Leveraging NLP For Automated Essay Scoring

Andrew Tjen
939206
The University of Melbourne
atjen@student.unimelb.edu.au

Thomas Bowes
99377
The University of Melbourne
tbowes@student.unimelb.edu.au

Kok Tong Low
683812
The University of Melbourne
kokl@student.unimelb.edu.au

Patrick Liu
947730
The University of Melbourne
zpliu@student.unimelb.edu.au

Abstract

Essays are an important metric for academic achievement, but they are expensive and time consuming for teachers to grade. An accurate, automated scoring system can provide a fast and effective solution to mass grade essays. Teachers could use the time and resource saved to focus on more important tasks. Our project is focused on using various artificial intelligence and machine learning techniques to automate essay grading. Our best final average kappa score was 0.744.

1 Introduction

When teachers or education authorities are creating assessments for exams, rather than consider the best way to assess a student's knowledge, they must ensure the tests are as cost and time effective to administer as possible. This lead assessment makers to preference multiple choice questions over essay questions as marking of multiple choice questions can be automated, where essays require teachers to mark them. This ruins the integrity of assessments as they do not adequately reflect the ability of students taking them (Sandra, 2020). Moreover, this also negatively affect students, who no longer get the opportunity to practice organising thoughts and deeply reflecting on their respective course content (Sandra, 2020).

This paper will attempt to address this problem by using machine learning and data science techniques to predict essay scores. The data is sourced from the Hewlett Foundation (Hewlett Foundation, 2012), which provides essays from students over multiple year levels that are labelled with marks generated by teachers. Considering this, success is defined as producing an algorithm that produce predictions that are reasonably close to human markers.

2 Dataset

The dataset contains over 20,000 entries, including the essay, the total marks (discrete scale), and each teacher's individual rating. There was also some censorship (to anonymize sensitive data) and word flags for words that were not completely legible (where a guess was made) in the essay. Each essay comes from one of the eight essay sets and has a different scoring metric, they range from an average length of 150 to 550 words. All of the essays were produced by students in grade seven to grade ten. The scores of the essays were done by hand, and each essay set has its own unique characteristics.

Below is a typical row in the dataset. As we can see, the instance below comes from essay_set 1, and there are various rating domains. The total_score is a feature we engineered ourselves, which we will cover more comprehensively in the feature engineering section.

essay_id	essay_set	essay	rater1_domain1	rater2_domain1	rater3_domain1	domain1_score	rater1_domain2	rater2_domain2	domain2_score	total_score
1	1	Dear local newspaper, I think effects computer...	4	4	0.0	8	0.0	0.0	0.0	8.0

Figure 1: Sample row from the dataset.

3 Evaluation metric

The Kaggle competition used a quadratic weighted kappa error metric, and thus after some research, we chose the same metric for our models. The quadratic weighted kappa error metric is a modified version of the Cohen's kappa coefficient. The Cohen's kappa coefficient is used to measure inter-rater reliability for categorical variables. To understand its advantages, over a simple percentage agreement metric, it is necessary to understand its formula:

$$\kappa \equiv \frac{p_o - p_e}{1 - p_e}$$

p_o is the relative observed agreement among raters, the same as accuracy. While p_e is the probability of chance agreement. For example, if marker A had a 50% chance of saying "Yes", while marker B had a 40% chance of saying "Yes", the probability of them both saying "Yes" is $0.5 \times 0.4 = 0.2$. The probabilities are point estimated using the observed sample.

Now, quadratic weighted kappa is very similar to Cohen's kappa coefficient. The quadratic weighted kappa is calculated as follows:

First, an N-by-N histogram matrix O is constructed, such that $O_{i,j}$ corresponds to the number of essays that have a rating of i (actual) and received a predicted rating j. An N-by-N matrix of weights, w, is calculated based on the difference between actual and predicted rating scores. An N-by-N histogram matrix of expected ratings, E, is calculated, assuming that there is no correlation between rating scores. This is calculated as the outer product between the actual rating's histogram vector of ratings and the predicted rating's histogram vector of ratings, normalized such that E and O have the same sum. $WeightedKappa = 1 - sum(w_{i,j}O_{i,j})/(w_{i,j}E_{i,j})$

The quadratic weighted kappa accounts for random agreements, this is contained in p_e. Thus, a zero-r baseline model gives us a kappa score of 0, and anything below 0, performs worse than randomly guessing. Therefore we use this metric as it clearly distinguishes between random guessing and predictions.

4 Pre-processing

4.1 Data Cleaning

Since the dataset comes from a Kaggle competition, the dataset was of high quality. There was very little data cleaning to be done. All we had to do was remove words labelled as illegible (where guesses were made) and standardised the wording for censored words (ie. @place1 -> @place). We do this so that we do not have additional words in our vocabularies that have no meaning, like "hello" and "helo" as well as @place1 and @place2, these needlessly add to the dimensionality of our tokenisation algorithms and may lead to over fitting in our models.

4.2 Preprocessing and Feature Engineering

The essay sets had different scoring systems, and different number of rating domains. Thus, we had to find a total essay score that would be used as our outcome variable. We took averages across the markers and domains and use this to form our total_score variable.

We also engineered three features.

- Word_count: A simple count for the number of words in the essay.
- Mistakes: The number of grammatical and spelling errors in the essays, comes from the python language_tool package (shivam5992, 2020)
- Reading_ease: Returns a Flesch Reading Ease Score, comes from the text_stat package. Basically a score for the readability of the essay. (Morris, 2020)

4.3 Doc2Vec

One of the word tokenisation technique that will be used is Doc2Vec, which vectorises a set of words into a fixed size vector (regardless of size). Doc2Vec is an extension of Word2Vec, so to understand Doc2Vec, Word2Vec must be understood. Word2Vec works by producing fixed length vectors corresponding to each word in a vocabulary (where vocabulary is defined by the number of unique words that the algorithm is trained on). It is founded on the idea that words in similar contexts have similar meanings. It is created by using a neural network to predict a word, from surrounding words. Once the neural network is trained, the weights associated with each word becomes the word embedding (word vector) for the given word. With these word embeddings we can now perform mathematical operations on these words, such as king – man + woman = queen (Shperber, 2017).

As mentioned previously, Doc2Vec is built on Word2Vec. It is trained exactly as before, except with a document id included in the input for every word in the training set. Once the neural network is trained, the weights associated with the document id becomes the document embedding (document vector) (Le, 2014). Much like before this allows us to perform mathematical operations on documents, for instance, checking similarity. This vector will then be passed as attributes in the prediction models. The model finds any links between certain elements of the vector and essay scores. Doc2Vec is far more effective than naïve alternatives like bag of words, as it takes into account the relative ordering of words and their contexts.

5 Models

In this report we treat our essay scoring problem as a regression problem as essays are marked on an integer scale between 1 and n. We expect our models to minimise the error of the predicted values and the actual values. We define this difference using Mean Squared Error (MSE) and hence, use this as our loss function.

Mean Square error is defined as:

$$MSE = \frac{1}{n}\sum_{i}^{n}(\hat{y}_i - y_i)^2$$

Where $\hat{y}_i$ is the predicted value, y_i is the actual value and n is the size of the dataset.

5.1 Feed Forward Neural Networks - FFNN

We consider Feed Forward architecture as one of our models. Mainly due to our goal of wanting to explore neural networks for this task and it being one of the simplest neural networks available. Feed Forward Neural Networks follows a unidirectional, acyclic architecture where each node is fully connected to every node in the next hidden layer (Fine, 1999, pp. 52-56). We hypothesize that this model will work well with our doc2vec vector representation due to the weight calculation it performs on each neuron for each feature produced. We will eventually fit a unidirectional two-layered neural network for this task, which will be explained in the later section.

5.2 Long Short Term Memory

The next Neural Network model we considered was LSTM. It follows the concept of a Recurrent neural networks where it takes into account the memory of previous neuron results. However, unlike Recurrent Neural Network which only takes into account memories of the one previous neuron,

LSTM extends this idea by taking into account both long and short term memory (memorise and updates additional previous results from neurons). This is possible as instead of having the usual single activation function on Recurrent Neural Networks, it now includes 4 extra interacting layers of functions.

- Forget Gate: Measures how much information to be discarded.
- Input Gate: Measures how much information to be updated
- Tanh Layer: Creates a candidate of new vectors to be added
- Output Layer: Measures what is needed to be output based on the cell's current state

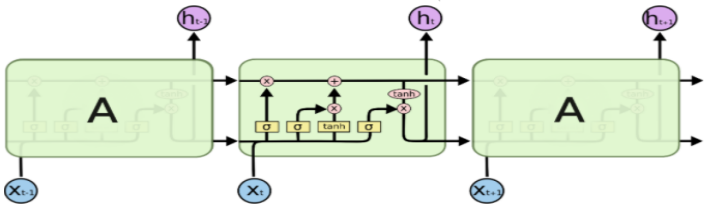

Figure 2: Single Cell LSTM functions (Sharma, 2017).

From the figure, we can see the yellow box represents the layers mentioned while the pink circles represents the mathematical function that updates the vectors. The arrow structure represents the neuron connections as data from the previous cell was calculated and added to the next, which represents the long-term memory component. While the function inside represents the short term concept as it updates the vector for each neuron. The blue circle represents the input representation and the purple box represents the output weight representation. The reasoning behind the importance of the memory concept can be argued as a word that tends to have context meaning on the sentence, which can be crucial in the NLP process. We will fit a two-layered unidirectional model with tuned parameters on the number of neurons and drop out rate, which is the regularisation parameter.

5.3 Bidirectional Encoder Representation from Transformer - BERT

Bidirectional Encoder Representations from Transformer (BERT) is a language representation model used for Natural Language Processing (NLP) tasks that is developed and pre-trained by Google. BERT is pre-trained on a large textual dataset - the entirety of English Wikipedia and Brown Corpus. The goal of BERT is to better understand human language with the use of Transfer Learning, which allows the model to perform particularly well on small datasets, as the model was already mostly trained.

BERT uses Transformer architecture, which normally includes an 'Encoder' and a 'Decoder'. However, BERT only uses the Encoder portion of a Transformer architecture as the key goal is to create a language model. The encoder takes in an input and runs it through a Self-Attention Process and then its outputs run through a Feed-Forward Neural Network. The Self-Attention process distinguishes which words are most important and weighs them accordingly. This means that the Feed-Forward Neural Network can focus on the words that are most important.

As the name suggests BERT encodes bidirectionally by reading the input as a whole, unlike many NLP methods that encode or take inputs sequentially in one direction (left to right or right to left). The advantage of this method allows the model to better understand bidirectional context, for example the following sentences "I went to the bank of the river" and "I went to deposit money into my bank account ", in unidirectional models such as LSTM (Long Short Term Memory) or RNN (Recurrent Neural Network) the word "bank" is encoded the same in both sentences, however this is not the case as the first sentence refers to a river bank not the financial institution. However, BERT reads the input as a whole and with the attention mechanism as mentioned above, it allows the model to focus on words that are important, such as "river" and "account". From the example above, the model can understand that the "bank" in the first sentence is different from the second sentence and encodes it respectively.

There are a few reasons BERT was considered for this report, the first reason is due to the historical performance of BERT, as it has been outperforming alternative NLP methods such as LSTM, RNN and

GPT (Generative Pre-trained Transformer) in GLUE (General Language Understanding Evaluation) as seen in (GLUE, 2020), which is used as a benchmark to compare performance of models on NLP tasks. Secondly, the bidirectional context capturing aspect of BERT may capture more information about essays than our alternate models.

5.4 Support Vector Regression-SVR

SVR uses the same concept as Support Vector Machines, but for regression, the main goal is to develop a hyperplane with maximised decision boundary. SVR also has different kernel functions - linear, polynomial, and radial based function (rbf). The kernel functions are responsible for taking input data and transforming them into a form where it is usable by the SVR. This is necessary because some data are not linearly separable, hence the need for the different kernel functions to separate data in different dimensions. With the data on the same scale, no assumption has to be made for SVR as it is quite forgivable to input data.

With the use of rbf kernel the equation is (Chung et al., 2003):

$$K(x, x') = exp(-\frac{\|x - x'\|^2}{2\sigma^2})$$

This function allows the model to calculate the similarities between data points, the kernel value decreases as the difference in data points increases. The model will be able to predict the scores based on the rbf kernel.

6 Experiments and Results

We designed our experiment to fit models and predict scores per essay set category. The reason was to account for different scoring style in each essay sets. We evaluated the model using five fold cross validation with a train test split of 20%. Finally, rounding the predicted score to the nearest integer and calculating the Quadratic weighted Kappa score per essay set category.

6.1 Baseline: Logistic regression

We used three variables for logistic regression, word count, mistakes, and reading ease. We used a simple selection of metadata attributes for this model so that we could see the benefits that NLP provided us. The output was a percentage score, which was then rounded to the nearest essay score, depending on the specific essay set. For example, a logistic prediction of 33% was rounded to a 3, for when the essay score was ranged from 0-10. The results are below:

Essay Set	1	2	3	4	5	6	7	8	Average
Kappa Score	0.303	0.279	0.475	0.397	0.518	0.34	0.0416	0.0299	0.297

Table 1 - Kappa scores for each essays set on the Logistic Regression Baseline.

Essay sets 7 and 8 had a scoring range of 0-30 and 0-60 respectively, which explains the poor kappa score. The final average kappa score was ≈ 0.29. We will use this as our baseline model.

6.2 Neural Networks

We used the Keras deep learning package to build our model (Chollet et al., 2015). A doc2vec representation of the data set is chosen to tokenise text for both neural networks, we run a doc2vec algorithm for each cleaned essay set. We choose a vector size of 50 and window size of 2 in this project. The lack of time and computing power does not allow us to tune the vector or window size parameters to observe how it affects model performance. We then added the reprocessed features mentioned earlier to the data set and train the model using root mean squared error as our loss function. Finally early stopping was implemented so that when performance plateau the neural network stopped training further.

6.2.1 Feed Forward Neural Network

For the feed-forward neural network, we designed a two-layered neural network with Rectified Linear Unit (Relu) as the activation function. Each layer's number of neurons was tuned to 3 values,

$[65, 80, 95]$, please refer to the graph below. The model then converges into one neuron and outputs the predicted value. The model was built on a rmsprop optimizer with early stopping, and was fitted with a validation split of 0.2, epochs of 1000, and a batch size of 32.

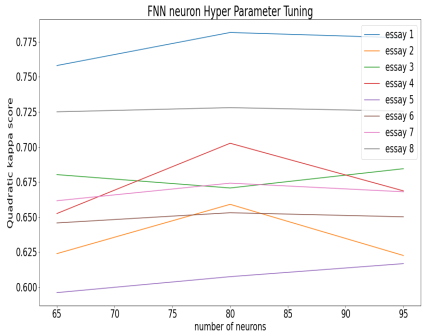

Figure 3: Shows the tuning of the hyper-parameter number of neurons over 8 essay sets with kappa score as performance metric.

From the tuning graph we can observe some very weak correlation between the neuron sizes, possibly due to the small range of tuning. We then outputted the best score for each essay set.

Essay Set	1	2	3	4	5	6	7	8	Average
Neurons	80	80	95	80	95	80	80	80	
Kappa Score	0.782	0.659	0.684	0.703	0.617	0.653	0.674	0.728	0.687

Table 2 - Best FFNN results (with hyper-parameters) for each essay set and average score over all sets.

6.2.2 Long Short Term Memory

We also designed a two-layer LSTM, each layer has the same number of neurons which converges into a Rectified Linear Unit (Relu) activation function. This model was built on top of the rmsprop optimizer with early stopping. We used an epoch of 1000, validation split of 0.2, batch size of 32, and added early stopping monitoring on loss in performance. We conducted hyper-parameter tuning on the two parameters of the LSTM layer, the dropout rates, and the neurons. Drop out rates represents the level of regularisation, while neurons represent the number of nodes in the model. We implemented a grid search parameter tuning by experimenting with 3 neuron values of $[50, 100, 200]$ and 3 drop out rates of $[0.2, 0.4, 0.8]$. If we had less time and hardware restrictions, a more extensive hyper-parameter tuning could have been completed.

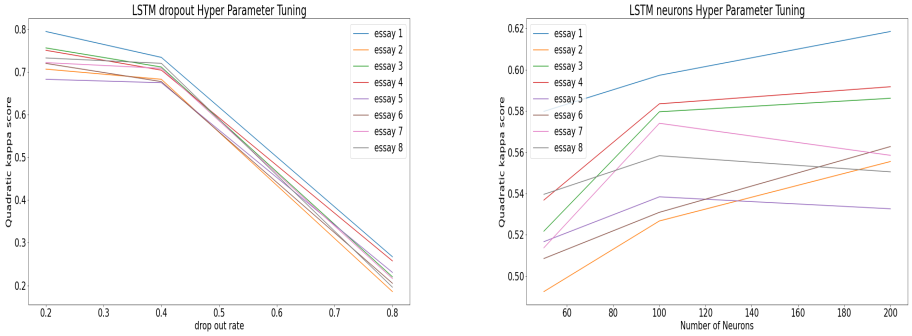

Figure 4: Shows the tuning of the hyper-parameters, drop out rate and number of neuron per layer (left to right) with kappa score as performance metric.

From the first graph in figure 4, we were able to observe that lowering the drop out rate gave us better performance. The result was consistent for every essay set. In the second graph in figure 4,

we chose the best number of neurons for each essay set. Then, we took the best possible parameter combination of each essay set.

Essay Set	1	2	3	4	5	6	7	8	Average
Neurons/Dropout	200/0.2	200/0.2	200/0.2	200/0.2	100/0.2	200/0.2	100/0.2	100/0.2	
Kappa Score	0.796	0.704	0.748	0.729	0.686	0.725	0.732	0.730	0.720

Table 3 - Best LSTM results (with hyper-paramaters) for each essay set and average score over all sets.

The model performed best for essay set one. The model successfully tackled the more complex score rubric of essay sets 7 and 8 which the baseline model failed to do so.

6.3 BERT-SVR

For the BERT implementation we had to preprocess the essays by using a pre-trained BERT model, it takes in 2 main input parameters; input_ids (BERT Tokenized ids), and attention_masks (a value 1 was assigned for available input_ids and a 0 value was assigned for padded values). The BERT model has multiple conditions for inputs. Firstly, it takes in specific tokenized values created by a huggingface library, BertTokenizer (huggingface, 2020), where a classifier token known as CLS was added at the start of each document, and a separator token known as SEP which was added at the end. Secondly, BERT model takes in a maximum of 512 tokens including the 2 special tokens mentioned above. Thirdly, inputs have to be of the same length, in this dataset, documents had varying length but majority of documents had token length of 500 - 600, hence the maximum 512 tokens were used in the BERT model. Documents that had fewer than 512 tokens had to be padded by adding 0 to the end and documents that had more than 512 tokens had to be truncated.

With the documents in a BERT recognised format, the data was fitted into the pre-trained BERT base model. An output object with parameters: Layer number = 13, batch number = 4,token number = 512, and feature number = 768 was returned. There were many ways to extract the features from the model output, in this report only the final layer's first token (CLS) was extracted, with each token corresponding to its respective document. This is because CLS contains all the relevant information required for classification. Each CLS token (document token) has 768 elements and was fed into a Support Vector Regression (SVR). Hyper-parameter tuning was performed on the model by performing a grid search on the C parameter, selecting the one that produced the highest 5-fold cross validated kappa score.

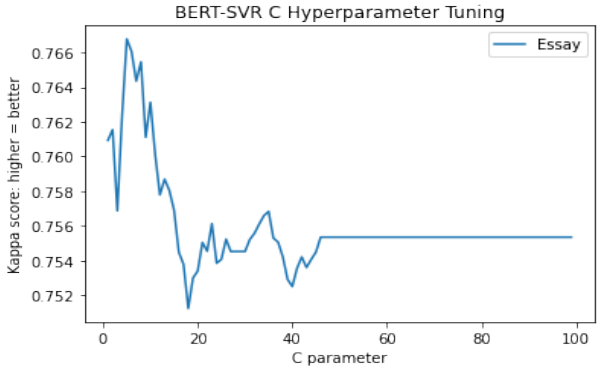

Figure 5: Shows the tuning of BERT-SVR reguralisation parameter with kappa score as performance metric.

Essay Set	1	2	3	4	5	6	7	8	Average
C Parameter	26	5	5	5	1	6	26	26	
Kappa Score	0.757	0.692	0.627	0.777	0.763	0.770	0.806	0.760	0.744

Table 4 - BERT-SVR results for each essay set and the average score over all essay sets.

As seen from Table 4, the results from using the BERT-SVR model performed well, with each essay sets' Kappa scores relatively close to each other, which indicates a stable performance, with essay set 7 having the highest Kappa score.

7 Discussion

From the table results, our models showed a competitive performance when compared to the Kaggle leader board, with our best model being BERT-SVR. The bidirectional learning model of BERT as mentioned above may indicate that the model captured a more accurate contextual meaning behind words in the essays. Thus, explaining the better performance. Next, due to the degree of complexity behind the neural network structure, a regression task was not the optimal choice. The reason for this is, if not tuned properly, they are more prone to overfitting.

Results of essays set 7 and 8 were more interesting than others as it had a more complex scoring style. With BERT-SVR coming on top, we can conclude that BERT-SVR regularized the data better due to the kernel and soft margins. However, we can observe the positive impact of regularisation as LSTM's model that includes drop out rate as a regularization parameter, performed better than Feed Forward Neural Network that has none.

From the result, we can argue that BERT may be a stronger word tokeniser than doc2vec. This was expected due to the historic high performances. However, more experimentation are needed to solidify that BERT performs better in an essay scoring scenario. One of which is running BERT on Neural Network models such as LSTM and Feed Forward. A hyper-parameter search can also be done on Doc2vec representation, exploring larger vector sizes with different epoch and window sizes.

In this project the initial idea of purely using BERT was attempted, since our task was considered a regression task, we needed to add a regression layer on top of the final layer of BERT. However, given that BERT was rather complex, and that documentation of using BERT for regression tasks was sparse, this led to the code constantly running into errors, especially memory errors, which ultimately led the team to decide to drop the model. Next, BertForSequenceClassification was used, however due to hardware limitations, hyper-paramaters could not be tuned to the recommended parameters as mentioned by (Devlin et al., 2018). This led to an unsatisfactory performance which resulted in the removal of the model. The BERT-SVR model could be inaccurate if the essays lengths are over 510 tokens, which is the maximum size allowed by BERT. This might result in the model not being able to capture key information required for a fair score.

Although BERT-SVR was our best performing model, many other methods were not tested. For instance, ensemble methods could be used to further improve the performance as seen in (Xu et al., n.d.), where many of the BERT models performed better when ensemble methods were used.

Another thing we attempted was to create a general essay marker that could accept any essay. However, its performance was sub-par. This makes sense as our training data comes from a diverse set of assessments with different marking requirements, as well as different vocabularies, making the data far noisier and encouraging the algorithm to over fit.

A limitation of our model that may prevent it from being applicable is the large training data required. This means that from our observations, human markers are required for at least the first 1000 essays on a given subject, to train the algorithm before the remaining papers can be automated. This limits its potential use cases to only assessments taken by many students.

8 Conclusion

In conclusion, we found that BERT-SVR displayed the best performance, with LSTM performing the second best. Teachers could use the models in this report to mass grade essays. For extreme values produced by the models, teachers can re-grade these essays by hand. On the contrary, the algorithm could be cheated by students, for example students might purposely add overly complex words, thus receiving a higher automated grading score. The limitation of the algorithm would be for essays in which the topic is chosen by the writers themselves, for which creativity, context, and ideas would play a larger role. This report clarifies the importance of choosing word vectorization techniques such as BERT and Doc2vec, as well as the use of neural networks such as LSTM and FFNN in regards to natural language processing.

References

[1] Abadi, Martx27;in, Barham, P., Chen, J., Chen, Z., Davis, A., Dean, J., ... others. (2016). Tensorflow: A system for large-scale machine learning. In 12th $USENIX$ Symposium on Operating Systems Design and Implementation ($OSDI$ 16) (pp. 265–283).

[2] Albert, Y. (2020, June 19). BERT - Tokenization and Encoding [Blog Post]. Retrieved from https://albertauyeung.github.io/2020/06/19/bert-tokenization.html

[3] Alex, M. (2009, June 4). How to make a flat list out of list of lists? [Blog Comment]. Retrieved from https://stackoverflow.com/a/952952

[4] Anon. (2016). Dear, Students I just don't have time to mark your essay properly. Retrieved from https://www.theguardian.com/higher-education-network/2016/may/20/dear-student-i-just-dont-have-time-to-mark-your-essay-properly

[5] BatmantoshReturns. (2020) Has anyone used Transformer models to predict continious variables? [Blog Post]. Retrieved from https://www.reddit.com/r/LanguageTechnology/comments/f2o3ty/

[6] Fine Tuning Pre-trained BERT (2018). Retrieved from https://gluon-nlp.mxnet.io/examples/sentence_embedding/bert.html

[7] Dhami, D. (2020). Understanding BERT — Word Embeddings. Retrieved from https://medium.com/@dhartidhami/understanding-bert-word-embeddings-7dc4d2ea54ca

[8] Chollet, F., others. (2015). Keras. GitHub. Retrieved from https://github.com/fchollet/keras

[9] Chung, K. M., Kao, W. C., Sun, C. L., Wang, L. L., Lin, C. J. (2003). Radius margin bounds for support vector machines with the RBF kernel. Neural computation, 15(11), 2643-2681.

[10] Devlin, J., Chang, M. W., Lee, K., Toutanova, K. (2018). Bert: Pre-training of deep bidirectional transformers for language understanding. arXiv preprint arXiv:1810.04805.

[11] Dima, S. (2019, June 6). BERT to the rescue! [Blog Post]. Retrieved from https://towardsdatascience.com/bert-to-the-rescue-17671379687f

[12] Ezen-Can, A. (2020). A Comparison of LSTM and BERT for Small Corpus. arXiv preprint arXiv:2009.05451.

[13] Fine, T. L. (1999). Feedforward Neural Network Methodology (Information Science and Statistics) (1999th ed.). New York, United States Of America: Springer.

[14] GLUE. (2020). Glue benchmark leaderboard. Retrieved from https://gluebenchmark.com/leaderboard

[15] Hewlett Foundation. (2012). Develop an automated scoring algorithm for student-written essays. Retrieved from https://www.kaggle.com/c/asap-aes

[16] Huggingface. (2020). Transformers. Retrieved from https://huggingface.co/transformers/model_doc/bert.html

[17] Hunter, J. D. (2007). Matplotlib: A 2D graphics environment. Computing in Science amp; Engineering, 9(3), 90–95.

[18] Jack Morris, Python package language-tool, open-source grammar tool. Retrieved from https://pypi.org/project/language-tool-python/

[19] Jay, A. (2018). A Visual Guide to Using BERT for the First Time [Blog Post]. Retrieved from http://jalammar.github.io/a-visual-guide-to-using-bert-for-the-first-time/

[20] Jonathan, H. (2019, November 5) NLP — BERT Transformer [Blog Post]. Retrieved from https://jonathan-hui.medium.com/nlp-bert-transformer-7f0ac397f524

[21] Le, Q., Mikolov, T. (2014). Distributed representations of sentences and documents. In International conference on machine learning (pp. 1188-1196).

[22] Marc, S. (2012, June 23). What is the influence of C in SVMs with linear kernel? [Blog Comment]. Retrieved from https://stats.stackexchange.com/questions/31066/what-is-the-influence-of-c-in-svms-with-linear-kernel

[23] McKinney, W., others. (2010). Data structures for statistical computing in python. In Proceedings of the 9th Python in Science Conference (Vol. 445, pp. 51–56).

[24] Michel, K. (2019, September 15). BERT for dummies — Step by Step Tutorial [Blog Post]. Retrieved from https://towardsdatascience.com/bert-for-dummies-step-by-step-tutorial-fb90890ffe03

[25] Nadeem, F., Nguyen, H., Liu, Y., Ostendorf, M. (2019). Automated Essay Scoring with Discourse-Aware Neural Models. In Proceedings of the Fourteenth Workshop on Innovative Use of NLP for Building Educational Applications (pp. 484-493).

[26] Nigam, V. (2019). Natural Language Processing: From Basics to using RNN and LSTM . Retrieved from https://towardsdatascience.com/natural-language-processing-from-basics-to-using-rnn-and-lstm-ef6779e4ae66

[27] Nitin, P. (2020, June 8). Text Classification using BERT, sklearn and Pytorch [Blog Post]. Retrieved from https://medium.com/naukri-engineering/text-classification-using-bert-sklearn-and-pytorch-7665433b56c7

[28] Oliphant, T. E. (2006). A guide to NumPy (Vol. 1). Trelgol Publishing USA.

[29] Paszke, A., Gross, S., Massa, F., Lerer, A., Bradbury, J., Chanan, G., ... Chintala, S. (2019). PyTorch: An Imperative Style, High-Performance Deep Learning Library. In H. Wallach, H. Larochelle, A. Beygelzimer, F. d extquotesingle Alchx27;e-Buc, E. Fox, R. Garnett (Eds.), Advances in Neural Information Processing Systems 32 (pp. 8024–8035). Curran Associates, Inc. Retrieved from http://papers.neurips.cc/paper/9015-pytorch-an-imperative-style-high-performance-deep-learning-library.pdf

[30] Pedregosa, F., Varoquaux, Ga"el, Gramfort, A., Michel, V., Thirion, B., Grisel, O., ... others. (2011). Scikit-learn: Machine learning in Python. Journal of Machine Learning Research, 12(Oct), 2825–2830.

[31] Python package textstat, library to calculate statistics from text Retrieved from https://github.com/shivam5992/textstat

[32] Rehurek, R., Sojka, P. (2011). Gensim–python framework for vector space modelling. NLP Centre, Faculty of Informatics, Masaryk University, Brno, Czech Republic, 3(2).

[33] Sandra, W. (2020, June 13). Difference Between An Essay vs Multiple Choice Tests. Retrieved from https://www.iwriteessays.com/essays/difference-between-an-essay-vs-multiple-choice-tests

[34] Senguttuvan, V. (2019). What are the advantages of LSTM? . Retrieved from https://www.quora.com/What-are-the-advantages-of-LSTM-in-general: :text=LSTM

[35] Sharma, S. (2017). Activation Functions in Neural Networks. Retrieved from http://colah.github.io/posts/2015-08-Understanding-LSTMs/

[36] Shperber, S. (2017). A gentle introduction to Doc2Vec. Retrieved from https://medium.com/wisio/a-gentle-introduction-to-doc2vec-db3e8c0cce5e .

[37] Vishwanathan, S. V. M., Murty, M. N. (2002). SSVM: a simple SVM algorithm. In Proceedings of the 2002 International Joint Conference on Neural Networks. IJCNN'02 (Cat. No. 02CH37290) (Vol. 3, pp. 2393-2398). IEEE.

[38] Xu, C., Barth, S., Solis, Z. (n.d.) Applying Ensembling Methods to BERT to Boost Model Performance.

[39] Yashu, S. (2019, June 12). BERT Explained – A list of Frequently Asked Questions [Blog Post]. Retrieved from https://yashuseth.blog/2019/06/12/bert-explained-faqs-understand-bert-working/

[40] Zaki, M. (2019, September 25). Demystifying BERT: A Comprehensive Guide to the Groundbreaking NLP Framework [Blog Post]. Retrieved from https://www.analyticsvidhya.com/blog/2019/09/demystifying-bert-groundbreaking-nlp-framework/