**1. Introduction**

The surge of usage in internet media and e-commerce, hospitality and employment platforms like Amazon, Airbnb and Glassdoor has resulted in troves of reviews for commodities we may want to purchase, accommodation we want to book, and enterprises we wish to work at. Speculating the polarity of free-form text by using machine learning algorithms is highly important in this information era. While there have been many successful attempts and good results at binary sentiment analysis of reviews, fewer attempts have been made to classify texts into ratings as review rating prediction is an albeit hard problem in machine learning (understandable given how free text reviews do not always match perfectly with star rating). The aim of this project is to predict rating based on user’s review about top companies in the range of 1 to 5. This research can potentially be applied to settings where all reviews about top companies is available, but the corresponding ratings are missing. We also explore other features such as work life balance, culture values, career opportunities, company benefit and senior management ratings and use them to improve performance of the model built only on review. Not only giving example, we would also prove the hypothesis saying that the relationship between review and rating is not obvious from the perspective of sentiment analysis.

**2. Background**

Online user reviews have become an essential part of websites that run e-commerce, hospitality, employment, or restaurant services. Users can deliver their opinions regarding to enterprises, products and services through text-based comment and rating. According to a recent study (Chen, Wu, & Yoon, 2004), online user reviews generally have a strong influence on users’ decision as well as earnings and losses of enterprises.

However, there are hundreds or thousands of reviews on products, services or businesses. Given an average human reading speed is around 200 to 250 words per minute (Smith, 2016). If there are 1000 reviews and each review average around 20 words, 100 minutes are needed for a user to read all the reviews, which would be very time consuming.

In fact, users tend to see the rating first before deciding whether to read the reviews although the relationship between review and rating is ambiguous. A review text cannot reflect the corresponding rating perfectly and vice versa. Nevertheless, it is an undeniable fact that rating also has a significant impact on users’ decision. If ratings given by majority of the reviewers are relatively low (1 or 2), there is a high chance that users will decide not to purchase/join the product/company before reading the reviews.

On the other hand, if the ratings in most of the reviews are extremely high (5), users may get interested in it and decide to look over the reviews to find out the reasons that impress the reviewers to give such high ratings. In reality, it is common for reviewers to leave comments and forget to give ratings, which causes users to spend additional time reading reviews. If there exists a model that can predict the rating based on the review, the time it takes for users to read the reviews one by one can be greatly reduced.

In this report, this review rating prediction problem is considered as a multi-class classification problem in machine learning, where each rating represents a label. In the following section, we would describe what types of machine learning classification algorithms are used to solve this multi-class classification problem and the reason of using them, followed by project management and process, and discuss our outcomes and findings.

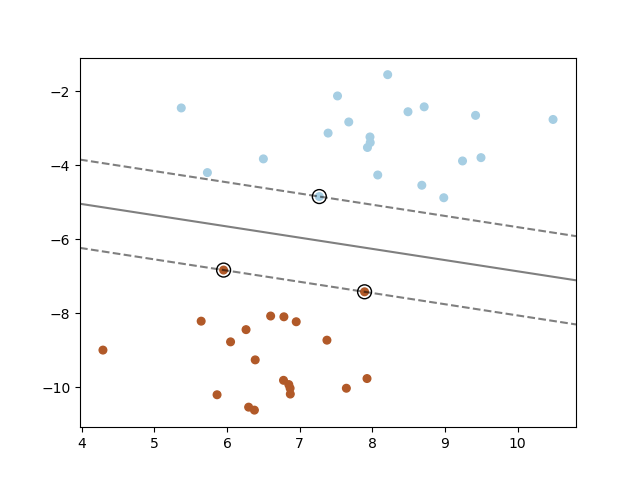
**3. Methodology**

**3.1 Model description**

**3.1.1 Support Vector Machine (SVM)**

Support vector machines are supervised learning models which are based upon the Structural Risk Minimization principle from computational theory: it attempts to find a hypothesis h, in which the minimal true error is achieved (Vapnik, 2013).

The mathematical formulation behind the classification task can be summarized as attempting to construct a hyper-plane or set of hyper-planes in a high- or infinite dimensional space, which allows for the best separation between data points (Pedregosa et al., 2011).

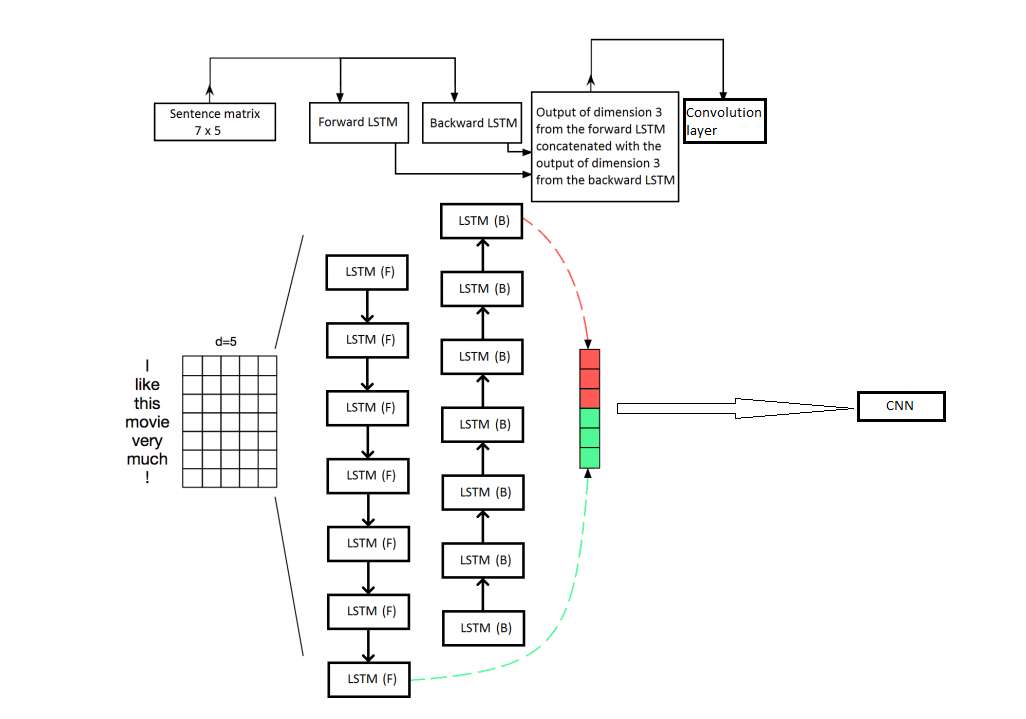


*Figure 1: The maximum-margin hyperplane and margins for the linear SVM (Pedregosa et al., 2011).*

**3.1.2 Long-Short Term Memory Neural Networks**

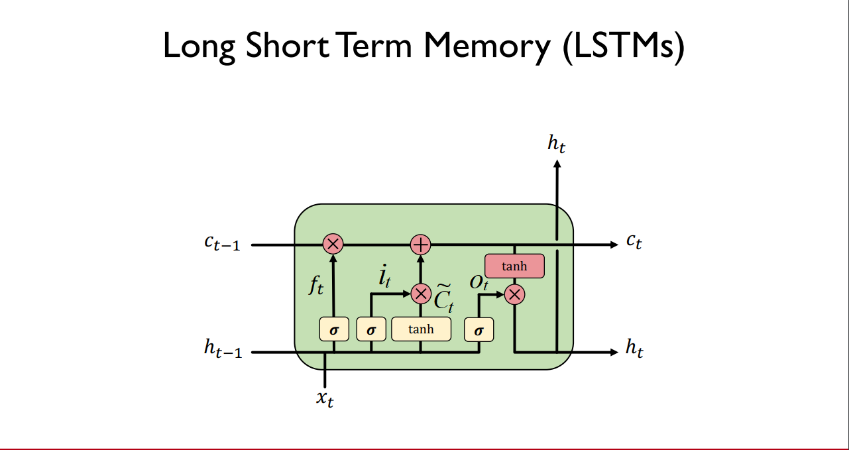
Long-Short Term Memory (LSTM) is a type of recurrent neural networks (RNN) developed to deal with the exploding and vanishing gradient problems often occurred during the back propagation training stage of classic recurrent neural networks (RNN) (Hochreiter, 1998) by having more complex structure within the recurrent unit.

With same set of weights shared across every time step, LSTM are really well-suited for handling variable length sequential data like text, audio, video and time series, and they are really good at learning long-term dependencies (relevant information is separated by a lot of irrelevant data from the place it is needed). Thus, LSTM are great for this review rating prediction problem in which they can take as input a review, which composed by a sequence of words, and output a rating that’s associated with that review. Our LSTM architecture is almost is illustrated below.



*Fig. 2: Smaller version of our LSTM architecture. From (Cliche, 2017) with minor modifications.*

A LSTM unit contains four different interacting layers that control the flow of information, which are cell state, forget, input and output gates. An illustration of LSTM unit is shown below.



*Fig. 3: A single LSTM unit. From (Soleimany, 2019) with minor modifications.*

For each review, each word in that review is mapped to its word vector and subsequently fed into the LSTM unit along with the previous hidden state to compute the next hidden state. To make it clearer, the hidden state *h* at time step *t* can be computed by (Zaremba, Sutskever, & Vinyals, 2014):

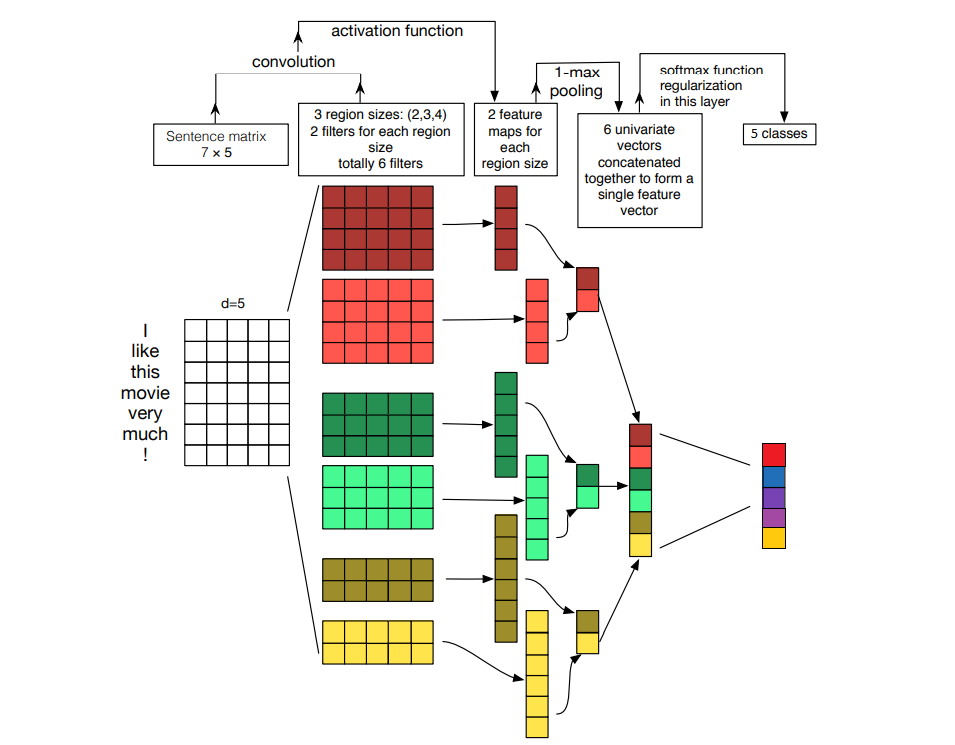
is our *d* dimensional word vector input (above case d = 5, our case d =300), is the previous hidden state (previous hidden state at time step 0 is just vector of zeros), and are weight matrices that having shape of *m x d* and *m x m* respectively, where *m* is output dimension of the LSTM (above case 3, our case 50). are bias terms, is the forget gate that decides what information should be retained or thrown away, is the input gate that decides what information should be added to the cell state, is the candidate layer that holds possible values to add to the cell state, is the output gate decides how much of the internal state is exposed to the output, is the cell state that carries relevant information throughout the processing of the sequence (like “memory” of the network), is the sigmoid function and is element-wise multiplication. Regular LSTM struggles to take into account the future information because it processes the sequence only in a single direction. Therefore, we are using bidirectional LSTM instead. The greatest advantage of using bidirectional LSTM is that when it runs forwards and backwards, we are able to preserve information from both the past and the future. Outputs of the LSTM that runs forwards and backwards are then concatenated to form a 2*m* dimensional vector (above case 2\*3= 6, our case 2\*50 = 100). We finish the LSTM part by passing this vector to our CNN.

**3.1.3 Convolutional Neural Networks**

Convolutional Neural Networks (CNN) is a type of deep neural networks invented for computer vision tasks like image classification and object detection. However, they have subsequently been applied to various NLP tasks and have been shown to be able to perform remarkably well in sentence classification (Kim, 2014). The architecture of our CNN is mostly identical to the CNN proposed by Kim (2014), but with the choice single channel architecture (1D convolution layer) as opposed to Kim’s multichannel architecture (2D convolution layer).

One of the reasons is that single channel architecture is computationally less costly than multichannel architecture. Single channel requires only one embedding layer, and the output shape of that embedding layer also matches with the input shape required for Conv1D. Thus we can directly feed the output of the embedding layer into Conv1D without reshaping it. On the other hand, multichannel architecture requires two channels of pre-trained word embeddings, one’s weights remain static throughout the training process, whereas the other one allows model to fine-tune its weights through back propagation. Besides, the output shapes of both embedding layers are incompatible with the input shape required for Conv2D. In order to feed them into Conv2D, we have to first concatenate the outputs of the two embeddings layers and then reshape it into a 4D tensor with a shape of (batch size, rows (sequence length), columns (output dimension of previous layer), channels).

These procedures become more complex when a model stacks different class of deep neural networks together. Especially in our LSTM-CNN model, we need to have two LSTM layers where each of them corresponds to one embedding layer, and then repeat the concatenation and reshape processes as described above. Most importantly, we tested that these additional procedures do not make multichannel architecture superior to single channel architecture as their final results are not significantly different, which is in compliance with Kim’s (2014) conclusion that the results of multichannel and single channel models are mixed after his experiment. A smaller version of our model is illustrated below.

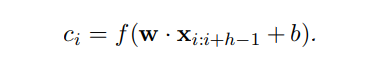


*Fig. 4: Smaller version of our CNN architecture. From (Zhang & Wallace, 2015) with minor modifications.*

In fact, the input fed into our CNN is the output of our LSTM layer (encoded original input). For simplicity, we will ignore the existence of previous LSTM layer and assuming that the input fed into our CNN are employee review, which are tokenized into words and each word is mapped to a vector of real numbers that represents the word. Thus, a review can be mapped to a matrix of size *n x d*, where *n* is the total number of words in the review and *d* is the dimension of the word vector (above case *d* = 5, our case *d* = 300).

However, the number of words in each review may vary widely, and we have to feed a stream of reviews that has the same number of words. To address this problem, we make the shorter reviews as long as *n’* by filling the shortfall by zeros and also truncate the longer reviews to the *n’*, where we chose *n’* = 200. With this zero padding strategy, we now have a sentence matrix ***X*** with the shape of *n’ x d* (above case 7*x*5, our case 200*x*300) for all reviews.

We then apply three 1D convolution operations, where each of them having a filtering matrix with different region size, over the reviews to extract local features. Since each word in a review is mapped to its word vector, the filtering matrix ***W*** would have a shape of *h x d* (above case [2,3,4]*x*5, our case [3,4,5]*x*300) *in* order to overlay across the word vectors, where *h* is the region size that representing the number of words we want to apply the filtering matrix on to produce a new feature. Following equation provided by Kim (2014) shows how a feature is generated:



*Fig. 5: Convolution operation (Kim, 2014)*.

Here, ***W*** are element-wise multiplications, where each value in the sentence matrix is multiplied by the corresponding value in the filtering matrix. These values are all summed up once the element-wise multiplications has finished.

To obtain the feature, we add a bias term and apply ReLU function to introduce non-linearity to the network (we have only been computing linear operations like element-wise multiplications and summations throughout the convolution operation). This filter continues to overlay across each possible window of wordsin the review to generate a vector of features that can be used to form a feature map ***C***.

Recall that we apply three 1D convolution operations, where each of them having a filtering matrix with 3, 4 and 5 region size respectively. The reason of doing this is to allow the CNN to detect contiguous sequence of 3, 4 and 5 words (like n-grams). We also use 50 filtering matrices for each region size to learn multiple/different features (so the total number of filters are 150 in our case).

After the convolution layer, we apply a max pooling operation to extract the most important (highest value) feature for each feature map ***C***, regardless of where it appears in the review. We then concatenate all the extracted of each filter to form a *z* dimensional single feature vector, where *z* is the total number of filters (above case 6, our case 150).

This vector is then fed into our first densely connected neural network layer that encodes the input into 50 dimensional output vector, and is subsequently passed into the softmax activation function to generate the probability distribution over classes. To generalize our model, we add two dropout layers with 0.1 dropout rate, one after the max pooling operation and the other one after our first densely connected neural network layer.

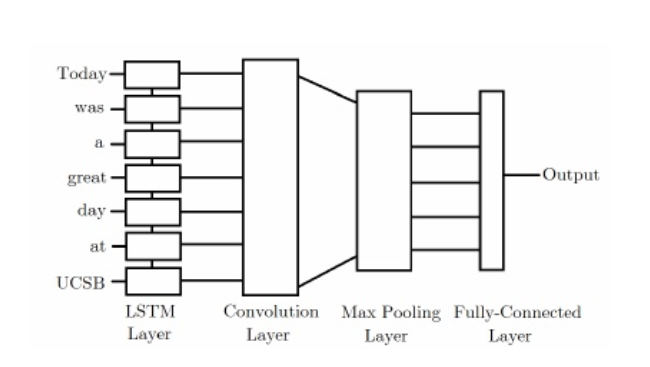
**3.2 Intuition**

**3.2.1 Support Vector Machine (SVM)**

The intuition behind using SVM for the text categorization task comes down to its characteristics of being able to handle high dimensional input space, performing well with sparse document vectors, and the problem itself often being linearly separable (Joachims, 1998).

Input features from reviews are generally very high-dimensional: the representation scheme using TF-IDF in our dataset containing 10484 dimensions. SVM has the advantage of being able to handle such features space, due to its overfitting protection characteristic, compared to other classifiers, namely naïve Bayes model. Researches has also shown that SVM are suited for problems with sparse instances due to its use of “additive” algorithms (Kivinen, Warmuth, & Auer, 1997). The findings from the Ohsumed corpus has shown the text categorization problems are linearly separable in nature, and thus, we believe SVM is an appropriate model for the task (Baeza-Yates & Ribeiro-Neto, 1999).

**3.2.2 LSTM-CNN**

 *Fig. 6: LSTM-CNN Model* (Sosa, 2017).

The advantage of using LSTMs when dealing with text analysis is that the network are able to store the information it has read previously and can capture the historical contextual information of inputs that is important for making accurate predictions. Especially in our case, each input consists of positive and negative reviews. LSTM might be able to capture reviews that possess sentimental changes such as “Working environment is good, but having zero work life balance”. For CNN, we expect it to discover and learn patterns from well-structured text that would otherwise be lost in a regular feed-forward network. For instance, CNN might be able to capture that using “down” in the phrases like “down to earth person” is actually carrying positive sentiment contrary to other phrases like “look down on”. In addition, we also expect CNN to extract these important features independently of where these features locate in the review. Through combining LSTM and CNN, our model would first receive the word vector for each word in the review as input and fed into the LSTM layer. The intuition is that the outputs from LSTM of different time steps will retain information of any previous inputs, meaning that the LSTM is actually producing a new encoding for the original input. Subsequently we feed this encoded inputs into our convolution layer to discover local features. Then the outputs from convolution layer are pooled to smaller dimensions to extract the most important features and finally fed into our densely connected layer to predict the label.

**3.3 Datasets and Experimental Setup**

**3.3.1 Google, Amazon, and more Employee Reviews Dataset**

The employee reviews data used for this analysis was downloaded from the Kaggle Datasets and it was sourced from Glassdoor, a website where current and former employees anonymously review companies and their management. The dataset contains around 67,000 employee reviews for Google, Amazon, Facebook, Apple, Microsoft and Netflix. Each of the review has an overall rating, which is on the scale of 1-5.

**3.3.2 Preprocessing**

**3.3.2.1 SVM**

TFIDF = Transform documents into representation suitable for SVM’s learning algorithm. The input data (text reviews) must be transformed into suitable representations before being fed into the classification model. Researches into the Information Retrieval field have provided us with various representation units (Yang & Pedersen, 1997).

In our research, we have decided to use the term frequency-inverse document frequency (TF-IDF) weighting scheme due to its characteristics of treating frequently occurring words (stop words) to be less important than other distinguishing words and giving more priority to more distinguishing rare words (Ramos, 2003). This technique ultimately allows for the filtering of our initial dataset to retain relevant features and improving generalization accuracy.

While going through our dataset, we have also noticed that emoticons has been often used by the reviewers to further express their sentiments and emotions. In the attempt to capture these semantics, we substituted these emoticons with their corresponding words, such as “😊” to smile.

**3.3.2.2 LSTM-CNN**

Before feeding the reviews to our deep learning model, they are pre-processed using the following procedure:

i) Drop rows with none or NA values

ii) Concatenate pros review and cons review

iii) Emoticons such as :), :(, :’(, and XD are replaced by words like smile, sad, cry.

iv) Remove all digits and non-alphabetic characters such as “!”, “#”, “$”, “%”, and so on.

v) All reviews are converted to lowercase.

**3.3.3 Hyperparameters**

**3.3.3.1 SVM**

**Non-linear vs. linear (Choice of kernels – linear, RBF, sigmoid)**

SVM employs techniques to allow for classification of non-linear problems. There exists plenty of kernels to map data to higher dimensional space for categorizing non-linear data, namely polynomial kernels, Gaussian kernels and so forth (J. Fan, Heckman, & Wand, 1995). However, as mentioned above, due to the nature of textual data being linearly separable and having many features, we have decided to proceed with the experimentation using linear kernel for our SVM.

Also, optimization of parameters with the non-linear kernels would require much longer time with the size of our dataset (~60,000 rows), roughly an hour for the training of model, making exploration impractical.

**Liblinear vs. LibSVM**

The current popular library choices for SVM are LIBSVM and LIBLINEAR (Chang & Lin, 2011; R.-E. Fan, Chang, Hsieh, Wang, & Lin, 2008). In the paper for LIBLINEAR (R.-E. Fan et al., 2008), it was stated that “LIBLINEAR is very efficient for training large-scale problems.

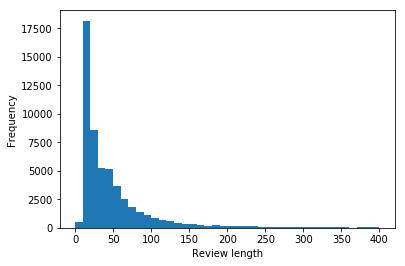
For example, it takes only several seconds to train a text classification problem from the Reuters Corpus Volume 1 (rcv1) that has more than 600,000 examples. For the same task, a general SVM solver such as LIBSVM would take several hours.”

During the training of both models, it was observed that the implementation of SVM through LIBSVM requires over an hour (63 minutes) for the whole dataset, as opposed to the 4 seconds using the LIBLINEAR library. The negligible difference in accuracy and the large reduction in training time has incentivized us to use the LIBSVM implementations of the SVM classifier model.

Other hyperparameters tuning through APIs has been used in attempt to increase the overall performance of the SVM classifier.

**3.3.3.2 LSTM-CNN**

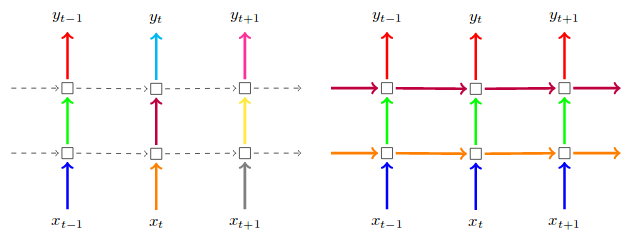
Since we need to feed a stream of data that has a consistent length, we use padding to make the shorter reviews as long as the others by filling the shortfall by zeros, and trim the longer reviews to the same length as the short ones. However, we might lose some useful features that could cost us some accuracy points down the path if we put it too short. On the other hand, if we put it too long, our LSTM cell will have to be larger to store the possible values. Therefore we decided to plot the distribution of the number of words in reviews. As shown below:



*Fig. 7: Distribution of number of words in reviews.*

As we can see, most of the review length is around 20 to 30. We could set the maximum length to pad or to trim to about 50. But we have set the maximum length to 200 just in case of losing several useful features.

In addition, there are two hyperparameters in LSTM which are worth noticing, called LSTM dropout and recurrent dropout.



*Fig. 8: Depiction of the dropout technique (Gal & Ghahramani, 2016).*

LSTM Dropout is a probabilistic drop out layer on the inputs in each time step, as illustrate on the left diagram (arrows pointing upwards). On the other hand, recurrent drop out is something like a dropout mask that applies drop out between the hidden states throughout the recursion of the whole LSTM network, which is depicted on the right diagram(arrows pointing to the right). These two parameters are very useful as they can generalize the model and thus prevent overfitting.

Initially, we did use these two hyperparameters in our LSTM layer. However, the training time of the regular LSTM per epoch is too long. Thus in the case that our hardware equipment can support, we decided to use the fast LSTM implementation backed by cuDNN, which is called CuDNNLSTM. The CuDNNLSTM is absurdly faster than the regular LSTM. In our case, it is actually 20 to 30 times faster than regular LSTM during the training time (regular LSTM is around 5 minutes per epoch, while CuDNNLSTM only requires few seconds per epoch). Nevertheless, the tradeoff is that by using CuDNNLSTM, we are not able to use the LSTM dropout and recurrent dropout hyperparameters as both hyperparameters are still not supported by the cuDNN API.

During our training stage, we also set the EarlyStopping callback that uses validation loss to determine whether we should stop training the model. This callback will halt the training process when validation loss continuously drops over several epochs, and save the optimal weights as a checkpoint and rewrite the weights if loss decreases. With this callback, we can let the model train for many epochs while avoid overfitting. Configurations of other hyperparameters are shown below:

|  |  |  |
| --- | --- | --- |
| Hyperparameters | Description | Configuration |
| Embedding Dimension | Dimensions of the word vectors | 300 |
| Max Features | Unique words to use (i.e num rows in embedding vector) | 20,000 |
| Epoch | Number of forward and backward passes through all of the training examples | 20 |
| Batch Size | Number of training examples in each pass | 128 |
| LSTM Dimension | Dimensionality of the LSTM output space. | 50 |
| CNN Kernel Size | Length of the 1D convolution window. | 3, 4 and 5 |
| Patience | Number of epochs with no improvement after which training will be stopped. | 2 |

**3.3.4 Pre-trained Word Vectors**

A recent study has suggested that initializing word vectors with those obtained from an unsupervised language model is a recognizing way to improve performance in the absence of a large supervised training set (Collobert et al., 2011). Thus, we experimented with 3 unsupervised learning algorithms, which are Word2vec from Google(Mikolov, Sutskever, Chen, Corrado, & Dean, 2013), GloVe from Stanford (Pennington, Socher, & Manning, 2014), and FastText from Facebook (Bojanowski, Grave, Joulin, & Mikolov, 2017). Word2vec learns word vector representations by attempting to predict context words around an input word. FastText builds word vectors by summing vectors of character n-gram level. GloVe on the other hand is a model based on global word-word co-occurrence statistics (Cliche, 2017). All of these pre-trained word vectors have 300 dimensionality. Words that are not existed in these pre-trained word vectors are randomly initialized using the same mean and standard deviation of that particular word embedding.

**4. Results**

**4.1 Model comparison**

In this section we show the results of our review rating prediction using different machine learning models. For LSTM-CNN model, we also compare the performance of our model with two configurations (true and false) on the trainable parameter in Embedding layer, which basically tells our model to or not to fine tune the weights associated with each word in the pre-trained word embedding.

As mentioned in part 3.1.3 (Convolutional neural networks), we also compare against LSTM-CNN with multichannel architecture (having two ‘channels’ of word vectors, one that is kept static throughout training and one that is fine-tuned via backpropagation).

The results are summarized in the Table 1. This table is not meant to be an exhaustive list of all the experiments performed, but it does illustrate the relative performances of the most important variations on the models explored here (comparison between different pre-trained word vectors, retrain or not to retrain the selected word vector, and comparison between single channel and multichannel architecture).

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Model | Accuracy | Micro average F1 | Macro average F1 | Weighted average  F1 |
| SVM (TFIDF) | 0.4668 | 0.47 | 0.37 | 0.44 |
| LSTM-CNN (Word2vec, Trainable) | 0.4869 | 0.49 | 0.42 | 0.48 |
| LSTM-CNN (GloVe, Trainable) | **0.5021** | **0.50** | **0.44** | **0.50** |
| LSTM-CNN (FastText, Trainable) | 0.4962 | 0.50 | 0.43 | 0.49 |
| LSTM-CNN (Word2vec, Not trainable) | 0.4765 | 0.48 | 0.37 | 0.45 |
| LSTM-CNN (GloVe, Not Trainable) | 0.4862 | 0.49 | 0.40 | 0.47 |
| LSTM-CNN (FastText, Not Trainable) | 0.4932 | 0.49 | 0.41 | 0.48 |

*Table 1. Performance metrics of models with different configurations*

It seems that SVM has the lowest performance metrics among other models with different configurations. We also observed that setting embeddings to not trainable actually delivered a slightly worse result. Seems like in common practice, using pre-trained embeddings as initialization and adjust their weights during training will boost the performance of model.

In our case, LSTM-CNN model with GloVe set to trainable has achieved 50% in accuracy, micro-average of F1 and weighted-average of F1, as well as 44% in macro-average of F1. Since this configuration has the highest performance metrics, we will be using this setting for evaluating the performance of single channel architecture vs. multichannel architecture LSTM-CNN. The results are summarized in the Table 2.

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Model | Accuracy | Micro avg F1 | Macro avg F1 | Weighted avg F1 |
| LSTM-CNN (Single channel) | **0.5021** | **0.50** | **0.44** | **0.50** |
| LSTM-CNN (Multichannel, trainable GloVe and untrainable GloVe) | 0.4857 | 0.49 | 0.42 | 0.48 |

*Table 2. Performance metrics of LSTM-CNN with different architecture*

These results seem to indicate that Kim’s (2014) experiment result was correct, which saying that the result of single channel and multichannel models are mixed. Initially, we expect multichannel architecture model would not overfit by ensuring that the learned embeddings do not deviate too far from the initial pre-trained values and thus can achieve better result.

However, in our experiment, multichannel LSTM-CNN model did not meet the expectation. We can see that single channel model outperforms multichannel model in terms of performance metrics. This single channel architecture could be upgraded via adding an extra dimensions which are allowed to be fine-tuned during training (Kim, 2014).

**4.2 (X.X use font 12, times new roman, bold, blue colour like this; X.X.X use font 11 and the rest are same)**

References

Baeza-Yates, R., & Ribeiro-Neto, B. (1999). *Modern information retrieval* (Vol. 463): ACM press New York.

Bojanowski, P., Grave, E., Joulin, A., & Mikolov, T. (2017). Enriching word vectors with subword information. *Transactions of the Association for Computational Linguistics, 5*, 135-146.

Chang, C.-C., & Lin, C.-J. (2011). LIBSVM: A library for support vector machines. *ACM transactions on intelligent systems and technology (TIST), 2*(3), 27.

Chen, P.-Y., Wu, S.-y., & Yoon, J. (2004). The impact of online recommendations and consumer feedback on sales. *ICIS 2004 Proceedings*, 58.

Cliche, M. (2017). BB\_twtr at SemEval-2017 task 4: twitter sentiment analysis with CNNs and LSTMs. *arXiv preprint arXiv:1704.06125*.

Collobert, R., Weston, J., Bottou, L., Karlen, M., Kavukcuoglu, K., & Kuksa, P. (2011). Natural language processing (almost) from scratch. *Journal of machine learning research, 12*(Aug), 2493-2537.

Fan, J., Heckman, N. E., & Wand, M. P. (1995). Local polynomial kernel regression for generalized linear models and quasi-likelihood functions. *Journal of the American Statistical Association, 90*(429), 141-150.

Fan, R.-E., Chang, K.-W., Hsieh, C.-J., Wang, X.-R., & Lin, C.-J. (2008). LIBLINEAR: A library for large linear classification. *Journal of Machine Learning Research, 9*(Aug), 1871-1874.

Gal, Y., & Ghahramani, Z. (2016). *A theoretically grounded application of dropout in recurrent neural networks.* Paper presented at the Advances in neural information processing systems.

Hochreiter, S. (1998). The vanishing gradient problem during learning recurrent neural nets and problem solutions. *International Journal of Uncertainty, Fuzziness and Knowledge-Based Systems, 6*(02), 107-116.

Joachims, T. (1998). *Text categorization with support vector machines: Learning with many relevant features.* Paper presented at the European conference on machine learning.

Kim, Y. (2014). Convolutional neural networks for sentence classification. *arXiv preprint arXiv:1408.5882*.

Kivinen, J., Warmuth, M. K., & Auer, P. (1997). The Perceptron algorithm versus Winnow: linear versus logarithmic mistake bounds when few input variables are relevant. *Artificial Intelligence, 97*(1-2), 325-343.

Mikolov, T., Sutskever, I., Chen, K., Corrado, G. S., & Dean, J. (2013). *Distributed representations of words and phrases and their compositionality.* Paper presented at the Advances in neural information processing systems.

Pedregosa, F., Varoquaux, G., Gramfort, A., Michel, V., Thirion, B., Grisel, O., . . . Dubourg, V. (2011). Scikit-learn: Machine Learning in Python Journal of Machine Learning Research.

Pennington, J., Socher, R., & Manning, C. (2014). *Glove: Global vectors for word representation.* Paper presented at the Proceedings of the 2014 conference on empirical methods in natural language processing (EMNLP).

Ramos, J. (2003). *Using tf-idf to determine word relevance in document queries.* Paper presented at the Proceedings of the first instructional conference on machine learning.

Smith, C. (2016). By the numbers: 60+ amazing reddit statistics. In.

Soleimany, A. (2019). *Deep Sequence Modeling*.

Sosa, P. M. (2017). Twitter Sentiment Analysis using combined LSTM-CNN Models.

Vapnik, V. (2013). *The nature of statistical learning theory*: Springer science & business media.

Yang, Y., & Pedersen, J. O. (1997). *A comparative study on feature selection in text categorization.* Paper presented at the Icml.

Zaremba, W., Sutskever, I., & Vinyals, O. (2014). Recurrent neural network regularization. *arXiv preprint arXiv:1409.2329*.

Zhang, Y., & Wallace, B. (2015). A sensitivity analysis of (and practitioners' guide to) convolutional neural networks for sentence classification. *arXiv preprint arXiv:1510.03820*.