SCIENTIFIC DOCUMENT CLASSIFICATION

Group 51

NOVEMBER 8, 2020

Abstract

Published scientific works have been increasing in numbers yearly and now faces a challenge of effectively categorising and grouping relevant articles. In this paper, we proposed three models for classifying scientific journal articles, including Multinomial Linear Regression, Linear Support Vector Machine and ULMFiT (FastAI) models. These models are built on datasets sourced from Arxiv, an open-access archive for scholarly articles. Several pre-processing steps were done on the data, such as removing unwanted characters, removing stop words and lemmatizing. Feature generation such as word embeddings and autoencoder were also applied. These models resulted in accuracies of 54.84%, 53.94% and 58.7% with the test data for Multinomial Linear Regression, Linear SVM and ULMFiT (FastAI) respectively.

Table of Contents

1 Introduction

Published works in academia continue to grow every year, expanding published knowledge at an increasing rate and adding to an ever growing corpus of scholarly knowledge. One estimate by Jinha (2010) postulated that, at the end of the last decade, there were over 50 million published papers in existence [2]. Another estimate from Landhuis (2016) more recently found that PubMed, a popular database for biomedical related research, adds a new paper every thirty seconds on average [8]. A victim of its own success, academia now faces a credible challenge of effectively categorising and grouping relevant articles such that new and existing scholarship can bring the maximum benefit to new research and society at large.

This paper investigates this issue through the exploration of a subset of academic papers and aims to build a multi-class classification model capable of effectively categorising scientific journal articles using the journal abstract. The subset of abstracts will be sourced from the arXiv archive, a curated collection of over 1.7M journal articles focussing on topics in science and technology.

The authors undertaking this challenge approached the problem through a decomposition into three distinct tasks. First, the model cleaning stage was initially implemented by member 1 before being reviewed and improved upon by member 2. Second, feature engineering was undertaken by each of the three members to maximise exploration of potential feature extraction options, these included term frequency - inverse document frequency undertaken by member 3, word embedding undertaken by member 1, and autoencoding undertaken by member 2. Finally, the model construction and selection phase was initially distributed where all team members pursued multiple different models using the generated feature data, however, as time progressed, the team took the most promising models generated and focused on these, until the team was exclusively developing a neural network using the fastai package. Other models investigated include neural networks models using the Keras package created by member 1, while member 3 created and evaluated multiple models including a linear support vector machine, random forests and multinomial logistic regression classifiers. Tweaking of the final model performance was managed by member 3 and member 2.

Ultimately we will put forth a fastai implementation of a LSTM neural network which uses an encoder to extract features from the abstracts and predict the paper class (ULMFiT).

2 Pre-processing and feature generation

2.1 Pre-processing

Text preprocessing is an important step in natural language processing as raw data is rarely suitable for immediate use in machine learning models. This paper employed a straightforward approach for data cleaning, leveraging the nltk and string python packages to remove unwanted characters, stopwords and punctuation before lemmatizing remaining words. During cleaning, the non-word characters were deliberately replaced with a whitespace to break out equation strings in each document. This extracted additional valuable words like ‘pi’ and ‘phi’ which would otherwise have been removed or resulted in extremely long and nonsensical words.

The figure 1 shows the top 10 words that are commonly used in our training dataset after we did the text pre-processing part. We can see that the top 5 most commonly used words are “model”, “use”, “show”, ”result” and ”method”. It is in accordance with our datasets, which are abstracts of scientific documents, that use these words often. On the other hand, it was observed that there is a huge class imbalance in the training set as shown in figure 2 showing the 10 most frequent labels . The class “cs” had a remarkably large number of entries compared to other categories.

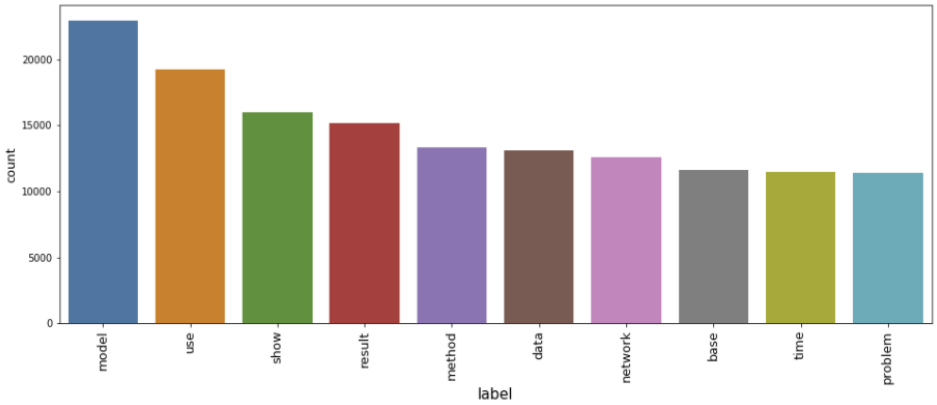

Figure 1. Top 10 words after pre-processing on the training dataset

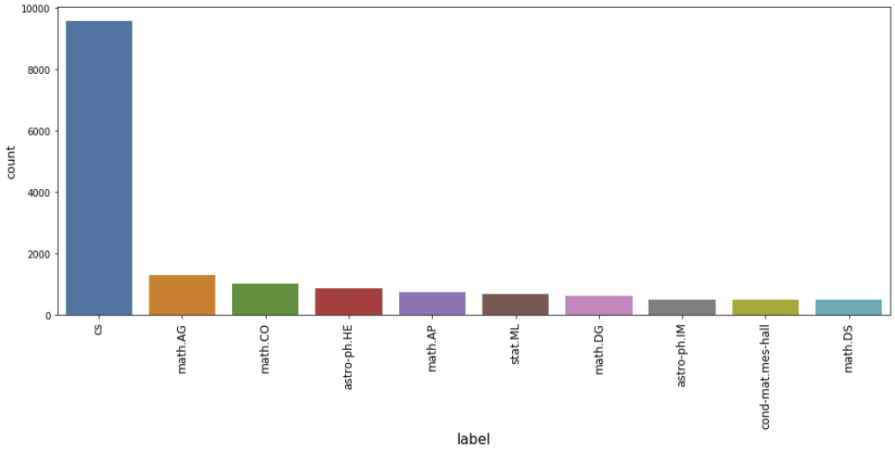

Figure 2. Top 10 labels after pre-processing on the training dataset

2.2 Feature generation

A significant challenge for feature engineering when it comes to text data is how to represent variable length input strings in a way that a computer can manage effectively. There have been multiple approaches developed to tackle this problem such as bag of words, n-grams, TF-IDF, word embedding, GLOVE and autoencoding. In this paper, we selected the word embedding and encoder method as feature generators for our classification models. This was done to more effectively manage sparsity in input vectors.

2.2.1 Word Embeddings

Word embeddings are a representation of words that translate words into a vector space where similar words share similar representations within the space. This paper utilised the Word2Vec package which uses a continuous bag of words approach where a neural network is trained to predict a word given a window of surrounding words. In this way, the network can extract word semantics based on surrounding words in a document. We elected to attempt to generate features using word embeddings as the resultant vector tends to be a dense representation meaning that the vector sparsity is less of an issue for word embedding than it is for other methods. The word embeddings were averaged for each document and these resulting vectors were used to generate the linear support vector machine and multinomial logistic regression models [15]. The result of the embeddings can be viewed in figure 3 which shows a PCA reduced sample of our vector space. Note that words like ‘scale’ and ‘larg’ are grouped as are words like ‘construct’ and ‘bound’, demonstrating the semantic information learned by the network.

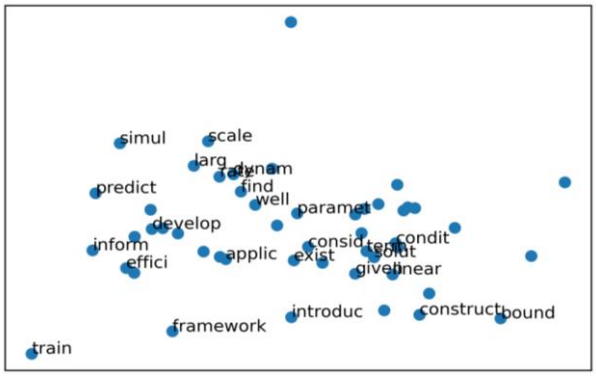

Figure 3. 2D Embedding visualisation using PCA

2.2.2 Autoencoder

An autoencoder also effectively manages the sparsity of input vectors although in a manner somewhat different to word embeddings. An encoder functions by building a model that learns a task different to our initial classification problem. In this case, we train a model that, when given a word, will predict the next word in the document. In doing so we create an encoder-decoder structure, where processed text is encoded into a dense vector and then passed to the final layers of the network to be decoded into a prediction. This is valuable as we can extract the encoding portions of the network and effectively append this encoder onto another network performing a different classification task.

To build this model we used the AWD-LSTM model from the fastai package. An LSTM, or long-short term memory, uses a recurrent network architecture which features feedback loops, meaning that the impact of prior inputs can persist within the network as new inputs are fed in. The AWD-LSTM adds a dropout function which generalises the DropConnect method from Wan (2013) by randomly dropping connections in the hidden layers with a probability 1 - p [17].

Since we are using a function from the fastAI library, we cannot generate any statistics from that function result because it does not have any attributes or function to be called that can give us a statistic visualization. The only thing we can do with the function/learner result is to see the accuracy. From the figure 4 below, we can see that after training the data 8 times (epochs) using several learning rates and freezing-unfreezing layers, on the last epoch we get the accuracy of 12%. The number can already be considered to be decent since we have a large number of words and word sequences.

train_loss	valid_loss	accuracy
5.895255	5.944240	0.120648

Figure 4. Accuracy on last epoch

3. Models

3.1 Multinomial Logistic Regression

Multinomial logistic regression is one of the supervised machine learning algorithms for multi-class classification, which is a simple extension of binary logistic regression for binary classification. In addition, as it is one of the probabilistic discriminative models, it uses maximum likelihood estimation to evaluate the probability of categorical labels. One of the assumptions of the multinomial logistic regression is each independent variable has a single value for each case [5].

In our case, the multinomial logistic regression was built on the dataset which has the word2vec as independent variables and labels as dependent variables. The dataset was split into training and validation datasets in the ratio of 80:20 for getting model accuracy on the validation dataset. After the model was trained on the training dataset, it was able to get the model accuracy on both validation and kaggle test datasets, by comparing the real classes and estimated classes which had the highest probability among the labels.

One thing to be noted here was there were a number of cases which had multiple labels in the given training dataset as shown in figure 5. Thus, it was required to compare the accuracy of the model with others to see how violation of the assumption of the multinomial logistic regression affected its performance.

	train_id	abstract	label
24966	24967	$ \newcommand{\r}{\ensuremath{\mathbb{r}}} \ne...	math.FA
25022	25023	$ \newcommand{\r}{\ensuremath{\mathbb{r}}} \ne...	math.NT
6774	6775	$\mathbb{Z}_2$ topological insulators for phot...	cond-mat.mes-hall
3380	3381	$\mathbb{Z}_2$ topological insulators for phot...	cond-mat.quant-gas
2085	2086	'big' high-dimensional data are commonly analy...	cs
...	...	...	...
23718	23719	zero-curvature representations (zcrs) are one ...	math.RA
23320	23321	zonal jets in a barotropic setup emerge out of...	nlin.PS
16582	16583	zonal jets in a barotropic setup emerge out of...	physics.ao-ph
9548	9549	{\it ellsberg thought experiments} and empiric...	cs
27707	27708	{\it ellsberg thought experiments} and empiric...	q-fin.EC

9892 rows × 3 columns

Figure 5. The cases with multiple labels

3.2 Linear Support Vector Machine (Linear SVM)

Linear support vector machine, like any other support vector machine(SVM), will try to find a hyperplane (decision boundary) which represents the largest separation between two categories. This can be done by finding the hyperplane that the distance between it and the nearest data point of each category is maximized [10].

Since our data does not only have two categories (multiclass classification problem), we will use the help of one-vs-all (one-vs-rest) method. The way this method works is by creating N-binary classifier models with N is our number of class instances in our dataset [3]. Each model is trying to distinguish one class from all of the other classes, so here since we have 100 different classes/labels, we will also build 100 linear models where each model resulted in a hyperplane that separates one specific class with the other classes. In the end we will combine all of the hyperplanes to give our final result.

3.3 ULMFiT (FastAI)

ULMFiT (Universal Language Model Fine Tuning) is an approach in FastAI library that uses an AWD-LSTM model that was designed to produce a multi-class output. This approach is suitable for almost any Natural Language Processing task without the need to tune any of the hyperparameters. The training approach used here is to train the layers of the network in sequence rather than all at once. This method follows the findings of Lee (2019) which found that training the final layers of the network in sequence when appending a pre-trained encoder network can improve model performance [9]. Fit_one_cycle method is called to train both the feature engineering and classification model, which applies the cyclical learning rate that involves varying the learning rate up and down during the training rather than the monotonic reduction approach normally used [13]. This results in much faster training times over fewer epochs and reduces the need for parameter turning on models.

This model consists of these three steps [16]:

1. Pretrained Language Model: ULMFit already has a pre-trained language model which is trained from Wikitext-103 that uses Wikipedia on the task of guessing the next word after reading all the words before. It learns especially from articles in Wikipedia using the language English.

2. Custom Language Model: Since the language model using Wikitext-103 is still very broad and general, we need to take an extra step which is making our own language learner on top of it using our training dataset that we have. In this step, we can control and specify the epoch and learning rate combination to use. This step ensures that our model can learn specifically from the collection of words in our dataset.

3. Custom Classifier: Finally, after the language model is built, we can then build and train a classifier model. The method that we use here is Averaged-Weight Dropped - Long Short-Term Memory (AWD_LSTM) as its classifier.

3.4 Discussion of model difference(s)

The types, advantages and disadvantages of the models used in the project can be found in the table below.

Model	Type	Advantages	Disadvantages
Multinomial logistic regression [1]	supervised	- did not need feature scaling - did not require much computational resource	suitable for the data that each target variable has only one value
Linear Support Vector Machine [6]	supervised	- relatively memory efficient	no probabilistic explanation for the classification
ULMFiT (FastAI) [16]	can be both supervised and unsupervised, but here we used supervised since the labels were given	- minimal feature engineering - simple setup - flexible decision boundary - came pre-trained with wikitext	- required significant computational resources (GPU acceleration was required) - model interpretability was not intuitive

Table 1. Model differences

4 Experiment setups

4.1 Text Pre-processing

As mentioned above, in preprocessing both the train and test data, non-character and non-whitespaces were removed using the regex re.sub(r'[^\w\s]',' ', input_string). Next, case normalisation was performed using Python's lower() function. Following this, tokenization was performed by splitting each document using Python's function split(' '). To ensure all punctuation was removed, we made use of the punctuation method from the string package which removes the "!"#$%&'()*+, -./:;<=>?@[\]^_`{|}~" characters. Afterwards, we removed all stopwords using the English stopwords dictionary from the nltk package, and any blank entry word tokens were also removed. Lemmatization was then performed. Here, from the nltk pos_tags function, we identified the tag for each token, either noun, adjective, verb or adverb with words not conforming being treated as nouns. Token and tag pairs were then inputted to the nltk lemmatize function. Finally, any words having less than two digits were removed. Also note that we specifically removed a row having only the word "none" as its abstract after all the pre-processing steps ("there is none" in the original data), since it does not carry any meaningful result. After this step was done, we ended up with 29637 rows of training data which we will use to train our models with.

4.2 Feature engineering

Two features were selected for model development, word embeddings and an autoencoder.

4.2.1 Word Embeddings

Word embeddings were generated through the use of the Gensim package. Gensim offers a function, Word2Vec, which generates a word embedding given an input corpus. The final parameter selections for the Word2Vec function were vector size = 100, window = 5, min_count = 0 and seed = 1. The vector and window size parameters were selected by comparing the performance of resulting models on a combination of different parameters. To utilise the word embeddings as a representation for a document, the word embeddings making up each document were summed and divided by the count of words which yielded a final average vector for each document. Additional experiments were performed by taking both element wise average and standard deviation for each document, however this resulted in poorer performance.

4.2.2 Autoencoder

This autoencoder is being done when we use the ULMFiT model in fastai. Using the AWD_LSTM (Averaged-Weight Dropped - Long Short-Term Memory) network that will serve as the encoder, we build a custom language model on top of the pretrained Wikitext-103 model. To do this, we first load the training data, with the columns Label and Words where label is the response variable and words is the abstract with each word separated by a space, using the function TextDataLoaders. We set the validation to be 10% of the data (valid_pct) and set is_lm to True to alert that we're using the data to construct a language model.

Then, we call the function language_model_learner with AWD_LSTM set as the architecture of the network, accuracy as the metric that we want to use to measure the performance and the weight decay (penalty) to the loss function of 0.1. The model will then be trained once using a learning rate of 1e-2, which can be one of the best learning rates found by the lr_find() function. Then we will freeze all the layers except the last two and train these once with learning rate of slice(1e-2/(2.6**4), 1e-2). Afterwards, we will freeze again the layers but this time keeping the last three layers unfrozen instead of two and train the model once with learning rate slice(5e-3/(2.6**4), 5e-3). Finally, we will unfreeze all of the layers and train the entire model five times using the learning rate 1e-3. The result of this model will then be saved to be used later in the classifier model.

4.3 Model Construction and Training

There were three models we have developed for predicting the corresponding scientific fields of the source documents: Multinomial logistic regression, linear support vector machine (linear SVM) and fastai (ULMFiT).

4.3.1 Multinomial Logistic Regression

The multinomial logistic regression model was built on the word2vec dataset which contains the non-numeric categorical classes as labels. Thus, the labels were converted to numeric values by using the pd.factorize() function from the pandas package to be used for making predictions readily. Then the dataset was split into training and validation datasets from train_test_split function as mentioned in the Models section. For yielding the best multinomial logistic regression model on our given dataset, the LogisticRegression(multi_class='multinomial', max_iter=500, random_state = 123) and GridSearchCV(cv=3) functions were used together to get the best combination of the hyperparameters by borrowing the concept of cross-validation. As a result, we could get the following parameter values: solver = 'newton-cg' (which is one of the kernels for multiclass problems), penalty = l2 (which represents Ridge Regression) and C = 0.04 (which stands for Inverse of regularization strength) [12]. Thus, we built the multinomial logistic regression model on the training dataset based on these hyperparameter values with the fixed seed for reproducibility. Afterwards, by importing the sklearn.metrics package, we could get the accuracy and confusion matrix of the model on the validation dataset. In the same way, it was possible to get the accuracy of the multinomial logistic regression model on the kaggle test dataset.

4.3.2 Linear Support Vector Machine (Linear SVM)

The Linear support vector machine was also developed on the word2vec dataset. Therefore, we need to convert the categorical target values to numerical values using the pd.factorize() function. Generally, it is recommended to do feature scaling prior to the model training, because the optimal hyperplane of the linear SVM is sensitive to the scale of the input features, since the optimization of the model occurs by minimizing the decision vector [14]. Thus, the scale of our feature data was also normalized by using the StandardScaler() function from the sklearn.preprocessing package. Afterwards, the dataset was split into training and validation datasets in the same way as described in the multinomial logistic regression section. To get the best hyperparameter of the linear SVM which is called as C(Inverse of regularization strength), the LinearSVC() and GridSearchCV(scoring= 'accuracy', cv=3) functions were implemented on the training dataset. The best C found as the result of those functions was 0.1. Furthermore, the model evaluation metrics, which are the confusion matrix and accuracy, were calculated from the sklearn.metrics package to check the model performance of the linear SVM with the best C on both the validation and kaggle test datasets.

4.3.3 ULMFiT (FastAI)

Generally, the result of this model cannot be reproduced. Thus, we used seed to be able to reproduce the same exact model by,

1. Use a seed_value equal to 123456789 for the following functions: np.random.seed(seed_value), random.seed(seed_value), os.environ['PYTHONHASHSEED'] = str(seed_value), torch.manual_seed(seed_value) & torch.cuda.manual_seed(seed_value).
2. Adjust the following torch backend parameters to avoid the use of non-deterministic algorithms: torch.backends.cudnn.deterministic = True & torch.backends.cudnn.benchmark = False

The final classifier model was performed through the application of an AWD_LSTM network incorporating the encoding network generated above. The model can be produced by following the below steps:

1. Use the fastai method TextDataLoaders.from_df to load the training data into a data bunch object that can be used to train our model, the parameters are as follows:
    - df = the data frame created in the feature engineering section; text_cols = 'Words', this parameter identifies the column containing the words in the input dataframe; label_col = 'Label', this parameter identifies the column containing the response variable for the given document; valid_col = 'test' that specifies which data is used as validation (20%) and training (80%)
2. Create a new model using the fastai text_classifier_learner method with the following parameters:
    - dls is equal to the TextDataLoaders.from_df object created in step 1; arch = AWD_LSTM is the model architecture for the classifier; metrics = accuracy; drop_mult = 0.5, which scales the default dropout probabilities of the AWD_LSTM network.
3. Load the encoder built earlier using the fastai load_encoder method. Note that the only parameter is the file location and name of the saved encoder.
4. Perform training on the classification model according to the following schedule:
    - Call the lr_find() function to find the most optimal range of learning rates
    - Call the fasitai function model.fit_one_cycle with 1 epoch (numEpochs) and 1e-2 learning rate
    - Call the fastai function model.freeze_to with setting the parameter n= -2. This will unfreeze the model up to the second last layer, making these unfrozen layers available for weight updates, while the rest of the model remains frozen.
    - Train the model again with 1 epoch and learning rate = slice(1e-2/(2.6**4), 1e-2)
    - Call the fastai function model.freeze_to again with n = -3 (freeze layers except the last three) and call fasitai function model.fit_one_cycle again with 1 epoch and learning rate = slice(5e-3/(2.6**4), 5e-3)

- Finally, call the fastai function model.unfreeze with default parameters to make the entire model available for training and then call the fasitai function model.fit_one_cycle for the last time with 2 epochs and slice(1e-3/(2.6**4), 1e-3) learning rate.

5 Experimental results

	Accuracy	
Model	Validation dataset	Kaggle test dataset
Multinomial Logistic Regression	53.45%	54.84%
Linear Support Vector Machine (Linear SVM)	52.56%	53.94%
ULMFiT (FastAI)	55.7%	58.7%

Table 2. Model comparison based on accuracy

As shown in table 2, it was observed that ULMFiT (FastAI) had the highest accuracy both on the validation and test datasets between the models. From this result, since the multinomial logistic regression is also one of the generalized linear models, we can assume that the models which have no linear decision boundaries could show better performance on our given datasets [4]. In addition, with respect to the assumption of the logistic regression that each independent variable has a single value, the violation of the model assumption did not have a big impact on its performance on our datasets when comparing its accuracy to the linear SVM.

6. Conclusion

The objective of this paper was to build multi-classification models that can accurately predict the corresponding scientific fields of scientific documents from arXiv. Since the raw text data was not suitable for immediate use in machine learning models, data preprocessing was performed on both the train and test data. Afterwards, we implemented the feature engineering and selected the word embedding (word2vec) and autoencoder as our feature engineering methods. In the model building process, the multinomial logistic regression and linear support vector machine (Linear SVM) were developed using the word embedding (word2vec), whilst the ULMFiT (FastAI) was built based on the autoencoder. To yield an optimal classifier, the hyperparameter tuning was performed for the models. As a result, the ULMFiT (FastAI), using the autoencoder, showed the best performance with the accuracy 55.7% on our validation dataset and 58.7% on the Kaggle dataset.

The challenge left in the future will be to retrain the models until the accuracy cannot improve again. This task can be achieved by overcoming the difficulties found in carrying out the project. One of the difficulties we have had in this project was the lack of background knowledge on scientific research on various subjects. With the expert knowledge in related fields, we could extract more useful information especially in the preprocessing or feature generation steps. On the other hand, the imbalance of classes was observed in the training dataset, which can make it difficult for models to be trained in the right direction. Even though there were a number of attempts to make the dataset balanced by enriching data and oversampling, we were not able to see meaningful results. Thus, investigating more methods about the issue can contribute to increasing the accuracy of the classifier. Additionally, to test the assumption we have made in the experiment results section, which is that our given data is not likely to be linearly separable, we could try developing new non-linear machine learning models, for predicting the corresponding labels more accurately. Lastly, as we found in the earlier section, there were some entries with multiple labels. However, the models used in this task only returned one label. From this point, it is expected that the accuracy can be improved with building models which can solve the multi-label problems.

References

[1] Advantages and Disadvantages of Logistic Regression. Retrieved 3 November 2020, from http://www.jaqm.ro/issues/volume-5,issue-2/pdfs/bayaga.pdf

[2] Jinha, A. (2010). Article 50 million: an estimate of the number of scholarly articles in existence Learned Publishing, 23 (3), 258-263 DOI: 10.1087/20100308

[3] Band, A. (2020). Multi-class Classification - One-vs-All & Pne-vs-One. Retrieved 2 November 2020, from https://towardsdatascience.com/multi-class-classification-one-vs-all-one-vs-one-94daed32a87b.

[4] Carpita, M., Sandri, M., Simonetto, A., & Zuccolotto, P. (2014). Data Mining Applications with R. Retrieved 4 November 2020, from https://www.oreilly.com/library/view/data-mining-applications/9780124115118/xhtml/CHP014.html

[5] Jon, S., & Moske, A. (2011). Multinomial Logistic Regression. Retrieved 2 November 2020, from https://it.unt.edu/sites/default/files/mlr_jds_aug2011.pdf

[6] K, D. (2019). Top 4 advantages and disadvantages of Support Vector Machine or SVM. Retrieved 2 November 2020, from https://dhirajkumarblog.medium.com/top-4-advantages-and-disadvantages-of-support-vector-machine-or-svm-a3c06a2b107

[7] Kowsari, K. (2019). Text Classification Algorithms: A Survey. Retrieved 2 November 2020, from https://medium.com/text-classification-algorithms/text-classification-algorithms-a-survey-a215b7ab7e2d

[8] Landhuis, E. Scientific literature: Information overload. *Nature* 535, 457–458 (2016). https://doi.org/10.1038/nj7612-457a

[9] Lee, J., Tang, R. & Lin, J. (2019). What Would Elsa Do? Freezing Layers During Transformer Fine-Tuning. *arXiv preprint.* Retrieved 31 October 2020 from: https://arxiv.org/abs/1911.03090

[10] Lei, Y. (2017). Intelligent Fault Diagnosis and Remaining Useful Life Prediction of Rotating Machinery. Retrieved 2 November 2020, from http://www.sciencedirect.com/science/article/pii/B9780128115343000032

[11] Merity, S., Keskar, N. & Socher, R. (2017). Regularizing and Optimizing LSTM Language Models. *arXiv preprint*. Retrieved 31 October 2020 from: https://arxiv.org/abs/1708.02182

[12] sklearn.linear_model.LogisticRegression — scikit-learn 0.23.2 documentation. Retrieved 2 November 2020, from https://scikit-learn.org/stable/modules/generated/sklearn.linear_model.LogisticRegression.html

[13] Smith, N. (2017). Cyclical Learning Rates for Training Neural Networks. *2017 IEEE Winter Conference on Applications of Computer Vision*. Retrieved 31 October 2020 from: https://arxiv.org/abs/1506.01186

[14] Sotelo, D. (2017). Effect of Feature Standardization on Linear Support Vector Machines. Retrieved 2 November 2020, from https://towardsdatascience.com/effect-of-feature-standardization-on-linear-support-vector-machines-13213765b812

[15] Tand, D., Wei, F., Yang, N., Zhou, M., Liu, T. & Qin, B. (2014). Learning Sentiment-Specific Word Embedding for Twitter Sentiment Classification. *Proceedings of the 52 Annual Meeting of the Association for Computational Linguistics.* Retrieved 4 November 2020 from: https://www.aclweb.org/anthology/P14-1146.pdf

[16] Transfer learning in text | fastai. (2020). Retrieved 3 November 2020, from https://docs.fast.ai/tutorial.text

[17] Wan, Li., Zieler, M., Zhang, S., LeCun, Y. & Fergus, R (2013). Regularization of Neural Networks using DropConnect. *Proceedings of the 30th International Conference on Machine Learning.* Retrieved 30 October 2020 from: http://http://yann.lecun.com/exdb/publis/pdf/wan-icml-13.pdf