Scientific Document Classification

FIT5149 S2 2020 Assignment 2 Report

GROUP NUMBER - 43

Sarthak Sareen- 30761182

Dishi Jain- 30759307

Table of Contents

1 Introduction

This project focuses on the abstract extracted from some documents and then predicting and returning the corresponding scientific fields of the source documents. The main goal of this project is to -
Create a classifier model and predict the labels based on the abstract feature given.

The project has been done in a group(equal participation by each member) which included data loading, text pre-processing, model creation and prediction. The data used for this project has been created in two parts -

1) The train_data_labels.csv that was provided to us.
2) Using the data from kaggle.com[1]

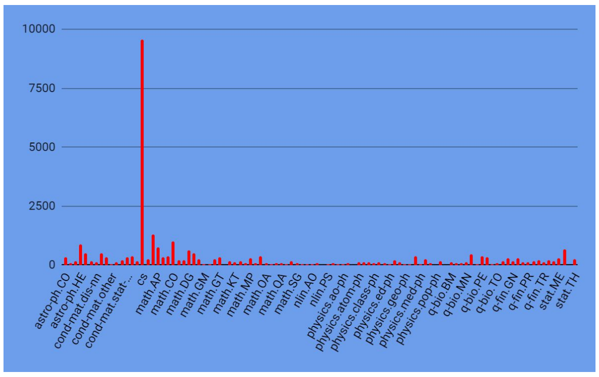
The data from kaggle.com has been used after reading the paper on arxiv.org[2]. The data on kaggle contains 1.7 Million rows which also contains extra labels as given in the original train_data_labels.csv. Hence we have extracted only those rows which predicted the same 100 labels as given and we have then concatenated these rows(roughly 500,000) with the original train_data_labels.csv. This data is then finally used for training of the models.

(As the limit of submission on Moodle was 500MB, the original data from kaggle.com could not be attached. Hence the data created after using the train_data_labels.csv and label specific data from kaggle.com(roughly 500,00 rows) is uploaded on Moodle.)

2 Pre-processing and feature generation

The feature that needs to be processed is the 'abstract' column in the dataset. As ths is a textual column we need to apply various steps of processing on it so that it can be fed to the model for the prediction purpose. After the processing only the textual data can be used for analysing and predicting. Using the pre-processing steps, we can apply feature generation to add new information while creating models.
Also to check the balance of the data, I have grouped the data by label and used the count function to get count of records in each label. Originally the given data train_data_label.csv had imbalanced data as shown in Figure 1 below. It had a very high count of the label "cs". To correct this imbalance I have used the concatenated data from kaggle.com and finally the count of records for each label looks more balanced.

Figure 1 - Class imbalance in original data



2.1 Pre-processing

This task has been carried out by following the given steps-

1) **Tokenization -** Using the RegexTokenizer from NLTK.tokenize I have used the regex "[a-zA-Z]+(?:[-'][a-zA-Z]+)?" to get only the valid tokens including the alphabets only hence removing any numbers, punctuations and invalid characters from the text. This regex will also include the words that are separated by a dash hence words like "pre-processing", "build-up", etc and will also include apostrophe to include words like "don't", "can't", etc.
2) **Lower Case Conversion -** All the tokens generated have been converted to lowercase using the .lower(). This will help in case normalization and hence all the words will be in the same lower format.
3) **Stopwords Removal -** The stopwords include words that do not contribute any significance to the text. These may include words like 'a', 'the' , 'and' , etc. Such words/tokens can be removed from the abstract column. To do this we have used the stopwords.words('english') function that gives all the stopwords in the english language using the NLTK.corpus library.
4) **Stemming -** Each word after tokenization has also been reduced to its stemmed version to the roots of the word known as lemma. The importance of stemming is that it facilitates reduction in the noise in the data. As we have textual data as a feature, hence by applying stemming to the tokens we can get cleaner data. Words like playing, played, play will all be stemmed to their stemmed version play.

2.2 Feature generation

For the feature generation we have used several concepts based on the different model creation. For the Logistic Model and SVM we have used TfidfVectorizer() from sklearn.feature_extraction.text. This will give the tf-idf score for each token present in the individual rows of the abstract column. The tf-idf score is calculated by calculating the TF and IDF separately first.
The TF(Term Frequency) is defined as the Number of time the word occurs in the text/ Total number of words in that text
The IDF(Inverse Document Frequency) is defined as the Total number of documents / Number of documents that have the word t in it.
Hence the final TF-IDF score for a word t is TF*IDF
It is implemented in Logistic Model creation by creating an object of TfidfVectorizer() and then using the object's fit_transform() on the abstract column.
For the Neural Network Model, we have created a Tokenizer() using keras.preprocessing.text that helps in the creation of tokens. Also we have used the fit_on_texts function on the abstract column that updates the vocabulary based on the text provided where each token is assigned a number based on the frequency of that word. Also we have used the texts_to_sequence() on the abstract column to finally get the sequence of integers for each row/abstract based on the words it contains. We have also used pad_sequence() for feature engineering which ensures that all the sequences in the given list have the same length. This is done by padding/adding a zero for those abstract's that have a length less than the specified length(which is the maximum number out of the lengths of all the abstract text)[3]

3. Models

The three models used for the classification are Logistic Regression, SVM(Support Vector Machine) and Neural Network(LSTM - Long Short Term Memory Network). After applying different techniques and predicting new data for these three models, we found that the Logistic Model was giving the best accuracy and predictions. These three models have been selected after trying various other models(approximately 6 others models)

3.1 Logistic Regression

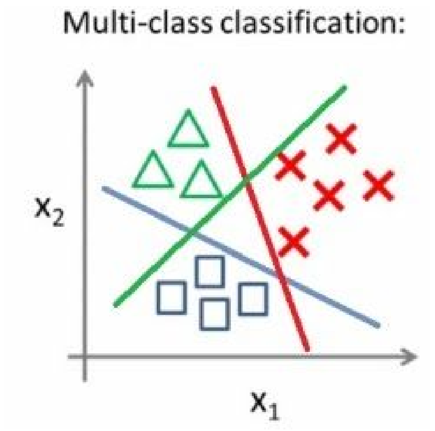
The logistic regression model is a Supervised Machine Learning technique that is used for the classification of binomial or multi classes from a given set of features where the classes are categorical in nature. As the dependent variable is categorical and not ordered hence the Logistic Regression Model is considered as Multinomial. This is also specified while creating the object of LogisticRegression()

Working -
The Logistic Regression Model works with the construction of a linear predictor function. It is a statistical classification problem that is used to predict 1 out of k-possible outcomes. It has various assumptions including the independence between the dependent variable classes. The MLR(Multinomial Logistic Regression) uses a function that gets the set of input features and then performs multiple mathematical operations to normalize the input values into a vector of values that follows a probability distribution. The input features are X consisting of features and the output variable is Y consisting of probabilities for y1,y2,y3...yk for k different target classes(in our case k is 100). The MLR consists of two functional layers-

1. <u>Linear prediction function</u>
   Also known as the logit function, it maps the score of each outcome or the odds of each event of our target variable in the range (-infinity,+infinity)
2. <u>Softmax function</u>
   The softmax function simply converts the logit scores of the outcomes of the feature set to the probability values

Hence for K classes we basically create K-1 independent binary logit models. The MLR also uses a maximum likelihood estimation for determining the regression coefficients. The MLR also uses the 'saga' solver. The saga solver is the only one that allows elastic net regularization. Regularization is a concept that helps in the reduction of the complexity of the model hence preventing overfitting of the model. Hence when the regularization technique is applied with logistic regression then it tends to penalize the large regression coefficients. Hence by using regularization concept we tend to improve the accuracy and the performance of the model while predicting the labels/classes for unseen data.

Figure 2 - Multi Class Classification



3.2 Neural Network - LSTM(Long Short Term Memory Network)

Recurrent Neural Networks or RNNs are used for predicting a category from a sequential input. The RNN uses feedforward neural networks that have an internal memory. By feedforward we mean that the algorithm first takes an input x(0) and then produces an output y(0). This y(0) is then put together with the next sequence of input x(1) to produce the next output y(1). This goes on and hence it remembers the context while training. LSTM is an updated version of RNN.

Working -
For the LSTM model creation we define certain parameters that are required for the building and training of the model. These include -

1. vocab size(number of top common words)
2. embedding dim(size of feature vector where the features are embedded words of the input)
3. max length for each textual input tokens

After the specification of these parameters, we tokenize the given input text and create a vocabulary index/ word index based on the frequency of each word/token. This is done by using the fit_on_texts() of the tokenizer object(`tensorflow.keras.preprocessing.text import Tokenizer`). After this we use the texts_to_sequence() of the tokenizer object to convert each text/sentence to a sequence of integers based on the word index dictionary created. Now each sequence of integers created for each sentence is passed through pad_sequences() to convert each sequence to the same length for further processing and model generation. This is achieved by adding 0s before(truncating_type = pre) or after(truncating_type = post) the original sequence. Now finally after the input text is prepared and in the correct format, we create the Sequential Model and add layers to it. The layers that we used were -

1. Embedding Layer -
   This layer converts the input sequence of strings into the corresponding word vectors. Here word embedding is done and hence each word is represented as a vector and words with similar meaning tend to have similar vectors.
2. Dropout Layer -
   This layer is added to avoid overfitting of the model.
3. BiDirectional Layer -
   This layer trains two LSTMs instead of just one on the input sequence. The first is done on the input sequence as it is and the second on the reverse of the input sequence and then the result is concatenated. This allows the LSTM to learn more.
4. Dense Layer -
   This is the last layer that uses the softmax activation function. This results in a multi class probability distribution over the target class.

While creating the model we also use the Adam optimizer and the sparse_categorical_crossentropy as the loss function. Then the model is created by using the X after the last padding step(pad_sequences()) and Y after using pd.get_dummies() on the label column. [4]

3.3 Support Vector Machine (SVM)

Svm is a well known Supervised Machine Learning Algorithm which is used for both classification and regression techniques. For us, we have a categorical target class and hence SVM is used. The idea of the SVM is that it creates a line or a hyperplane which separates the data into classes and aims to find the optimal hyperplane line between the classes. A hyperplane in an n-dimensional Euclidean space is a flat, n-1 dimensional subset of that space that divides the space into k disconnected parts where k is the classes/labels in our target.

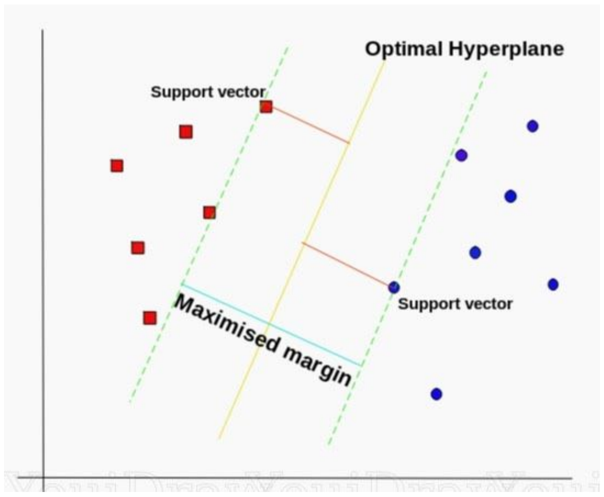
Working - [5]

1. The model finds the points/rows closest to the line from the classes. The points are called the support vectors. The distance between the support vectors and the line is called the

margin. The goal is to maximise this margin. The case where we get the maximum margin, then that line is called the optimal hyperplane.

2. C Value - The regularization parameter controls the tradeoff between the smooth decision boundary and classifying the training points correctly. The larger the value of c means the more training points correctly. A larger value of c will choose a smaller margin hyperplane and hence the training happens more efficiently whereas a small c value will cause miss classifications in the training data as well.
3. Gamma value basically tells the reach of a training example's influence. Hence a high gamma value will only take into consideration the points that are close to the hyperplane and a low gamma value takes into consideration even those points that are far away from the hyperplane.
4. Linear Kernel is the most commonly used kernel on datasets which have a huge variety of labels in the target class.

Figure 3 - SVM Model Hyperplane Concept



3.4 Discussion of model difference(s)

Logistic Regression

Advantages -

1. There is no assumption of homogeneity of variance
2. The independent variables need not be normally distributed or have equal variance in each group
3. It allows models to be updated easily to reflect new data.
4. It does not assume a linear relationship between independent and dependent variables.

Disadvantages -

1. It doesn't perform well when the independent variables are not correlated to the target variable and are similar to each other
2. It requires a large dataset and sufficient training examples for all of the classes/labels to be identified

Neural Network LSTM

Advantages -

1. Can work on inputs of different lengths.
2. Useful when the input is present as a sequence as it allows the words in the text to relate with the words preceding it.

Disadvantages -

1. Require a huge amount of space and time for training the model.
2. They are prone to overfitting of the data hence giving a good accuracy for the validation data but wrong predictions on testing/unseen data

SVM

Advantages -

1. Functions nicely with unstructured and semi structured data.
2. It has good regularization techniques.
3. SVM models show a low risk of overfitting.

Disadvantages -

1. Selecting the correct kernel function is sometimes not easy.
2. The final model generated does not support small calibrations.
3. It takes a long time to train the model and also requires high memory specifications.
4. It is not easy to fine tune the SVM hyperparameters.

4 Experimental setups

For the SVM Model, the hyper parameters included C value, kernel, degree and gamma. The value of degree and gamma were chosen as the default values only of 3 and 'auto' respectively. For the C value it was experimented between 0.5, 1 and 1.5. The c value basically intends on controlling the outliers. A low value of C will mean that the outliers are allowed and a high value means that errors/outliers are not allowed. Hence by selecting a value of 0.5 and 1.5 we were getting less accuracy as compared to when we chose 1 as the c value. For the parameter kernel, there are multiple options. We can have 'linear','poly' and 'rbf' as the kernel values. However when there is a multi label classification we use linear as the kernel value. When we used 'poly' or 'rbf', we found that due to certain conditions the predicted values were completely wrong and had a very low accuracy.

For the LSTM model, we experimented with the epochs value. The epochs value is basically the number of times the entire dataset is fed forward and backward to the model for training purposes. The better the epoch value the better the model is able to convert the underfitting to optimal fitting to overfitting. While adding the layers to the model, the activation function used is softmax and the dropout value is taken as 0.2 which implies the regularization to avoid overfitting.

For the Logistic Regression Model we have a default parameter of 'multinomial' for the argument multi_class. This is used for multi classification. For the solver argument we have options like 'sag', 'liblinear' and 'saga' however 'saga' is most suitable for huge datasets. For the Logistic Regression Model we also tried to use CountVectorizer() instead of the TfidfVectorizer(). However it was yielding a lower accuracy score and hence it wasn't used. The TfidfVectorizer() performs better as it transforms the count matrix to a normalised representation. TheLogistic Model is the final one that we have chosen. The construction of this model can be done by following the below steps -

1) Reading the csv file for training purposes.
2) Creating a Regex Tokenizer object with the given regex, creating an object of PorterStemmer for stemming and an object of TfidfVectorizer() and downloading the stopwords file from nltk.corpus. Next create a dictionary with text as key and list of strings for each abstract(after tokenization,stopwords removal and stemming) as value, and another key as topics with the value as the labels.
3) Fitting and transforming the abstract values from the dictionary created above with the tfidf object.
4) Creating an object of LogisticRegression with parameters as multi_class='multinomial', solver='saga' and using the above abstract values(after fit_transform) and the label values from the dictionary created above into the model for fitting.
5) Use this model to predict the values of abstract from test_csv file after cleaning(as above) and transforming the abstract column(using tfidf objet). Create a new column called label and drop the abstract column for final submission hence containing only two columns test_id, label.

For all the models, we have also experimented with the text pre processing. We tried to modify the abstract feature with and without the stemming process. However it was seen that when the stemming was carried out the accuracy of the model and the correctly predicted values were higher. Hence finally, we decided to implement stemming for each abstract.
Also for the validation and finding accuracy we have used a splitting of 80%-20% on the data. Based on this the X_train and Y_train are used for model creation(just for validation purpose) and then the X_test is used for prediction. Then the predicted value is compared with the Y_test to find accuracy and the count of correct predictions and the count of wrong predictions.

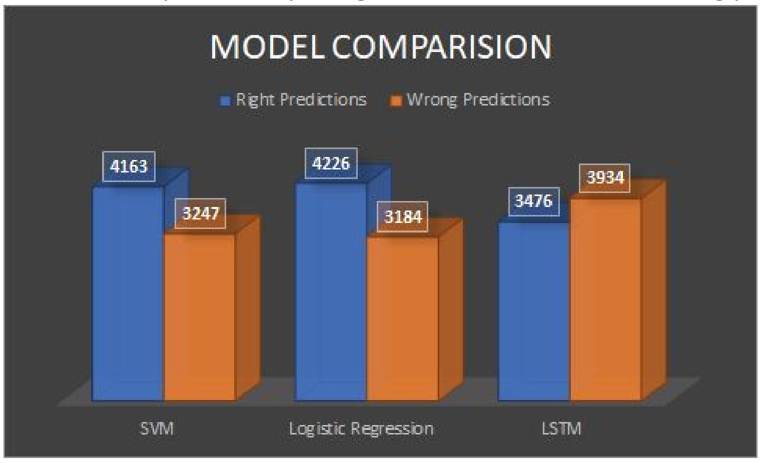
5 Experimental results

The models' accuracy, correctly predicted values and wrongly predicted values are presented in the table above. The accuracy is derived from the Kaggle link as given to us. The Correct predictions and wrong prediction counts are calculated by the % values of accuracy and the count of total rows in the test_data.csv (hence they are approximate values presented just for comparison). Hence by looking at the table above we can see that the Logistic Regression Model performs better than the other two.

Table 1 - Comparison of Models

Models	**Accuracy on Validation Data(in %)**	**Correctly Predicted Values(approximately)**	**Wrongly Predicted Values(approximately)**
SVM	56.19	4163	3247
Logistic Regression	57.04	4226	3184
LSTM	46.91	3476	3934

Figure 4 - Model Comparison depicting number of correct and wrong prediction



Hence when comparing the three models we find out that a Logistic Model is better and creates a better fitting model when the dataset is huge. Considering our case here we have roughly 500,000 rows and hence a Logistic Model is a good choice. Also as we have multi labels i.e. 100 labels we can not predict and create models efficiently for the SVM. This is because it needs to solve quadratic programming problems in order to find a separation hyperplane which hence causes an intense computational complexity for the huge number of rows. Also when talking about the LSTM model we observe overfitting from the graphs plotted between the accuracy and val_accuracy and the graph plotted between loss and val_loss. This hence tells us that the accuracy is better on the validation data but poor on the testing data. Hence out of the three the best model is Logistic Regression Model.

6. Conclusion

1) After training and testing various classification models which are Support Vector Machine, Multinomial Logistic Regression and Long Short Term Memory(LSTM) networks with the training dataset by using different parameters, we got the accuracy highest forLogistic Regression Model amongst all the classification models.
2) In the Logistic regression model the accuracy is 57.04% with the parameters as follows : multi_class = Multinomial and solver = saga
3) Before the model is created, we have undergone the text pre-processing. The feature column given to us is in textual format and hence processing of it is required including stopwords removal, tokenization, stemming and case normalisation. This helps in engineering better features for feeding into the model.
4) The Support Vector Machine model takes a lot of time for training the model as compared to the Logistic regression model.
5) To get different accuracy we can use more models such as KNN and SGDClassifier for the classification of the text in the future.
6) As known to us from the No Free Lunch Theorem that no model is perfect and a model can work well for a certain set of objective functions and still it can perform poorly for another set of objective functions.

References

[1] arXiv Dataset. (2020). Retrieved from https://www.kaggle.com/Cornell-University/arxiv

[2] (2020). Retrieved from https://arxiv.org/pdf/1309.0326.pdf

[3] (2020). Retrieved from https://stackoverflow.com/questions/51956000/what-does-keras-tokenizer-method-exactly-do

[4] Multi Class Text Classification with Keras and LSTM. (2020). Retrieved from https://djajafer.medium.com/multi-class-text-classification-with-keras-and-lstm-4c5525bef592

[5] Support Vector Machines(SVM) — An Overview. (2020). Retrieved from https://towardsdatascience.com/https-medium-com-pupalerushikesh-svm-f4b42800e989#:~:text=SVM%20or%20Support%20Vector%20Machine,separates%20the%20data%20into%20classes