perform better than the baseline bag-of-words methods with the exception of the LSTM using Word2vec embeddings learned on the Amazon dataset, which lags behind the other LSTM variations and even the SVM models and some MNB models on the Yelp dataset. The LSTM with GloVe embeddings performs the best across the Amazon dataset with an accuracy of 95.03% and and AUC score of .9889. Meanwhile, the LSTM using dataset specific Word2vec embeddings performs best with 95.75% accuracy and an AUC score of .9924 on the Yelp dataset.

The promising results from the LSTM models indicate that the LSTMs can handle sequential information well and the addition of word embeddings help improve model performance, where the models using embeddings narrowly outperform the models using self-initialised weights. An interesting result is the difference in performance between the LSTM models using domain-specific Word2vec embeddings on the Amazon and Yelp datasets, where these features resulted in a worst performing and best performing LSTM model respectively. A reason for this could be due to attributes of the corpora involved. The Amazon dataset had an average of five words per sentence, whilst the Yelp dataset had an average of eight words per sentence. The longer sentences in the Yelp dataset could give rise to a scenario where more semantic information can be captured by the Word2vec model, resulting in better embeddings.

With respect to the original datasets: As with the balanced datasets, the SVMs perform better than the MNB models, with SVMs using TF-IDF features

		Amazon		Yelp	
Distribution	**Model**	**Accuracy**	**AUC**	**Accuracy**	**AUC**
Balanced	SVM	87.15	.8713	90.97	.9097
	SVM TF-IDF	88.95	.8894	92.91	.9290
	MNB	85.67	.8565	87.07	.8705
	MNB TF-IDF	86.27	.8626	88.72	.8871
	LSTM Word2vec	94.77	.9869	94.22	.9875
	LSTM Self Initialised	93.75	.9778	94.41	.9836
	LSTM GloVe	**95.03**	**.9889**	95.31	.9913
	LSTM Word2vec-domain	87.98	.9496	**95.75**	**.9924**
Original	SVM	91.66	.7817	93.12	.8859
	SVM TF-IDF	92.81	.8146	94.17	.9047
	MNB	90.42	.8268	89.10	.8601
	MNB TF-IDF	88.51	.6350	89.70	.8132
	LSTM Word2vec	95.75	.9826	95.89	.9900
	LSTM Self Initialised	94.86	.9612	95.27	.9794
	LSTM GloVe	**95.84**	.9832	96.25	.9901
	LSTM Word2vec-domain	95.75	**.9845**	**96.51**	**.9925**

Table 2: Model results across datasets and distributions. **Self Initialised:** Self-initialised weights. **Word2vec-domain:** Word2vec embeddings learned from the individual dataset.

performing better on the Amazon and Yelp datasets. Whilst the baseline models look good at an initial glance in terms of accuracy, the AUC scores on the datasets following the original distribution paint a rather different picture. We are able to observe that there is a much wider gap between the accuracy and AUC scores in the original distribution than in the balanced distribution. This vindicates a prior assumption, that the greater success of a number models on the original distribution in terms of accuracy can somewhat be attributed to the increased likelihood of classifying the majority class. This is not as relevant to the LSTM models on the original datasets as they still manage to achieve very successful AUC scores.

With regard to the LSTMs, the LSTMs which use GloVe embeddings perform best in terms of accuracy on the Amazon dataset and Word2vec embeddings learned on domain-specific corpora perform best on the Yelp dataset and on the Amazon dataset in terms of AUC. The models using GloVe embeddings narrowly outperform the models using Word2vec embeddings across all tests, which echoes the results on the balanced datasets where GloVe scores, in the majority of cases, outperformed the scores of models using pre-trained Word2vec embeddings. Similarly, models using self-initialised weights slightly lag behind models using pre-trained weights in most cases. Despite this, the features which consist solely of word indices and have no prior knowledge of word meaning, still act as good features for the LSTM. The performance of LSTM models

using self-initialised weights is very stable across both datasets and distributions, indicating that the LSTMs can learn meaningful information from the words in the corpora without having a semantic understanding of the words. The fact that these models do not use pre-trained weights and still outperform the baseline bag-of-words methods, which also have a limited understanding of word meaning, gives credence to the role of LSTMs in tasks which involve modeling sequential data such as text.

6 Conclusion

In this study we have compared bag-of-words and neural network based approaches for sentiment classification. Firstly, we used unordered bag-of-words models, and secondly, we used an LSTM model which can handle sequential data as well as leverage the use of pre-trained word embeddings. From our analysis, for the baseline approaches, the SVM models outperform the Multinomial Naive Bayes classifiers. The LSTM models outperform the bag-of-words models across both metrics for the majority of tests. Nevertheless, bag-of-words models can still perform very well, particularly with respect to their shorter training period. LSTM models using pre-trained GloVe embeddings and Word2vec embeddings learned on domain-specific corpora performed best. In most cases, pre-trained GloVe embeddings served as better features than pre-trained Word2vec embeddings. The strong performance of the models using domain-specific Word2vec embeddings could justify using such an approach provided there is an adequate amount of text to train on.

We also compared our results across two different dataset distributions, a balanced distribution and one which follows the original distribution. While the greatest accuracy was achieved on the original distribution using Word2vec domain-specific embeddings, there was less disparity among AUC scores on the balanced datasets, particularly with respect to the baseline models. The inclusion of sampling measures which balance the distribution of ratings can help ensure the models are less likely to overfit on positive reviews given their higher respective share in the data. The fact that the LSTM models achieved greater AUC scores than the baseline models highlights their ability in NLP tasks. The LSTM models are able to learn more subtle relationships which the baseline models fail to pick up on as evident in their comparative AUC scores.

References

1. Bakliwal, A., Foster, J., van der Puil, J., O'Brien, R., Tounsi, L., Hughes, M.: Sentiment analysis of political tweets: Towards an accurate classifier, Association for Computational Linguistics (2013)
2. Pang, B., Lee, L., et al.: Opinion mining and sentiment analysis. Foundations and Trends® in Information Retrieval **2**(1–2) (2008) 1–135
3. Zhang, W., Skiena, S., et al.: Trading strategies to exploit blog and news sentiment. In: Icwsm. (2010)

4. McAuley, J.J., Leskovec, J.: 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) 897–908
5. Yelp: Yelp Dataset Challenge. `$https://www.yelp.ie/dataset_challenge$` (2017) [Online; accessed 23-June-2017].
6. Pang, B., Lee, L., Vaithyanathan, S.: 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) 79–86
7. Dai, A.M., Le, Q.V.: Semi-supervised sequence learning. In: Advances in Neural Information Processing Systems. (2015) 3079–3087
8. Le, Q., Mikolov, T.: Distributed representations of sentences and documents. In: Proceedings of the 31st International Conference on Machine Learning (ICML-14). (2014) 1188–1196
9. Mikolov, T., Chen, K., Corrado, G., Dean, J.: Efficient estimation of word representations in vector space. arXiv preprint arXiv:1301.3781 (2013)
10. Pennington, J., Socher, R., Manning, C.D.: Glove: Global vectors for word representation. In: EMNLP. Volume 14. (2014) 1532–1543
11. Hochreiter, S., Schmidhuber, J.: Long short-term memory. Neural computation **9**(8) (1997) 1735–1780
12. Graves, A., Liwicki, M., Fernández, S., Bertolami, R., Bunke, H., Schmidhuber, J.: A novel connectionist system for unconstrained handwriting recognition. IEEE transactions on pattern analysis and machine intelligence **31**(5) (2009) 855–868
13. Turney, P.D.: 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) 417–424
14. Socher, R., Pennington, J., Huang, E.H., Ng, A.Y., Manning, C.D.: 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) 151–161
15. Socher, R., Perelygin, A., Wu, J.Y., Chuang, J., Manning, C.D., Ng, A.Y., Potts, C., 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). Volume 1631. (2013) 1642
16. Li, J., Luong, M.T., Jurafsky, D., Hovy, E.: When are tree structures necessary for deep learning of representations? arXiv preprint arXiv:1503.00185 (2015)
17. Cortes, C., Vapnik, V.: Support-vector networks. Machine learning **20**(3) (1995) 273–297
18. Joachims, T.: Transductive inference for text classification using support vector machines. In: ICML. Volume 99. (1999) 200–209
19. Dundar, M., Krishnapuram, B., Bi, J., Rao, R.B.: Learning classifiers when the training data is not iid. In: IJCAI. (2007) 756–761
20. Martin, J.H., Jurafsky, D.: Speech and language processing. International Edition **710** (2000) 25
21. Hochreiter, S., Bengio, Y., Frasconi, P., Schmidhuber, J., et al.: Gradient flow in recurrent nets: the difficulty of learning long-term dependencies (2001)
22. : Lecture notes: Part v2. `http://web.stanford.edu/class/cs224n/lecture_notes/cs224n-2017-notes5.pdf` Date last accessed 22-August-2017.
23. Dozat, T.: Incorporating nesterov momentum into adam. (2016)