The Impact of Pre-processing, Feature Design and Model Selection on Blog Posts Classification Performance

Thach Jean-Pierre **Wong Leo**
Department of Computer Science and Operations Research, University of Montreal

Abstract

Data pre-processing, feature engineering and model selection are essential components in text classification problems. The purpose of this paper is to explore the various effects of these elements on the prediction accuracy for blog post classification. Numerous text normalization techniques and derived features are applied and tested on different machine learning algorithms to measure their impact. The rationale for choosing to experiment with each of the selected components is explained in the analysis. Limitations, additional approaches and further refinements are discussed at the end.

1. Introduction

Selecting the appropriate data pre-processing and normalization techniques can have an effect on text classification accuracy. This paper investigates the impact of pre-processing tasks, feature design and model selection on blog posts classification performance.

2. Feature Design

2.1 Preprocessing methods

In the tokenization step, the words in the corpus are converted into lowercase format. The underlying reason is that, in most cases, capitalization of words has no effect on the meaning.

In addition, since social media posts contain many grammatical errors and noise in the data, stream editing is applied for text normalization with a series of *sed* commands (*TP2*). The purpose of this preprocessing method is to transform real data into a normalized format and to obtain consistent input that allows the separation of concerns before processing it.

2.2 Feature selection

The following features are hypothesized to have an effect on the likelihood that a given blog is part of a specific class, and thus improve the performance of the learning algorithms:

Blog post length, word and sentence count
The varying lengths of a blog, the different word or sentence counts can help make the distinction between classes. For instance, some blogs may be more elaborate and longer than others depending on the age group.

Sentence and vocabulary complexity
The complexity of the vocabulary and the difficulty of reading can be an indicator of the author's age group. We would expect a blog post written by children to contain simpler words and sentences. We used the *textstat* library to measure the following characteristics: Flesch Reading Ease; higher scores indicates that the material is easier to read, Flesch-Kincaid Grade Level; describes the U.S. grade level of education that would be able to read the material, polysyllabic words.

Swear words
Certain blogs may be more notorious for use of profanity depending on the age group while

others would tend to be more civil. The count of swear words is used as a feature.

Part-of-speech tagging
By combining words with their context, an overview of the structure representation of blogs can be obtained.

3. Algorithms

In order to analyze the impact of the features and pre-processing steps, several classification algorithms deemed to be adequate for text classification are carefully chosen for the analysis.

3.1 Multinomial Naive Bayes

For this learning algorithm, the motive behind using the multinomial distribution for text classification comes from the representation of the words as discrete features, (i.e. the frequency). Furthermore, given that the number of classes to predict is greater than two, a binomial distribution would be inadequate for this task. Laplace smoothing is implemented to account for unseen words by tuning the hyper-parameter *alpha*.

3.2 Multinomial Logistic Regression

For this learning algorithm, an advantage of selecting this model for this task is that there exists a slight collinearity of features in addition to a regularizer penalizing high variance, compared to Naive Bayes where complete independence of features is assumed. Also, Logistic Regression tends to catch up or even outperform Naive Bayes in terms of performance as the size of the data increases, a factor that can be favorable given the relatively large dataset in this experiment. The strategy for this algorithm is to set the maximum iterations to 100 for all experiments, have class weight balanced, set the solver to *lbfgs*, and to use different values of the hyper-parameter *C* (regularizer term).

3.3 Random Forest

Following the pre-processing and feature selection phases, using a tree-based classification technique could yield better prediction accuracy by leveraging dimensionality reduction of features given the sparsity of feature representation. Given the higher capacity of this algorithm compared to Naive Bayes and Logistic Regression, the hypothesis is that it may result in better performance due to the complexity increase of decision surfaces and boundaries. The optimization strategy revolves around varying multiple hyper-parameters, notably the desired dimensionality of the output for *TruncatedSVD*, the maximum depth and also the number of estimators for the trees of the classifier.

3.4 LSTM & BiLSTM

The principal motive for selecting this model is its high complexity. Neural networks tend to perform better given a large dataset. Additionally, since the features are learned from the model, handcrafted features would be unnecessary. Our sequential model consists of an embedding layer, a spatial dropout layer, LSTM or Bidirectional LSTM layer with dropout and recurrent dropout and a dense layer. The spatial dropout layer is defined as a layer that dropout entire feature maps. In the case of standard dropout for the LSTM layer, each input and output of each RNN unit will be considered independently in the dropout. With the same principle, a fraction of the recurrent connections between units will be dropped out. The dense layer will be used in the output layer to connect the neurons and produce probabilities for each class with a softmax activation function. Finally, our model uses the Adam optimizer with categorical cross entropy loss function.

3.5 Universal Sentence Encoder

This pre-trained model from the TensorFlow Hub encodes text into high dimensional vectors. It is trained and optimized for sentences, or short paragraphs on different data sources aimed to accommodate a variety of natural language processing tasks. It takes an input of variable length and outputs a 512 dimensional vector. Our strategy and goal in using this encoder is to build a model on top of it and see how well it would perform on noisy and unclean dataset.

4. Methodology/Experimentation Protocol

For this set of experiments, lexical normalization with stream editing *sed* on the datasets, along with stemming and lemmatization are used. For the representation of the features in vector form, CountVectorizer and TfidfVectorizer from the *sklearn's* library are implemented. Furthermore, the training data is split into 90% training data and 10% validation data for the purpose of hyper-parameter tuning in the validation step. Metrics including the accuracy, precision, recall and F1 are calculated to compare the performance of the different algorithms in predicting the age group of a new blog sample.

A simple classifier that predicts the most frequent class from the class distribution is used as the baseline of our experiments.

5. Results

5.1 Impact of Preprocessing methods

5.1.1 Tokenization

As a simple experiment, we've implemented a Tokenizer that processes blogs and transforms them into tokens by filtering the *nltk* stopwords, the urls, every character outside of the alphabet. In all of the cases, this method decreased the performance of our learning algorithms. This can be explained by the loss of information and potential features contained in the urls. For instance, tokens that are related to the age group class.

5.1.2 Stemming and Lemmatization

Stemming and lemmatization could potentially increase the performance of the classifiers, but in this particular case lead to worse results. Due to the nature of the data, normalizing the corpus via stemming and lemmatization removes relevant information and distinct features. For each of the models tested, the addition of these pre-processing steps consistently lower the performance accuracy. The reasoning behind this result can be explained by the inherent complexity of our data, where the *dirtiness* of the inputs captures valuable information unique to each class.

5.1.3 Normalization

Cleaning and normalizing the data through by using regular expression through a series of sed commands yield a slightly lower performance overall for all models. By standardizing certain features believed to convey the same meaning, there is a possibility that important details are lost. Despite reducing the size of the vocabulary, certain informative features could be discarded in the process.

5.1.3 Handcrafted Features

Blog post length, word and sentence count
In *Figure 1*, we show an overview of the distribution of the length of blogs per age group, it is clear that the length decreases as it tends to younger age group. Given the variation across the groups, the effect of this feature would, in

our intuition increase performance. As we tested this feature on different classifiers, there were very slight improvements in performance depending on hyperparameters used. With that same reasoning, we expected that the sentence count would have little to no effect and that the word count wouldn't be a relevant feature to incorporate because of its similarity to the blog post length feature as the *Figure 2* and *Figure 3* outlines. As a result of our experiments, the word count feature had little to no effect on the performance and the sentence count resulted in a minor increase in performance.

Sentence and vocabulary complexity

In *Figure 4*, the Flesch Reading Ease metric illustrates some variance for between the age group 0 and the others, this may indicate that this feature would be useful. However, our tests had a consistent performance decrease on different hyperparameter values of the learning algorithms. This may be explained by the scaler used to compress the metric scores into a range; by using the default *MinMaxScaler* in which scales the values between 0 and 1, the feature becomes irrelevant and thus, result in a decrease in performance.

In *Figure 5*, the Flesch-Kincaid Grade Level illustrates a similar pattern to the blog post length and word count features. Our theory in the effect of this feature is that it wouldn't carry any extra information useful for the learning algorithm in making decisions. Effectively, there were little to no impact on the performance of the algorithms by introducing this feature.

The polysyllabic words saw a decrease of 0.0026% in accuracy in the validation phase. In *Figure 6*, the plot describes a similar pattern that may be already captured by other features with comparable variability. Another contributing factor to this loss of accuracy might be the *MinMaxScaler* score compression.

Swear words

In *Figure 7*, the fractions containing swear words as the age group gets older decreases. Because there is considerable variance between the age groups, our intuition in regards to the impact of this feature on learning algorithms would be a net positive. The experiments reveal a slight boost in performance of 0.00017% for the Multinomial Naive Bayes algorithm.

Part-of-speech tagging

Combining the *Part-of-speech* tags with the tokens as feature for the supervised learning algorithms resulted in a decrease in performance. For instance, the accuracy decreased from 69.1% to 68.8% for the Multinomial Naive Bayes algorithm in the validation phase. This can be attributed to the fact that by introducing context to tokens, it creates sparsity in the feature representation.

5.2 Multinomial Naive Bayes

The hyper-parameter alpha represents the Laplace smoothing parameter, which enables the Naive Bayes model to account for unseen words. Alpha values within the range of *0.001* and *0.7* are tested, where *alpha=0.005* gives the highest prediction accuracy. As alpha increases, the accuracy decreases, since the model underfits by assigning a greater weight to unseen words.

5.3 Multinomial Logistic Regression

Given that the task involves multi-class classification methods, the *lbfgs* solver is used. Logistic regression can be paired with the use of a penalty-assigning hyper-parameter *C*, a regularization term used to calibrate the model. Different values of *C* are experimented with in order to determine that *C=7.0* yields the best results for the multinomial logistic regression model. As the value of *C* increases, there is an optimal value for which the accuracy of the model is at its highest, reaching a point of equilibrium. If the penalty value is too small, the model tends to overfit; whereas if the penalty is too high, it underfits. As for the maximum number of iterations hyperparameter, it was set to *max_iter=100* for all tests.

5.4 Random Forest

For the tree-based model, several hyper-parameters are tuned in order to explore the impact on the accuracy of the model. The value of *n_estimator* controls for the number of trees used in the model, while *max_depth* determines the depth of the trees. As the value for both hyper-parameters increases, the prediction accuracy improves, since the model becomes more complex. However, the downfall is that it would also increase the computational time and resources required to run the model. The other hyper-parameter *n_components* from *TruncatedSVD* reduces the feature dimensionality. In the current context, a smaller feature space would allow certain features to have a larger weight. Reducing the dimensionality too much would lead to an over-simplification of the features, which consequently would result in poor performances. Experimenting with different combinations for these hyper-parameter values, *n_estimator=300*, *max_depth=3*, *n_components=32* gave the best prediction accuracy.

5.5 LSTM & BiLSTM

Since neural networks are complex models, several hyper-parameter values are tuned in order to control the computational complexity, or essentially the time and resources required to train the model. Due to computational limitations, the vocabulary size, length of the blog posts and feature dimensionality needs to be reduced. For this reason, values of *vocabulary_size=10000*, *max_words_per_blog=250* and *embedding_dim=100* are chosen to mitigate this constraint. Additionally, there are also other hyper-parameters that can reduce overfit. The *dropout*, *recurrent_dropout*, *spatial_dropout1D* are all hyper-parameters that attempt to reduce overfit by discarding some information to encourage the model to generalize better. The value of dropout dictates the proportion of inputs in a unit to be forgotten, whereas recurrent_dropout removes certain links between units in the LSTM model. The hyper-parameter for the *SpatialDropout1D* layer omits information from the feature space. Testing with different values for these hyper-parameters, the model gives the best accuracy when *dropout=0.4*, *recurrent_dropout=0.4* and *spatial_dropout1D=0.4*. As we started training the LSTM model, overfitting occurred on the first epoch, see *Figure 8 and Figure 9*, this led to poor performance on the test splits (*Table 1 #6*). Furthermore, the BiLSTM model under the same configuration of hyperparameters started overfitting after the first epoch, see *Figure 10 and Figure 11*. This indicates that the hyperparameters or the overall network layers structure must be tuned in order to reduce the capacity of the model. As an experiment, we've reduced the number of units in the LSTM layer from 100 to 32 and saw no major improvement in reducing overfitting for the BiLSTM model (*Figure 12 and Figure 13*).

5.6 Universal Sentence Encoder

Our model uses this encoder as preprocessing as input to the following layers: an input layer, two dense layers of 32 units each with *relu* activation function, a dropout layer with the dropout hyperparameter set to 0.5 and a dense layer with a softmax activation function. The model is compiled with the Adam optimizer and categorical cross entropy loss function. Limited on RAM on the Google Colab's platform, the model was trained with 5% of the training dataset corpus. In our experiments, there weren't any signs of overfitting in the training phase, see *Figure 14 and Figure 15*. This means that the dropout hyperparameter fraction value could be lessened and the number of units in the dense layers could be raised to increase the capacity of the model and obtain possibly better performance.

6. Discussion

6.1 Computational Resources Increase

Since the tasks were executed on a personal laptop with limited RAM and CPU capacity, certain processes and models were constrained by this factor. For instance, the more complex models such as the decision trees and neural networks were simplified in terms of feature space or algorithm complexity, since the resources available were not sufficient to carry out these models to their full potential. It would be interesting to see how an increase in computational resources can influence the prediction accuracy. Perhaps a model that is less performant with the current settings would actually be the best given enough time and computational resources.

6.2 Exhaustive Hyper-parameter Tuning

For each of the models previously mentioned, a distinct set of user-defined hyper-parameters are tested to find the optimal values. There exists several hyper-parameter optimization approaches that could provide a more refined search of the best hyper-parameter values. For example, grid search is an exhaustive hyper-parameter sweeping technique used to finding the best values. However, since it suffers the curse of dimensionality, more computational power would be required to apply this technique. Another interesting optimization method involves progressively replacing sub-optimal hyper-parameters with a crossover of the other values, referred to as evolutionary optimization.

6.3 Refined Normalization Process and In-Depth Analysis

Given the more general scope of this project, all the various pre-processing and normalization steps were analyzed concurrently as sub-groups. It would be insightful to break down each component and measure the individual impact. For instance, analyzing the effects of standardization on a certain set of related words compared to another set could provide a finer and a more detailed understanding of their impact. Essentially, a refined micro-analysis of each component in the pre-processing and normalization process could potentially lead to unveiling new findings.

7. Conclusion

The current work presents an overview of the effects of pre-processing, feature design and model selection on the prediction accuracy of blog posts classification. Based on the results obtained from the experiments, it is shown that features related to sentence length/count and swear words count lead to a higher performance, while the other features and normalization processes resulted in worse prediction accuracy. The results also show that among the models tested with different hyper-parameters, the Multinomial Naive Bayes generated the best predictions on the provided test datasets.

8. References

Andrew Y. Ng, and Michael I. Jordan. On Discriminative vs. Generative classifiers: A comparison of logistic regression and naive Bayes. *23-Sept, 2002*.

Yarin Gal, and Zoubin Ghahramani. A Theoretically Grounded Application of Dropout in Recurrent Neural Networks. *arXiv:1512.05287*, 2016.

9. Appendices

Table 1 - Model performance comparison

#	Model	Training time (mins)	Accuracy[1]	Precision[1]			Recall[1]			F1[1]		
				C0	C1	C2	C0	C1	C2	C0	C1	C2
1	Dummy (most frequent class)	0.08	0.47	0	0.47	0	0	1.00	0	0	0.64	0
2	Multinomial Naive Bayes no features	3	0.6895	0.803	0.64	0.635	0.688	0.842	0.303	0.741	0.727	0.409
3	Multinomial Naive Bayes with features[2]	51	0.6896	0.804	0.641	0.635	0.688	0.842	0.305	0.741	0.727	0.409
4	Multinomial Logistic Regression with features[2]	53	0.6401	0.723	0.721	0.426	0.745	0.553	0.658	0.733	0.624	0.515
5	Random Forest with features[2]	64	0.5579	0.673	0.524	0	0.440	0.868	0	0.532	0.653	0
6	LSTM	582	0.5053									
7	BiLSTM	624	0.5021									
8	Universal Sentence Encoder (with Google Colab's GPU)	32	0.5614	0.580	0.601	0.372	0.758	0.524	0.290	0.658	0.561	0.326

C0, C1, C2 represents the class of the age group.

[1] Accuracy, precision, recall, F1 values are the average of the test splits

[2] Selected features were the blog post length, the sentence counts and the swear words counts

Figure 1 - Blog post length

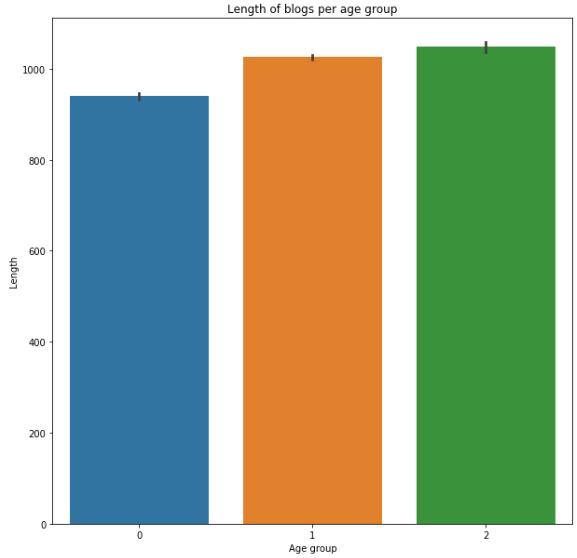


Figure 2 - Blog post word count

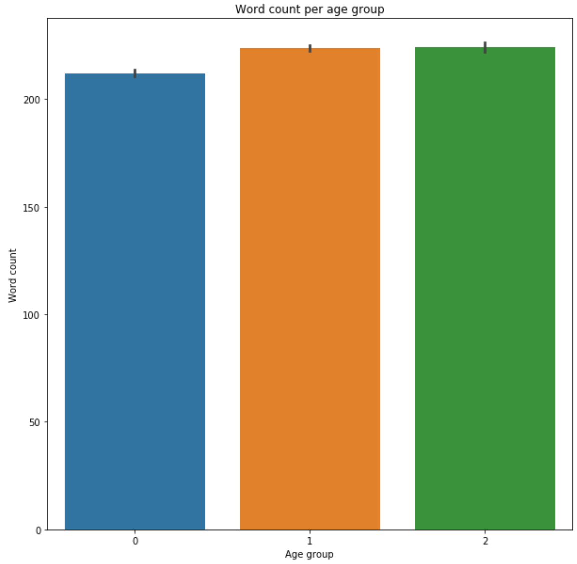


Figure 3 - Blog post sentence count

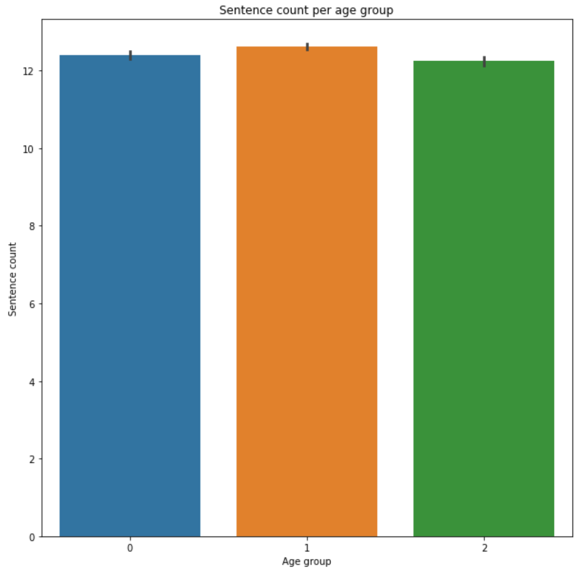


Figure 4 - Flesch Reading Ease

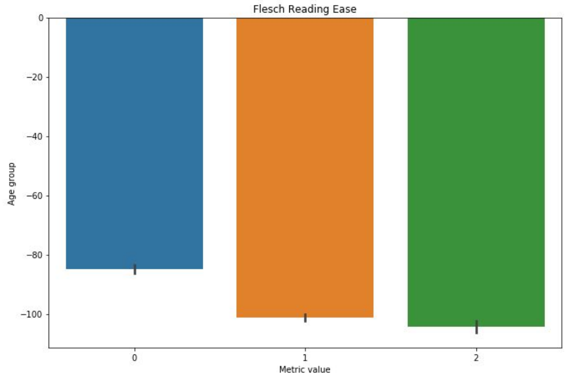


Figure 5 - Flesch-Kincaid Grade Level

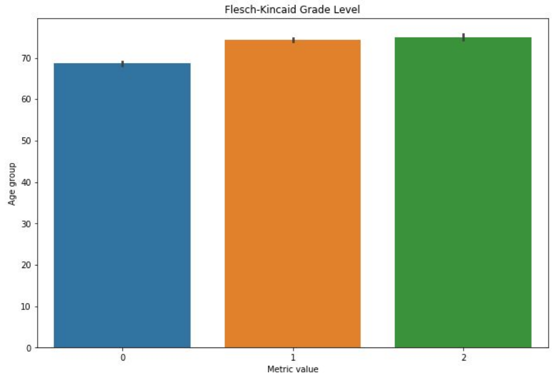


Figure 6 - Polysyllabic words

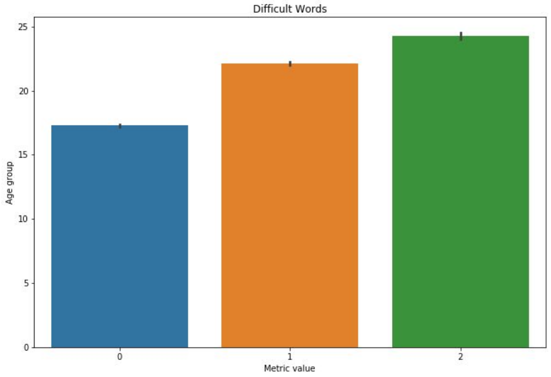


Figure 7 - Swear word count

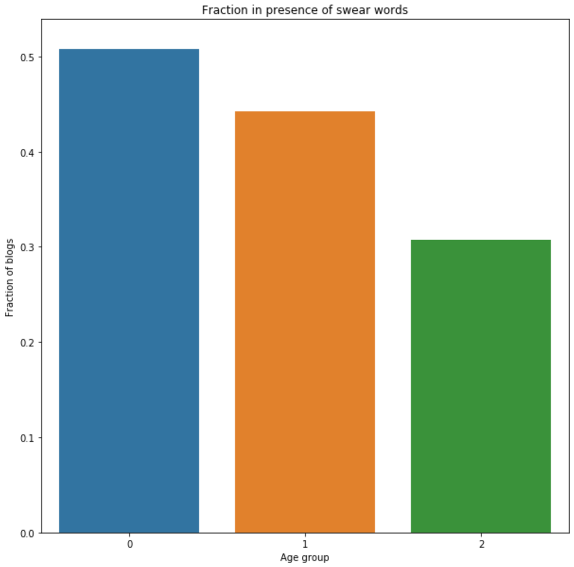


Figure 8 - LSTM loss curves

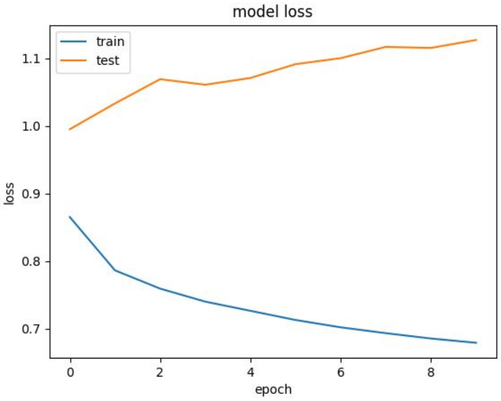


Figure 9 - LSTM accuracy curves

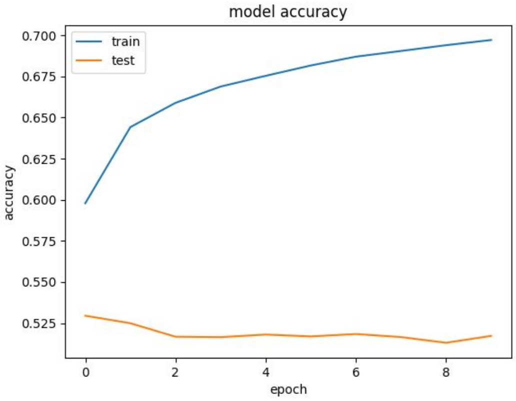


Figure 10 - BiLSTM loss curves with 100 units

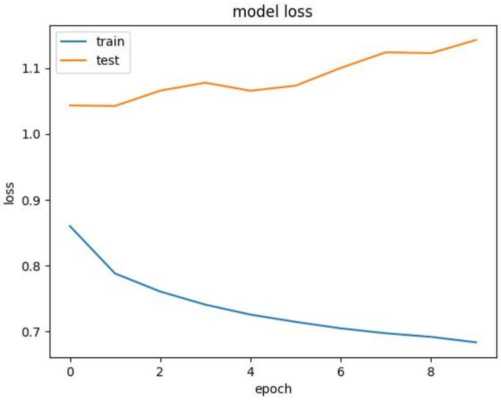


Figure 11 - BiLSTM accuracy curves with 100 units

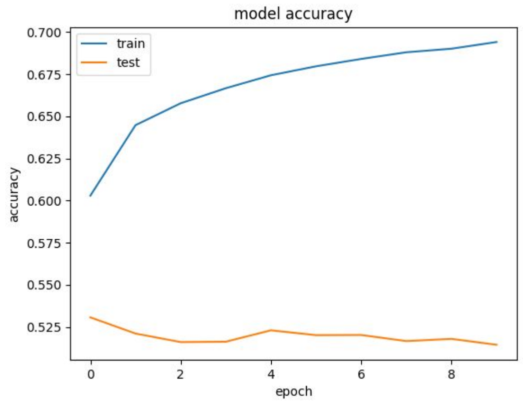


Figure 12 - BiLSTM loss curves with 32 units

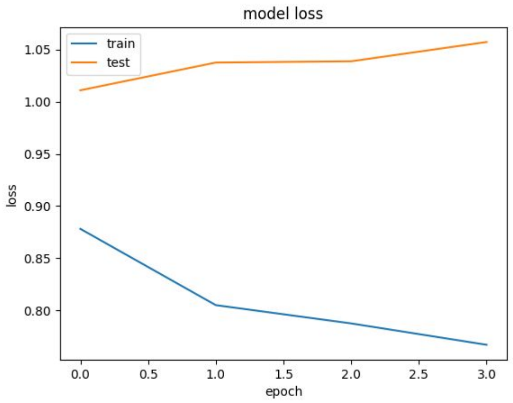


Figure 13 - BiLSTM accuracy curves with 32 units

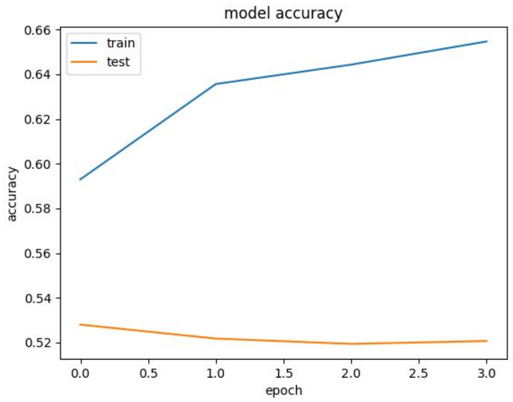


Figure 14 - Universal Sentence Encoder loss curves

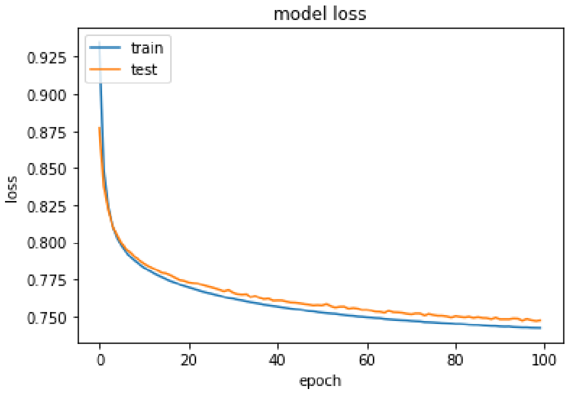
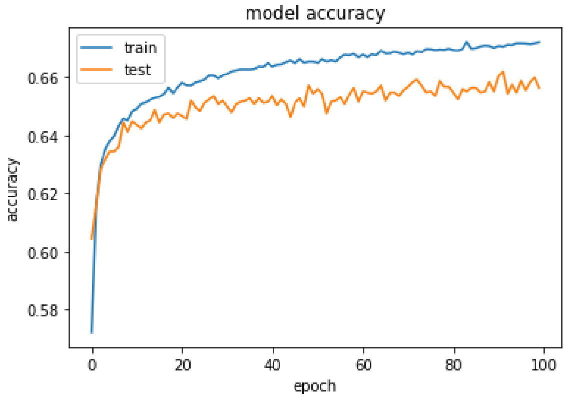


Figure 15 - Universal Sentence Encoder accuracy curves


model accuracy
train
test
accuracy
epoch