FIT5149 Assessment 2: Sentiment Classification for Product Reviews

Monica | 29873207 | mmon0012
Mukul Gupta | 29873150 | mgup003
Rachana Jobanputra | 29555248 | rjob0001

Department of Information Technology, Monash University
Caulfield East, VIC 3145

Abstract

All models are wrong, some are useful.

-George Box

Here, we took on a sentiment analysis challenge, also known as opinion mining, within Natural Language Processing (NLP). The sole purpose of this challenge is to classify text or reviews according to the sentiment polarities of opinions i.e. strongly negative (1), weakly negative (2), neutral (3), weakly positive (4) and strongly positive (5). For this task, an automatic sentiment classification system has been developed that relies on several machine learning techniques (logistic regression (LR), naïve Bayes (NB), support vector machine (SVM), neural networks (LSTM, ULMFiT)) that can assign a large set of product reviews to the above five levels of polarity of opinion as accurately as possible. We conclude by comparing all the models used for analysis and defining the one model which classifies the sentiment with the most precise accuracy.

1 Introduction

The dataset adapted for this sentiment analysis challenge is a large set of product reviews provided by Yelp. The dataset includes 5,00,00 labelled, 60,00,00 unlabelled records and 5,00,00 test records which spans up to 5 classes of sentiment (1, 2, 3, 4, 5) referring to the five polarity levels. This challenge was well taken by all the group members. Sentiment analysis setting requires three major steps, including generating features, developing a proper classifier, and applying the classifier to the unseen data. Pre-processing steps were equally allocated to all group members. After pre-processing, individual models' implementation was assigned to respective members.

Monica: LR, Fast AI ULMFiT
Mukul Gupta: Ensembled classification (LR, NB, SVM)
Rachana Jobanputra: LSTM neural network

The closure of this challenge, including analysing the results from all the models and laying the accountability of the preferred classifier was documented and assembled by all the members of the group, cooperatively.

2 Extracting Extra Features

Before the actual pre-processing of the data, some preliminarily features were extracted from each of the text reviews from all datasets (labelled, unlabelled, test data). The feature set is mentioned below.

- Text length of the review
- Number of words in the review
- Sentiment polarity of the review using TextBlob (TextBlob, 2019), range [-1, 1] where 1 is for most positive statement while -1 is for most negative statement.
- Number of capital letters in the review, generally capital letters in a review show strong emotion
- Number of exclamation marks in a review
- Number of emoticons used in the review
- Number of question marks in the review

3 Pre-processing

Pre-processing is a critical step involved in a Natural Language Processing (NLP) task, and is used to transform the text data into a set of features such that it is analysable for the task at hand. All the basic text pre-processing tasks were performed in the order mentioned below.

1. Removal of duplicate reviews
2. Removal of NA values
3. Removal of URLs in the review
4. Splitting hashtag phrases by spaces: Hashtag phrases tell a lot about the emotion behind the comment
5. Normalising the words i.e. lowercasing words
6. Removal of punctuations and numbers
7. Removal of single characters and multiple spaces
8. Words are tokenized
9. Spelling of words are corrected using Levenshtein distance.
10. Lemmatizing words using POS tagging: Lemmatizing is preferred over stemming in our approach since it ensures the lemma (root word) belongs to the language. After lemmatization, the adjective form of the word is extracted. The POS tag words which are filtered are adjectives, adverbs, and superlatives.
11. The words are then merged to create lists of reviews and saved as a csv file now containing the pre-processed text along with the 7 features created before.

4 Feature Generation

These features are specific to vocabulary. Two broad approaches are followed leading to feature generation.

4.1 Feature Set1: Unigrams, bigrams and trigrams extracted explicitly

In this approach, bigrams and trigrams were built without removing stop words. 1,000 bigrams and 250 trigrams were filtered. Unigrams were built after removing some stop words. Stop words list was created manually so that important stop words did not include words like 'very', 'good', 'not' and others. More than 10,000 unigrams were filtered out after removing the least frequently used words. Overall number of features are 11470.

a. **Count Vectors**: TfidfTransformer (use_idf = False) on n-grams

Term frequency gives document frequency of the vocabulary created using the n-grams. The count is then normalised.

b. **TF-IDF** (term-frequency inverse document-frequency) features using TfidfTransformer (use_idf = True)

$$idf(t, D) = log\ (\frac{N}{|d \in D : t \in d|})$$

where N is the number of documents and the denominator is the number of documents the word 't' appears (Fredenslund, 2019)

Statistical Summary of Set1:

Vocabulary size: 11470
Total number of tokens: 6239035
Lexical diversity: 543.9437663469921
Total number of articles: 49994
Average document length: 124.79567548105773
Maximum document length: 1060
Minimum document length: 1
Standard deviation of document length: 114.08325817260047

4.2 Feature Set2: Unigrams, bigrams, trigrams were generated implicitly using max_features, ngram_range parameters.

The vectorizer is built with maximum features 1500 and ngram_range = (1, 3) [i.e. using unigrams, bigrams and trigrams]. The vectorizer is then fit over the training pre-processed data to create following features.

a. **Count Vectors**: In this case count vectors are created to find the term frequency using TfidfTransformer (use_idf = False)

b. **TF-IDF** Vectors: In this case TF-IDF vectors are created using TfidfTransformer (use_idf = True)

Statistical Summary of Set2:

Vocabulary size: 1500
Total number of tokens: 30423567
Lexical diversity: 20282.378
Total number of articles: 49994
Average document length: 608.5443653238389
Maximum document length: 4791
Minimum document length: 2
Standard deviation of document length: 555.6031010160556

The models and experiments in the next section incorporates both the approaches. The intuition behind it was that count vector features should perform better since they only take the frequency into account. On the other hand, TF-IDF gives weight to the words that distinguish a particular document that might not be the best approach for sentiment analysis.

5 Models | Classifiers

At this stage, the data contains the pre-processed data with both sets of features. Before building the model, the features are normalised using Min-Max standardisation since features were in different scales. The data is then split into 75:25 ratio as training and testing data.

5.1 Model 1 | Logistic Regression

Logistic regression is one of the base models used in classification problems. They are generally used in binary class problems like pass/fail but are extended to multi class problems. Multinomial Logistic Regression (MLR) is a form of linear regression analysis conducted when the dependent variable is nominal with more than two levels. It is used to describe data and to explain the relationship between one dependent nominal variable and one or more continuous-level (interval or ratio scale) independent variables. Nominal variable can be understood as, a variable which has no intrinsic ordering.

$$l = log_b \frac{p}{1-p} = \beta_0 + \beta_1 x_1 + \beta_2 x_2$$

where l is the log-odds, b is the base of log, and β are the parameters of the model ("Logistic regression", 2019).

The assumptions are that observations are independent of each other, little or no multicollinearity among the independent variables and independent variables are linearly related to the log odds.

Models were built using different feature sets generated before:

1. LR using explicit vocabulary and count vector features.
2. LR using explicit vocabulary and TF-IDF features.
3. LR with implicit vocabulary: Since LR was the better performing model in our analysis, we tried to use it without a pre-defined vocabulary. In this case, a logistic regression model is built using the pipeline technique in python. It doesn't use the pre-processing done in previous steps. Using the raw data, vocabulary length is tuned to 1500. TF-IDF and count vector features are created using unigrams, bigrams and trigrams.

5.2 Model 2 | Support Vector Machine

Another model which performs good with NLP is the Support Vector Machine (SVM). Therefore, it was used as one of our base models.

SVM is a discriminative classifier formally defined by a separating hyperplane mostly used for classification problems. In other words, given labelled training data (supervised learning), the algorithm outputs an optimal hyperplane which categorizes new examples. In two-dimensional space this hyperplane is a line dividing a plane in two parts where in each class lay in either side ("Chapter 2: SVM (Support Vector Machine) — Theory", 2019). It works on the following two assumptions, the margin should be as large as possible and the support vectors are the most useful data points because they are the ones most likely to be incorrectly satisfied.

Models were built using different feature sets generated before:

1. SVM using explicit vocabulary and count vector features.
2. SVM using explicit vocabulary and TF-IDF features.

5.3 Model 3 | Naïve Bayes

It is a classification technique based on Bayes' Theorem with an assumption of independence among predictors. In simple terms, a Naive Bayes classifier assumes that the presence of a particular feature in a class is unrelated to the presence of any other feature.

$$P(x) = \frac{P(c)P(c)}{P(x)}$$

where $P(x)$ is the posterior probability of class, $P(c)$ is the prior probability of class, $P(c)$ is the likelihood which is the probability of predictor given class, $P(x)$ is the prior probability of predictor. (Soni, 2019)

Classification is done by picking the c_i that has the highest probability (Maximum A Posteriori decision rule) (Ray, 2019).

Models were built using different feature sets generated before:

1. NB classifier using explicit vocabulary and count vector features.
2. NB classifier using explicit vocabulary and TF-IDF features.

5.4 Model 4 | Ensemble

Ensemble is known to give better results by combining different models together. In general the max voting method is used for classification purposes. Two or four models are ensembled to predict each data point. In this case, Logistic Regression Classifier, Support Vector Machine and Naive Bayes are ensembled using Voting classifier method provided by sklearn.ensemble. Each model's predictions are regarded as a vote. Predictions from most models predictions are used as final prediction.

5.5 Model 5 | Semi-Supervised LR

After regularization, logistic regression was giving the best results with C=0.2 (explained in experiment results) among the base models using the labelled data.

Unlabelled data can be combined with the model to further improve it. Note that only training data (75%) of data was included to build the model. Rest 25% was used to test the new model. Model were built in 2 ways:

- **Without threshold**: In this method, we predicted samples of unlabelled data (10,000 in the case), predicted them, build a new model including predicted data and the 75% training labelled data and tested on 25% labelled data.
- **With threshold**: In this case, we predicted samples of unlabelled data (50,000) and took only those values which were predicted with a probability of 0.95 or more for our new model. The 75% training labelled data and new samples were used to build the model and tested on the 25% testing labelled data.

5.6 Model 6 | Recurrent Neural Network

RNN makes use of LSTM layers. It stands for "Long Short-Term Memory", which is an artificial Recurrent Neural Network (RNN) architecture model used in deep learning tasks. For the purpose of this assignment, the LSTM method was used for supervised and semi-supervised text classification using the keras library to build a Sequential model in jupyter notebook.

The LSTM Model was implemented for both **Supervised** and **Semi-Supervised** Learning methodology (Li, 2019).

For this model, along with the pre-processing steps implemented earlier additional constraints and features are implemented as mentioned below:

- **Tokenization and vocabulary generation**: A maximum limit of 5000 was applied to the size of vocabulary during tokenizing of words.
- **Text to Integers**: Using the "tokenizer" packages included in keras for pre-processing of texts, the text_to_sequence function encodes the tokenized vocabulary in integer format, which is used for analysis in the model.
- **Transforming Labels into Arrays**: An array using dummies is created to transform the labels associated to reviews in an array format, such that, if label = 1, the corresponding array would be [1,0,0,0,0]; label=2: array = [0,1,0,0,0]; label=3: array = [0,0,1,0,0], etc.
- **Padding Sequences**: pad_sequences() function is used to ensure all sequences in a list have the same length.

5.7 Model 7 | ULMFiT using Fast AI

Multi-label text classification using ULMFiT and FastAI Library in Python

ULMFiT - Universal Language Model Fine Tuning

This method is one of the latest developments in the field of text classification in the hope that we can classify text much better. This concept focuses on understanding language rather than just text or words. ULMFiT makes use of transfer learning to create a Language Model.

ULMFiT proposed by Jeremy Howard and NUI Galway Insight Centre's Sebastian Ruder, is capable of predicting next word in the sentence based on unsupervised learning of the WikiText 103 corpus. It uses multiple LSTM layers, with dropout applied to every layer which the secret behind this method. This was developed by Steve Merity (Salesforce) as the AWD-LSTM architecture.

Implementation:

- Step 1: Importing the libraries
- Step 2: Reading the dataset and splitting it into training and validation datasets.
- Step 3: Preparing and pre-processing the dataset by creating DataBunch. TextLMDataBunch for text and TextClasDataBunch for labels.
- Step 4: Creating language Model.
- Step 5: Finding the Optimal Learning Rate to Train the Language Model.
- Step 6: Training the model with the Optimal Learning Rate.
- Step 7: Using the Language Model to train the Classifier.
- Step 8: Analysing the Results
- Step 9: Predicting the labels of the test data

6 Model Differences

The models used for the purpose of this challenge are listed below with their corresponding advantages and disadvantages as observed in the experiments conducted:

Model	Advantages	Disadvantages
Logistic Regression	Good baseline algorithm, highly interpretable, easy to regularize, and outputs well calibrated predicted probabilities.	Can be outperformed by some complex algorithms, relies highly on the presentation of the data, and is known for being vulnerable to overfitting.
Support Vector Machine	Performs well with high dimensional spaces, and is memory efficient.	SVM falls short in this application, where the dataset contains more noise, and is unsuitable for large datasets. Computationally expensive if the parameters are not tuned appropriately.
Naïve Bayes classifier	Low tendencies of overfitting. It has fast execution, low usage of memory and CPU as it is not computationally intensive. Works well with categorical multi-class classification.	It performed poorly due to its sensitivity towards skewed data, causing incorrect estimates created during training the classifier.
Ensemble	By providing extra degrees of freedom, this method allows solutions which would be difficult to reach through a single methodology, and is not prone to overfitting problems.	Ensemble requires high computational methods and can be a time consuming and memory expensive method.
LSTM Model	Nonlinear time series model, which defines nonlinearity based on the data, and is efficient for large datasets. Performs well in the handling noise in dataset.	It is not hardware friendly and consumes a lot of computational time and memory resources.
ULMFiT using FastAI	The algorithm understands the data language creating a more intuitive learning process, and providing higher accuracy for sentiment analysis.	Requires large amounts of computational time.

Figure 1:Model differences

7 Experimental Setups

The accuracy of different models was calculated using:

$$Accuracy = \frac{Number\ of\ correct\ predictions}{Number\ of\ all\ predictions}$$

7.1 Logistic Regression

Logistic regression is improved by tuning the regularization parameter C.

$$C = 1/\lambda$$

Lambda (ƛ) controls the tradeoff between the model complexity and model simplicity i.e. model overfitting and underfitting. If ƛ is low or 0, complex models will be created (overfit) by assigning large weight values for the parameters. Otherwise, by increasing the value of ƛ, simple models will be created (underfit).

Parameter C is inversely proportional to ƛ, Thus, for small values of C, regularization strength is improved leading to underfitting while big values lead to overfitting. Best value of C is the trade-off between the two.

Regularization is done with C values as [0.01, 0.1, 0.2, 0.5, 1, 10].

- multi_class='multinomial'
- solver='newton-cg': compatible with multi-class classification
- C=[0.01, 0.1, 0.2, 0.5, 1, 10]
- class_weight="balanced"
- fit_intercept=False,
- max_iter=250
- penalty='l2': Used in case of multi-class classification
- tol=0.0001: Tuned tolerance
- random_state = 5
- Training-validation split= 75-25

7.2 Support Vector Machine

Regularization is done using values of C. For larger values of C, the optimisation chooses a smaller-margin hyperplane if the hyperplane is able to classify the training points correctly. Otherwise, for lower values of C, optimiser looks for larger-margin hyperplane even if it misclassified more points.

Regularization is done using the same values of C as before i.e. [0.01, 0.1, 0.2, 0.5, 1, 10].

- kernel = [‘linear’, ‘rbf’]
- gamma = [‘0’, ‘10’]
- C=[0.01, 0.1, 0.2, 0.5, 1, 10]
- Training-validation split= 75-25

7.3 Naïve Bayes

No regularization or parameter tuning is performed.

7.4 Ensembling

The best version of the base models: logistic regression with regularization; SVM with regularization; Naive Bayes classifier were ensembled to create a Voting ensembling model. No regularizations were performed and training-validation split was 75-25.

7.5 Recurrent Neural Network

Model Fitting & Parameters:

The LSTM Model is built using keras Sequential Models, where the following layers are added to regularize the training model

- **Batch Size**: Defines how many samples the model goes through before updating the parameters, and the epoch value determines the number of times the model goes through the entire dataset during the training task**.** bs=128 for semi-supervised and bs=64 for supervised has been used here.
- **Embedding Layer**: This layer converts the pre-processed text into meaningful vectors.
- **SpatialDropout1D Layer**: Instead of a regular dropout, spatial dropout is used to regularize the noise_shape created by the embedded vectors (SpatialDropout1D=0.2).
- **LSTM Model Layer**: LSTM layer contains the units and recurrent dropout factors that regularize the linear transformation of inputs **(**LSTM=100).
- **Dense Layer with Activation**: The Dense layer regularizes the output to the 5 sentiment polarity labels, and a “softmax” activation is applied for normalization (Dense layer=5).
- **Compile Layer**: This is executed by loss=categorical_crossentropy which ensures that the output is a categorical variable.
- **Early Stopping**: In order to avoid the model from overfitting, an early stopping callback parameter is added to the model, which allows the model to stop training once the “val_loss” parameter stops improving by 0.01% for 3 epochs.
- **Training-validation Split**: We had split the training-validation dataset in 70-30, 80-20 and 90-10 ratio. The training-validation loss computation is inbuilt. Best accuracy was obtained for 90-10 split.

7.6 ULMFiT using FastAI

Regularization of Hyper Parameters:

- **Epochs**: This refers to the number of iterations. Generally larger epochs result with higher accuracy but can also lead to overfitting. The training loss should not be very less than the validation loss. To find the trade-off point is very necessary. For our model, epoch=3(approx. 66% accuracy) gave a better solution than epoch=4 (approx. 65% test accuracy).

- **Training and Test dataset size**: More the training data better the accuracy. But, the trade-off points between the two should also be considered which is quite important. On performing a 70-30 split, the validation and the test accuracy was almost the same. On performing 80-20 split, the validation accuracy was a little larger than the test accuracy. On performing a 90-10 split with epoch=3 gave the best accuracy which is 66.4% on the test data.
- **Learning Rate** - If the learning rate is too small then overfitting can occur. When starting with a small learning rate, the neural network begins to converge and, as the learning rate increases, it eventually becomes too large and causes the test/validation loss to increase and the accuracy to decrease. Hence, it is very important to find a minimum/constant learning rate so that the network will not begin to converge. Optimal learning rate was found using minimum gradient.
- **Batch Size(bs)**: Batch size should be used according to the number of GPU available. We experimented bs=500, bs=126, bs=64, bs=32. Since we used Google Colab to implement our model, we had just one GPU available for us. Hence, implementing larger batch size was extremely slow. Settlement with batch size=32 was done, with which the whole process took a maximum of 5 hours for building the model.
- **Momentum**: It is better to use a cyclical momentum (CM) that starts at this maximum momentum value and decreases with increasing learning rate to a value of 0.8 or 0.85 (performance is almost independent of the minimum momentum value). Using cyclical momentum along with the LR range test stabilizes the convergence of the network model when using large learning rates more than a constant momentum does. For our model we have used (0.8,0.7) momentum.
- **Dropout**: It is a technique to prevent overfitting by randomly dropping nodes of the hidden layers in a neural network at each phase of training. For our model we have implemented drop_mult=0.3 and 0.5. Earlier we saw that the test error was very large than the training error, leading to very low-test accuracy in spite of getting a great training accuracy. Clearly our model was overfitted. Hence, we tested drop_mult=2, 1.5, 1, 0.5, 0.3. Using large values like 2 and 1.5, our model was underfitted. The best results were attainable by using 0.3 and 0.5 as drop_mult values.

After performing regularization on all parameters and finding the best respective models/classifier, semi-supervised learning (with and without threshold on the prediction probability) has also been performed by sampling a part (5,000 and 10,000 records) of unlabelled data.

8 Experimental Results

*****all the tables show accuracies on the test dataset with respect to a specific model

Model	C = 0.01	C = 0.1	C = 0.2	C = 0.5	C = 1	C = 10
SVM (tf-idf + preprocess)	0.5574	0.5451	0.5326	0.5173	0.5031	0.4858
SVM (CV + preprocess)	0.5584	0.5474	0.5342	0.5183	0.5041	0.4869
LR (tf-idf + preprocess)	0.5254	0.5571	0.5614	0.5567	0.5479	0.5052
LR (CV + preprocess)	0.5645	0.5816	0.5865	0.5781	0.5685	0.5305
LR (pipeline + no preprocess + tf-idf)	0.4876	0.5529	0.5753	0.5977	0.6052	0.613
LR (pipeline + preprocess + tf-idf)	0.4932	0.554	0.5753	0.5993	0.6058	0.6097
LR (pipeline + preprocess + CV)	0.4812	0.5409	0.5631	0.5852	0.5902	0.5986

Figure 2: SVM and Logistic Regression

NB (tf-idf + preprocess)	0.5284	Ensembling (tf-idf)	0.5607
NB (CV + preprocess)	0.5314	Ensembling (CV)	0.5861

Figure 3: Naive Bayes Classifier

LR with semi-supervised (without threshold)	0.6092	LSTM with supervised	0.5627
LR with semi-supervised (with threshold)	0.6136	LSTM with semi-supervised (with threshold)	0.5725

Figure 4: Supervised & Semi-supervised Logistic Regression and LSTM Models

Fast AI with 80-20 train-validation split, epochs=4	0.6554
Fast AI with 90-10 train-validation split, epochs=3	0.6638

Figure 5: Fast AI

Logistic Regression: Logistic regression was also tuned using C values with results in Figure 2. Logistic regression with explicit pre-processing and count vectors performed better with an accuracy of 58.65% for C = 0.2 compared to logistic regression with TF-IDF with accuracy 56.14%.

SVM: The SVM models are tuned using different C values with results shown in Figure 2. The classification accuracies resulting from SVM by using explicit pre-processing and TF-IDF features are listed in the line (1) of Figure 1. On line (2), SVM model that used count vectors with explicit pre-processing performed slightly better than the previous model with accuracy of 55.84% for C = 0.01.

Naive Bayes: As seen in previous cases, in the case of Naive Bayes as well, count vector features performed better than TF-IDF features, although accuracy was only 53.14%.

Ensembling: Ensembled logistic regression, SVM and Naive Bayes didn't outperform logistic regression. It gave an accuracy of 58.61% for count vector features shown in Figure (3). This aligns with our intuition that poor models would not benefit with ensembling.

Logistic regression (Pipeline): Another approach was followed for logistic regression since it performed the best in case of base models. In this case of using pipeline method, features were generated implicitly. Number of features were tuned to 1500. In this case, TF-IDF features gave better results than count vectors shown in Figure 2. This was not intuitive. Also, when raw data was passed instead of the pre-processed data, the logistic model performed the best for C=10 giving an accuracy of 61.3%.

Semi-supervised: Since the best results were for logistic regression, semi-supervised learning technique was used for that model. Results of semi-supervised model on logistic regression with threshold were better (in Figure 4) with an accuracy of 61.36%.

Among the base models, Logistic Regression performed the best with semi-supervised learning.

LSTM:

Supervised Learning -

- Best Parameters: [batch_size=64; epoch = 5; cross-validation=0.1; callback = EarlyStopping parameters]
- The above parameters used in supervised learning provided an accuracy of 56.27% on the test data.

Semi-Supervised Learning-

- Best Parameters: [batch_size=124; epoch = 5; cross-validation=0.1; callback = EarlyStopping parameters]

When the model was trained on the combined pre-processed dataset of labelled data and the predicted unlabelled data, after setting the predicted probability threshold of the unlabelled data to 95%, the accuracy of the test dataset increased to 57.25%.

ULMFit using Fast AI:

Multiple train-validation split approaches was performed including 70-10, 80-20, 90-10 in combination with various batch sizes (bs=16, 32, 64, 128, 256) and epochs (3, 4) and drop_mult values (0.2, 0.3, 0.5, 1, 2). The best and most convenient results (no overfitting or underfitting) was obtained by using 90-10 train-validation split, bs=32, epochs=3 and drop_mult=0.3, 0.5. Also, it is very important to keep a trade-off between learning rate and validation loss which was achieved by finding the minimum gradient. With these parameters, highest accuracy of 67.06% was achieved on the validation dataset and 66.38% on the test dataset, which is a fairly good accuracy.

$$Precision = \frac{Number\ of\ correct\ positive\ predictions}{Number\ of\ all\ positive\ predictions}$$

$$Recall = \frac{Number\ of\ correct\ positive\ predictions}{All\ observations\ in\ actual\ positive\ class}$$

$$F\ score = \frac{2 * (recall * precision)}{reacll + precision}$$

Also, an F score of 0.6707, recall of 0.6717 and precision of 0.6701 was achieved. The precision obtained is quite good as it relates to low false positive rate. The recall obtained is also quite good as it relates to how many positive labels were correctly predicted. In this challenge, the dataset used was balanced i.e. it had almost similar number of labels of each class. Had it not be balanced, F score would be a more useful metric to look upon while comparing the models. Based on the precision and recall, F score obtained is equally good.

9 Conclusion

In order to find the best possible accuracy result on the test dataset, multiple models were trained using various features and parameters.

- Logistic Regression Model — The LR Model performed well upon selecting features through pipelining, however it performed unsatisfactorily when all features taken into consideration, providing the conclusion that the LR Model is highly dependent on the feature extraction and selection in its model training.
- Support Vector Machine — Although SVM is a good base model for high dimensional spaces, such as that involved in the case of text classification, changing parameters of feature selection did not increase accuracy satisfactorily due to the tendency of the SVM Model to handle noise poorly, as well as being unable to handle large datasets.
- Naïve Bayes Classifier — This model performed unsatisfactorily even when subjected to various feature selections, due to the fact that the model is highly subjective towards skewness, causing the test predictions to give low accuracies.
- Ensembling — The idea behind utilising this model was to combine the best performing base models and training them over the dataset, and removing the restrictions of a single best model. This model increased the accuracy but underperformed in the test data, as compared to the Logistic Regression Model.
- Semi-supervised Logistic Regression — From the above models, as Logistic Regression provided the best accuracy on the test dataset, a semi-supervised learning methodology was adapted on it, utilising the unlabeled dataset in the training process, increasing its accuracy by 1%.
- LSTM Model — This model was adapted to implement a nonlinear time series of Recurrent Neural Network approach on the dataset. However, the model performed poorly over the supervised as well as semi-supervised dataset approach and consumed large memory and time chunks, making it an unsatisfactory model implementation.
- ULMFiT using FastAI Library — This model performed the best on text classification when used on the test dataset, as it performs well with large datasets and is not prone to be influenced by the skewness of the datasets. Hence, this was chosen to be our final model and **optimal classifier** with parameters **train-validation split 90-10, epochs=3, drop_mult=0.3, 0.5, bs=32 and optimal learning rate as suggested by minimum gradient**.

Some other approaches that could be implemented further to achieve a much better accuracy:

- Ensembling Logistic Regression with FastAI's ULMFiT Model takin into consideration the average of the predictions with highest probabilities.
- Training the ULMFiT Model on semi-supervised learning methodology.

Some of the approaches implemented for this challenge that did not work:

- Removing duplicate records
- Using larger epochs in neural networks that lead to overfitting
- Using lower dropout values in neural network increased overfitting
- Larger batch sizes in neural networks, large values of C and gamma in LR and SVM lead huge computational time (minimum 10 hours)
- Over or Under Sampling
- Randomly initializing bidirectional AWD-LSTM in FastAI
- Using pre-processed data on RNNs
- Performing semi-supervised without taking probabilities of the predictions into consideration.

Thus, from the implementations and experiments taken up to complete this challenge, the lessons learned was that sentiment analysis is still a major area to be researched on. Not to mention, introduction of neural networks and FastAI greatly enhanced the quality of text classification and multi-class sentiment classification. The models that performed satisfactorily were those which were compatible with large datasets and handled the noise and skewness well, i.e. recurrent neural networks and FastAI.

10 References

Agrawal, A. (2019, February 18). Achieving world-class results using the new fastai library. Retrieved from https://towardsdatascience.com/transfer-learning-using-the-fastai-library-d686b238213e.

Blog, G. (2019, June 25). How to use Multinomial and Ordinal Logistic Regression in R ? Retrieved from https://www.analyticsvidhya.com/blog/2016/02/multinomial-ordinal-logistic-regression/.

Brownlee, J. (2019, August 6). A Gentle Introduction to Dropout for Regularizing Deep Neural Networks. Retrieved from https://machinelearningmastery.com/dropout-for-regularizing-deep-neural-networks/.

Chinchure, A. (2019, July 26). Using FastAI's ULMFiT to make a state-of-the-art multi-label text classifier. Retrieved from https://medium.com/technonerds/using-fastais-ulmfit-to-make-a-state-of-the-art-multi-label-text-classifier-bf54e2943e83.

Fredenslund, K. (2017, June 26). Retrieved from https://www.quora.com/What-is-the-difference-between-TfidfVectorizer-and-CountVectorizer-1

Jain, S. (2019, September 5). Natural Language Processing for Beginners: Using TextBlob. Retrieved from https://www.analyticsvidhya.com/blog/2018/02/natural-language-processing-for-beginners-using-textblob/.

Keras LSTM tutorial - How to easily build a powerful deep learning language model. (2018, February 3). Retrieved from https://adventuresinmachinelearning.com/keras-lstm-tutorial/.

Keras: The Python Deep Learning library. (n.d.). Retrieved from https://keras.io/.

Li, S. (2019, April 10). Multi-Class Text Classification with LSTM. Retrieved from https://towardsdatascience.com/multi-class-text-classification-with-lstm-1590bee1bd17.

Logistic regression. (2019, October 15). Retrieved from https://en.wikipedia.org/wiki/Logistic_regression.

Nabi, J. (2019, February 6). Machine Learning - Text Classification, Language Modelling using fast.ai. Retrieved from https://towardsdatascience.com/machine-learning-text-classification-language-modelling-using-fast-ai-b1b334f2872d.

Pang, B., Lee, L., & Vaithyanathan, S. (2002, July). Thumbs up?: sentiment classification using machine learning techniques. In *Proceedings of the ACL-02 conference on Empirical methods in natural language processing-Volume 10* (pp. 79-86). Association for Computational Linguistics.

Patel, S. (2017, May 4). Chapter 2: SVM (Support Vector Machine) - Theory. Retrieved from https://medium.com/machine-learning-101/chapter-2-svm-support-vector-machine-theory-f0812effc72.

Ray, S., & Business Analytics and Intelligence. (2019, September 3). 6 Easy Steps to Learn Naive Bayes Algorithm (with code in Python). Retrieved from https://www.analyticsvidhya.com/blog/2017/09/naive-bayes-explained/.

Ray, S. (2019, September 3). Understanding Support Vector Machines algorithm (along with code). Retrieved from https://www.analyticsvidhya.com/blog/2017/09/understaing-support-vector-machine-example-code/.

shrikant_1, bbroto06, & rohit30. (2019, February 26). What is the assumption of Logistic Regression. Retrieved from https://discuss.analyticsvidhya.com/t/what-is-the-assumption-of-logistic-regression/77334.

Singh, A. (2019, September 4). A Comprehensive Guide to Ensemble Learning (with Python codes). Retrieved from https://www.analyticsvidhya.com/blog/2018/06/comprehensive-guide-for-ensemble-models/

sklearn.feature_selection.chi2¶. (n.d.). Retrieved from https://scikit-learn.org/stable/modules/generated/sklearn.feature_selection.chi2.html.

Smith, L. N. (2018). A disciplined approach to neural network hyper-parameters: Part 1--learning rate, batch size, momentum, and weight decay. arXiv preprint arXiv:1803.09820.

Soni, D. (2019, July 16). Introduction to Naive Bayes Classification. Retrieved from https://towardsdatascience.com/introduction-to-naive-bayes-classification-4cffabb1ae54.

Support Vector Machines. (n.d.). Retrieved from https://brilliant.org/wiki/support-vector-machines/.

text.data. (n.d.). Retrieved from https://docs.fast.ai/text.data.html#TextLMDataBunch.