1 Introduction

2 Literature Review

2.1 Neural Networks

2.1.1 Recursive Neural Networks

Recursive Neural Networks (RvNN) approach the computation of word vectors by applying a particular compositionality function over a binary tree recursively. It is usually done bottom-up, so each child vector is calculated, which in turn calculates the parent vector. Socher et al. (2013) propose an enhanced version of this network, known as a Recursive Neural Tensor Network, where they use the same tensor-based composition function for all nodes in the binary tree. While Recursive Neural Networks are a powerful tool in natural language processing, as Mikolov et al. (2013) points out, they are often subject to high computational complexity due to their non-linear hidden layers.

2.1.2 Convolutional Neural Networks

Although Convolutional Neural Networks (CNN) are often used for tasks such as image classification, they're also extremely useful for sentiment analysis. Kim, Y. (2014) shows that a CNN trained with one convolution layer achieves excellent results across multiple benchmarks.

2.1.3 Recurrent Convolutional Neural Networks

Lai, S., Xu, L., Liu, K., & Zhao, J. (2015) propose a new type of neural network that takes advantage of recurrent neural network's O(n) time complexity. While RNNs are a biased model, putting emphasis on later words in a corpus rather than earlier words, RCNNs use an unbiased max-pooling layer that gives equal emphasis across the corpus. RCNNs also take advantage of a bidirectional recurrent structure, which produces considerably less noise than a traditional window-based neural network.

2.2 Types of Classification

2.2.1 Logistic Regression

Logistic regression is a statistical method that aims to categorise the dependent variable into a binary category – that is, given an input Y, the likelihood of Y being 0 or 1. In the context of machine learning, this usually means taking feature vectors (n-dimensional vectors that describe learned features relating to the set of training data), and fitting them into this function.

2.3 Types of Distributed Representations of Words and Documents

2.3.1 Word2vec (Shallow Neural Network Implementation)

Word2Vec is a powerful model developed by researchers at Google led by Tomas Mikolov that produces rich word embeddings. It improves on existing work in this space by proposing two new models for learning distributed representations of words. The goal of these new models was to minimise the computational complexity of the original models by removing the non-linear hidden layer from existing feedforward and recurrent neural net language models (NNLM).

Mikolov et al. (2013) propose a continuous bag of words model (CBOW) that is similar to the feedforward NNLM model where the non-linear hidden layer is removed. This reduces the computational complexity, allowing a much larger vocabulary to be used during the training step. It is trained by trying to classify a target word from a window of source words.

They also propose a continuous skip-gram model (SG), which is similar to CBOW, but instead performs the inverse training step by trying to classify source words based on a target word in the same sentence.

Mikolov (2013) suggested during a discussion of the various use-cases for these models that the CBOW model works better for a dataset with short sentences but a high number of samples (i.e. tweets, short reviews), while the SG model works better for a dataset with long sentences but a low number of samples (i.e. small quantity of large documents).

2.3.2 Doc2Vec (Shallow Neural Network Implementation)

Following the research led by Mikolov et al. (2013) that produced Word2Vec, researchers tried to extend the Word2Vec model to produce phrase-level or sentence-level representations. At first a simple approach was used, which simply created a weighted average for all words in a corpus, thus giving a weighted model for a complete sentence. This has weaknesses, as Mikolov (2013) states: “The first approach, weighted averaging of word vectors, loses the word order in the same way as the standard bag-of-words models do”.

The extended work done by Mikolov et al. (2013) borrows the same fundamental concepts that were used to create the Word2Vec models. The only difference being that instead of applying the SG and CBOW models to words in sentences, it is applied to sentences within a corpus.

2.4 Frameworks and Libraries

Existing frameworks and libraries can be used to save time implementing the neural network models previously mentioned. Thanks to their popularity, models such as Word2Vec and Doc2Vec have been implemented in various libraries. Gensim (Rehurek, R., 2014) is one such library that has implementations of the work done by Mikolov et al. (2013) on Word2Vec and Doc2Vec.

2.5 Datasets

Yelp (a popular online review website) offers a comprehensive dataset for academic use. It contains approximately 4.7 million text-based reviews.

2.6 Summary of Research

The research surrounding sentiment analysis highlights its often complex nature. Representing language in a model that can be consumed by a natural language processing tool while maintaining its context can prove difficult. There are multiple approaches to this problem, but it is clear from the research undergone that a neural network backed solution will perform the best. Various types of neural networks each offer their own unique advantages, and can be tuned to increase performance.

Based on the research of representations of words, for training the model, the Doc2Vec CBOW architecture will be used, as it should perform better than the SG model due to its high-volume of samples that are short in length (see Yelp dataset). Although there are existing libraries for the Doc2Vec model, it will still be important to know how the algorithms are implemented at a naive level.

In order to train the models in a neural network, it is useful to have a large text corpus to train with. Thankfully, Yelp (a popular online review website) have provided a dataset for academic use. It contains approximately 4.7 million text-based reviews that can be used for training and testing the models.

A logistic regression classifier will be used to classify reviews into either good or bad categories. A logistic regression classifier is particularly useful for binary classification tasks, so should perform well on the Yelp dataset.

The purpose of this research and project is to produce a sentiment analysis application tailored towards businesses with an online presence. Businesses often only get a very uninformative high-level insight into their reviews (i.e. star ratings, number of 1-5* reviews etc.).

These businesses would benefit from an application that uses sentiment analysis to analyse their reviews and output meaningful data that they can use to improve their services. As a proof of concept, this will be limited to predicting the sentiment of a

review as either ‘good’ or ‘bad’, and providing related reviews based on the language used in a review.

3 Development

In this chapter, we explore the development process that lead to the production of the sentiment analysis application. It will focus on relevant ideas learned while studying existing literature and applying them to the domain of sentiment analysis.

3.1 Experimental Development

Now that appropriate models for representing reviews in vector space and datasets had been identified, experimental development was undergone to find the most appropriate platforms and libraries for sentiment analysis.

3.1.1 Tensorflow

Tensorflow is an option for developing neural networks. The main benefit of using Tensorflow is that it abstracts a lot of the complexity around developing a platform to create a neural network. Things such as types of neural networks, regression models, training models and loss models are made easy to implement thanks to many of Tensorflow’s high-level libraries.

3.1.2 Doc2Vec in Tensorflow

Once a basic understanding of the Tensorflow library was attained, the viability of Tensorflow for implementing the work of Mikolov et al. (2013) on the Doc2Vec model was tested. Unfortunately, Tensorflow does not provide an implementation of the Doc2Vec model meaning that the model had to be implemented from scratch.

3.1.3 Gensim Doc2Vec

Alternatively, Gensim (an open-source vector space modelling library) provides a highly optimised implementation of the Doc2Vec model. Additionally, Gensim’s implementation of the Doc2Vec model provides convenience functions for inferring new vectors, finding similar documents, and storing models for future use.

3.1.4 Conclusions of Experimental Development

Although the algorithm implementation was fairly trivial to implement and Tensorflow provides some out-of-the-box convenience libraries for dealing with large amounts of data, the implementation was inefficient for the large size of the Yelp dataset. Gensim’s library clearly provides a more appropriate solution for our problem domain, appealing to developers wishing to handle large sets of training data.

Gensim’s training cycle is much faster than Tensorflow’s. This is most likely due to Gensim optimising their implementation of

Doc2Vec by providing efficient file streaming via their 'utils' library and also by using NumPy vectors for storing feature vectors within the model.

3.2 Developing and Training the Model

This section will describe the development process that was undergone to produce the Yelp Doc2Vec model. Figure 3.2.x shows – at a high level – the development process from start to finish. Firstly, it is established how Yelp's JSON reviews will be parsed into a format digestible by Gensim's Doc2Vec implementation. Secondly, the configuration of the training process is described, including how parameters were tuned to improve the accuracy of the model. Finally, now the model has achieved adequate accuracy, the process of fitting the model's feature vectors into an appropriate classifier is described.

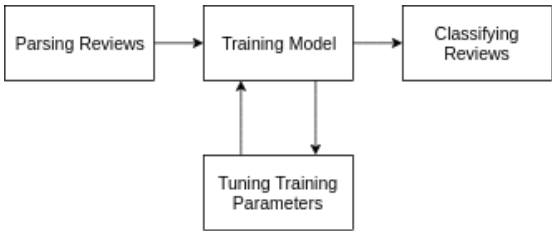


Figure 3.x shows the development cycle for producing the Yelp Doc2Vec model.

3.2.1 Parsing Yelp Reviews

Doc2Vec builds word embeddings by taking in a large corpus of text(s) and produces large dimensional feature vectors describing the contextual relationship between words in documents and documents themselves.

As with any machine learning process, data should be appropriately partitioned into training data and testing data. Yelp's dataset contains ~ 5.2 million reviews, giving us a more than sufficient amount of data. Mikolov et al. (2013) used approximately 912,000 sentences to train their model. To ensure similar results, a similar amount will be used. It is worth noting that the separation of testing and training data is important. Allowing the accuracy to be tested with previously seen and trained on data skews the accuracy of the model as it will be weighted/biased towards that data. Instead, it is required to use unseen data of which we already know the sentiment of, and use that to report our accuracy.

Yelp reviews are stored in the following JSON format:


```
{
 "user_id":"u0LXt3Uea_GidxRW1xcsfg",
 "review_id":"FKu4iU62EmWT6GZXPJ2sgA",
 "text":"I too have been trying to book an appt to use my voucher - it's been months and countless of phone calls - no response yet.\n\nAgree with the buyers beware warning. I only wish reviews on this place was posted previous to my purchase of this voucher.",
 "business_id":"fdnNZMk1NP7ZhL-YmidMpw",
 "stars":1,
 "date":"2012-10-23T00:00:00.000Z"
}
```


Figure 3.x – the JSON structure of a Yelp review.

Gensim's Doc2Vec model requires each sentence to be stored in a particular fashion.

Each sentence should be an array of words that form the sentence, with a corresponding label identifying the sentence. For training purposes, bad reviews are labelled bad_*i* where 'i' is the index of the review in the list of reviews. The same process is also done for good reviews. The review's ID is also transferred to a second label.

```
[['I', 'too', 'have', 'been',
'trying', 'to', 'book', 'an',
'appt', 'to', 'use', 'my',
'voucher', '-', "it's", 'been',
'months', 'and', 'countless',
'of', 'phone', 'calls', '-',
'no', 'response', 'yet.',
'Agree', 'with', 'the', 'buyers',
'beware', 'warning.', 'I',
'only', 'wish', 'reviews', 'on',
'this', 'place', 'was', 'posted',
'previous', 'to', 'my',
'purchase', 'of', 'this',
'voucher.'],
['bad_1',
'FKu4iU62EmWT6GZXPJ2sgA']]
```

Figure 3.x – a 'LabeledLineSentence' of the review in Figure 3.x.

Gensim refers to this as a 'LabeledLineSentence'. Unfortunately, their implementation of this class only supports loading from a text file with sentences separated by new lines. To avoid having to transform the ~ 4.2GB JSON file, our own 'YelpLabeledLineSentence' class is used that makes loading the Yelp dataset into this format easier by optimising the loading of a large JSON file using Python's JSON library and iterators, meaning the whole corpus is not loaded into RAM. This should ensure we can train on corpus sizes similar to the aforementioned 900,000 reviews.

It has been reported that repeatedly training the same data in the same order decreases the learning rate over each iteration. This is because the model's features are being 'reinforced' by observing the same contextual relationships between sentences in the same order repeatedly. An analogy for this is a heat map where the same data is layered over and over. To remedy this, the training data will be randomised on each iteration, providing near-unique sentence comparisons over each iteration.

Furthermore, as a logistic classifier will be used, our training data will be sorted into two features - 'good' and 'bad'. 'Good' describing any reviews with a star rating of 4 or more, and 'bad' describing reviews with a star rating of 2 or less. When the model is later fitted into the logistic regression classifier, we can maximise the accuracy of its predictions by providing well fit feature vectors. This is described in more detail in the Classifying Reviews section of this chapter.

To summarise, there is now an efficient way of parsing and loading large sets of Yelp reviews into the Doc2Vec model. The bespoke 'YelpLabeledLineSentence' class provides efficient data loading via Python iterators, convenience functions for retrieving and labelling sentences for our two features 'good' and 'bad, and the ability to randomise the training set for each training iteration.

3.2.2 Training Parameters

Gensim's Doc2Vec model provides parameters that are analogous to the

definitions in the original paper, they are as follows:

min_count – The minimum occurrences of a sentence label in the corpus to be considered in the model.

window – The maximum distance between the current and predicted word within a sentence.

size – The dimensionality of the feature vectors (how 'detailed' do we want our feature vectors to be).

sample – The threshold for configuring which words are randomly downsampled.

negative – Specifies how many 'noise words' will be drawn during negative sampling.

iter – The number of training iterations (epochs).

workers – The number of threads to train the model with. Provides faster training on multi-core machines.

3.2.3 Tuning Parameters

Finding the appropriate configuration was a mixture of drawing from well-established default values, and experimental trial-and-error tests. The ideal configuration would provide the best possible accuracy at the lowest possible computational cost. Comparing large n-dimensional vectors is computationally costly, especially with a large data set.

Number of Reviews

As the time taken to train the model correlates with the amount of training reviews used, accuracy was logged for each increase of training reviews to find an ideal amount. An ideal amount would provide sufficient accuracy at the smallest possible number of reviews. This will be helpful when tuning the iterations parameter, as a smaller number of reviews will reduce the time taken for each iteration.

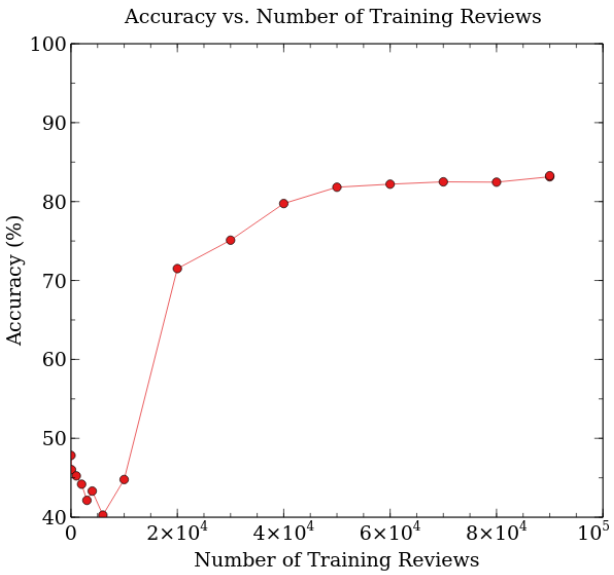


Fig 3.x shows accuracy (%) for a number of training reviews. It can be inferred that increasing the number of training reviews past 6×10^4 provides negligible improvement of accuracy. A table of results can be found at appendix x.

Min_count

"Min_count" was borrowed from the Word2Vec model that Doc2Vec was built upon. It originally meant the minimum threshold that words in a sentence should appear to be included in the model. When translated to Doc2Vec, this parameter has lost its usefulness, as sentences that appear within documents will most likely only ever appear once (they are unique). This leaves no choice but to set this value to 1.

Window

The maximum distance between the current and predicted word within a sentence. To provide rich feature vectors, there should be sufficient overlap between word comparisons within a sentence. The average length of a Yelp review is approximately 30 words, across all stars, therefore a window size of 10 should suffice.

<graph of average length of review>

Size

"Size" describes the dimensionality of the feature vectors produced for each document. It is important to set a size that sufficiently captures enough detail to produce meaningful classification, but not so high that the model is 'over-fit'. It was observed that a dimensionality of 10 is ideal.

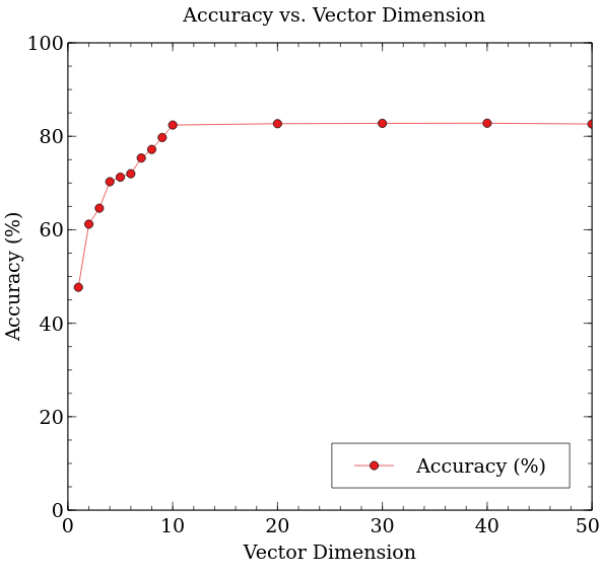


Figure 3.x shows accuracy for a particular feature vector dimension. It can be inferred that 10 is an ideal vector dimension.

Sample

When training over a large corpus, it may be desirable to downsample higher-frequency words within the corpus, so that they don't unduly bias the model. In practice, as Doc2Vec traverses through document sentences, it looks to see if the current context word in the sentence has been sampled against other words before.

Very frequent words tend to not provide much contextual information compared to rare words. For example, having the model consider the co-occurrence for the words "good" and "food" is much better than comparing the words "the" and "food" as "the" commonly co-occurs with nouns in sentences. According to Mikolov et al. (2013), An ideal value for sampling is usually 1×10^{-5}.

Negative

The idea of Doc2Vec is to maximise the similarity of the vectors for sentences that appear close together, and minimise the similarity of the vectors for sentences that appear far apart.

$$\frac{V_c \cdot V_w}{\sum V_{c'} \cdot V_w}$$

Fig 3.x shows the formula for computing the similarity between a document context V_c and a target sentence V_w. The similarity is summed across all other contexts $V_{c'}$.

This formula does not consider the computational complexity of calculating the similarity for all other contexts. Instead, it is better to only consider a few random extra contexts, reducing the time required to train the model.

Again, according to Mikolov et al. (2013), an ideal value for negative is 5.

Iter

"Iter" describes the number of passes over the corpus used to train the model. The most benefit is had when training against a relatively small corpus, where the number of unique documents is low, as it enables the model to produce models comparable in detail to those produced by much larger corpora.

While increasing the iterations provides improved relationships within the model, Mikolov et al (2013) states that "training a model on twice as much data using one epoch [iteration] gives comparable or better results than iterating over the same data for three epochs". For a data set as large as Yelp's, just 1 iteration should suffice.

Workers

Gensim's Doc2Vec library can parallelise the training process by spawning multiple worker threads on multicore machines. The PC training the Yelp model had 4 available cores.

These parameters should provide the most optimal performance for training the Yelp model.

Parameter	Value
Number of reviews	6×10^{4}
min_count	1
window	10
size	300
sample	1×10^{-5}
negative	5
iter	5
workers	4

Figure 3.x shows each Doc2Vec parameter with its corresponding value.

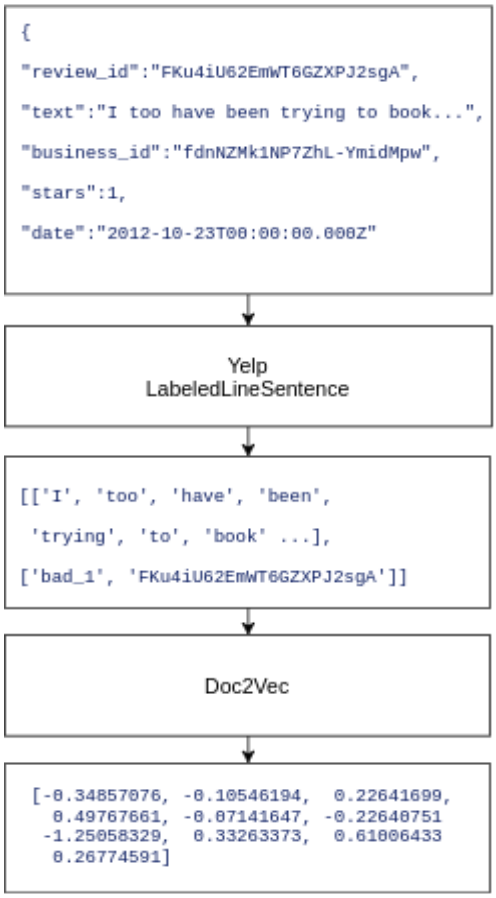


Fig 3.x summarises the steps taken in this section of the development process. Each review was parsed and loaded into a 'YelpLabeledLineSentence'. Then, Doc2Vec trained its model with the aforementioned parameters, producing n-dimensional feature vectors for each review.

3.2.4 Classifying Reviews

There are now n-dimensional vectors describing the contextual relationship between a specific review and the rest of the reviews. In order to take advantage of these vectors, they should be fit to a particular classifier. Classifiers are often derived from regression models found in statistical methods. Two notable examples worth considering for this classification task are linear regression and logistic regression.

Linear regression is useful when mapping probabilities of a continuous dependent variable. For example, what is the most likely amount of ice-creams sold on a day with the temperature x? However, this type of regression is not suited to the Yelp data set, as we are trying to measure the probability of our reviews falling into the binary categories "good" or "bad". There are no intermediary values between these categories for a linear regression model to make sense.

Instead, a logistic regression is much more appropriate. Logistic regression is used when the dependent variable is categorical (good/bad, yes/no). This fits the Yelp data set better – for a review, predict the probability of it being a good review or a bad review.

In order to fit our classifier, the feature vectors are recalled from the model and placed in the first index of a two-dimensional array. The second index of the two-dimensional array is for the Y value of the classifier. In logistic regression this is a 1 or a 0. As the sentiment of the training reviews is known, we can accurately specify the Y value for these reviews. 1 represents a good review, whereas 0 represents a bad review.

```
[[[-0.34857076, -0.10546194,...], 1]
…]
```

Figure 3.x shows a 2D classifier array entry containing a feature vector and corresponding Y value.

The logistic classifier's method is then called on this two-dimensional array that contains our feature vectors and corresponding Y values like so:

```
classifier.fit(train_arrays,
train_labels)
```

Now that the logistic regression classifier has been fit with our training data, the classifier is capable of inferring feature vectors of previously unseen reviews. This can be used to check the accuracy of our model. If the sentiment prediction (good/bad) of previously unseen reviews is known, the classifier can score the accuracy of its predictions by calculating the probability of the review's inferred feature vector belonging to the 'good' or 'bad' category, and cross-checking that with the known categorisation.

To do this, unseen reviews from the end of the Yelp dataset are imported into the algorithm. The same two-dimensional array structure is formed, with the first index being the inferred feature vector of the unseen review (calculated like so):

```
model.infer_vector(review[0])
```

Figure 3.x shows how vectors are inferred for unseen reviews. Where 'review[0]' is the review text from 'YelpLabeledLineSentence' (see figure 3.x).

As the sentiment classification is already known for these unseen test reviews, the Y values (0/1) are placed in the second index of this two-dimensional structure, forming an array structure that is exactly the same as used in the fitting stage of this process (see figure 3.x).

For accuracy testing purposes, 1000 good reviews and 1000 bad reviews were scored against the classifier. The following accuracy was reported.

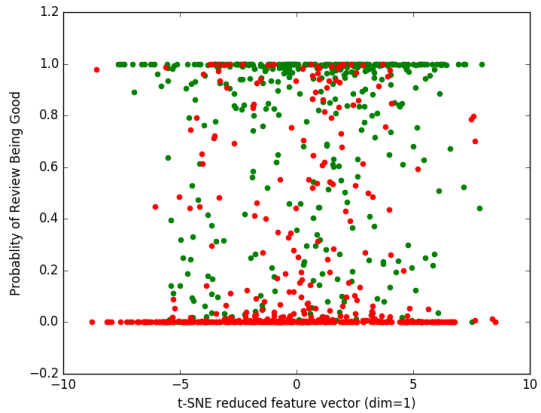
Sentiment	Review Size	Accuracy
Good	1000	69.2%
Bad	1000	85.2%

Figure 3.x shows the reported accuracy of the model using logistic regression classification.

This data can be visualised by plotting the probability of each review belonging to the good category against each review's feature vector. However, there is an issue with this, as each feature-vector has 10 dimensions. This is a common problem when trying to visualise machine learning relationships as so many dimensions are impossible to visualise graphically.

Fortunately, t-distributed stochastic neighbor embedding (t-SNE), developed by Van Der Maaten, L. J. P., & Hinton, G. E. (2008) can be used to reduce a feature vector to any n-dimension.

If the 10-dimensional feature vectors are reduced to just one dimension, they can be plotted against the probability of belonging to the ‘good’ category.



3.2.5 Storing Classified Reviews

- Large dataset (~100,000 reviews + ~150,000 businesses)
- MongoDB (easy in both Python and Node.js)
- Calculating sentiment before or after insertion to database? (before - much quicker lookup times)

3.2.6 Predicting Unseen Documents

- Inferring vectors of user input
- What is an unseen doc?
  - Keywords in docs (i.e. food, drink, service)
  - Full text corpus (i.e. a new review)
- Finding related documents

3.3 Developing the User Interface

Evaluation

Confusion matrix

Conclusion

Bibliography

Socher, R., Perelygin, A., & Wu, J. (2013). Recursive deep models for semantic compositionality over a sentiment treebank. Proceedings of the …, 1631–1642. Retrieved from http://nlp.stanford.edu/~socherr/EMNLP2013_RNTN.pdf

Santos, C. N. dos, & Gatti, M. (2014). Deep Convolutional Neural Networks for Sentiment Analysis of Short Texts. Coling-2014, 69–78. Retrieved from http://www.aclweb.org/anthology/C14-1008

Kim, Y. (2014). Convolutional Neural Networks for Sentence Classification. https://doi.org/10.3115/v1/D14-1181

Lai, S., Xu, L., Liu, K., & Zhao, J. (2015). Recurrent Convolutional Neural Networks for Text Classification. Retrieved from https://www.aaai.org/ocs/index.php/AAAI/AAAI15/paper/view/9745/9552

Mikolov, T., Corrado, G., Chen, K., & Dean, J. (2013). Efficient Estimation of Word Representations in Vector Space. Proceedings of the International Conference on Learning Representations (ICLR 2013), 1–12.

Mikolov, T., Sutskever, I., Chen, K., Corrado, G. S., & Dean, J. (2013). Distributed Representations of Words and Phrases and their Compositionality. In C. J. C. Burges, L. Bottou, M. Welling, Z. Ghahramani, & K. Q. Weinberger (Eds.), Advances in Neural Information Processing Systems 26 (pp. 3111–3119). Curran Associates, Inc. Retrieved from http://papers.nips.cc/paper/5021-distributed-representations-of-words-and-phrases-and-their-compositionality.pdf

Tomas Mikolov. (2013). de-obfuscated Python + question - Google Groups. Retrieved November 25, 2017, from https://groups.google.com/forum/#!msg/word2vec-toolkit/NLvYXU99cAM/E5ld8LcDxlAJ

RaRe Technologies. (2016). Doc2Vec Tutorial on the Lee Dataset. Retrieved from https://github.com/RaRe-Technologies/gensim/blob/develop/docs/notebooks/doc2vec-lee.ipynb

RaRe Technologies. (2015). Gensim Doc2vec Tutorial on the IMDB Sentiment Dataset. Retrieved from https://github.com/RaRe-Technologies/gensim/blob/82c394a9085d583e8a75c2bb32ecd37cf61236f0/docs/notebooks/doc2vec-IMDB.ipynb

Le, Q. V., & Mikolov, T. (2014). Distributed Representations of Sentences and Documents, 32. Retrieved from http://arxiv.org/abs/1405.4053

Abadi, M., Agarwal, A., Barham, P., Brevdo, E., Chen, Z., Citro, C., … Zheng, X. (2016). TensorFlow: Large-Scale Machine Learning on Heterogeneous Distributed Systems. Retrieved from https://arxiv.org/abs/1603.04467

Rehurek, R. (2014). gensim: Topic modelling for humans. Retrieved from http://radimrehurek.com/gensim/

Tensorflow. (n.d.). Vector Representations of Words | TensorFlow. Retrieved November 26, 2017, from https://www.tensorflow.org/tutorials/word2vec

Yelp Inc. (n.d.). Yelp Dataset. Retrieved November 25, 2017, from https://www.yelp.com/dataset

Van Der Maaten, L. J. P., & Hinton, G. E. (2008). Visualizing high-dimensional data using t-sne. Journal of Machine Learning Research, 9, 2579–2605. https://doi.org/10.1007/s10479-011-0841-3