**TWEET SENTIMENT EXTRACTION**

**Guided by**: Dr Robin Gras, University of Windsor

**Students**:Sanju Atwal (@atwal3@uwindsor.ca)

Margaret Arulmalar Rebeka Nesaraj(nesarajm@uwindsor.ca)

Abstract— The global transformations under novel innovations has drawn a raising interest in natural language processing and computer research ,along with the availability of large datasets. Internet has been increasingly used as a platform for online learning, exchange and sharing of ideas. Social media is being used by people of different age groups to express public and private opinion across a plethora of subjects. Collecting and analysing opinions expressed on social media like twitter has become vital for researchers. Investigating information present in the Tweets is helpful in classifying the opinions in unstructured data which can be positive, negative or neutral. This in turn is helpful in understanding people’s opinion over an issue or the current trends prevalent. We have implemented solutions using RNN’s, BERT and Machine learning to extract the Tweet data and classify them based on the sentiments. A thorough analysis was done on the performance of the solutions and the evaluation metrics were computed and compared to identify the optimal solution

1. **Introduction**

The advent of internet has changed the way people express their opinions. In this digital era , opinions are expressed in the form of product reviews , blogs or in a social media group. Social media generates huge volumes of data which has the ability to attract customers and influence them .People rely on generated content like reviews for decision making .This project aims in developing models for Tweet sentiment extraction using different Deep Learning techniques and classifying the sentiments under different categories like positive , negative and neutral .A variety of experimental models are implemented, trained and tested using different approaches .Twitter dataset is taken for testing and training. The results are analysed to determine the efficiency of the experimental outcomes of the models

1. **Solution**

RNN’s (Recurrent neural networks), Machine Learning techniques and BERT are used to train the models on Twitter dataset. We have attempted to baseline a simple model using LSTM with Glove vector and then improve the performance to extract the maximum efficiency using (i) RNN architectures: LSTM, SimpleRNN and GRU (ii) Machine learning techniques: Logistic regression, Linear Support Vector Machine and Random Forest classifier (iii) BERT. By far, the performance on BERT exceeded the other solutions.

1. **Dataset Overview**

The Tweet Sentiment Extraction dataset from the Kaggle competition is being used in this project. The phrases in this dataset are extracted from Figure Eight’s Data for Everyone platform. This dataset consists of two files train.csv and test.csv with 27481 rows and 3534 rows respectively. The select phrase present in the dataset determines the notion of the tweet and is helpful in categorising the tweets into positive ,negative and neutral sentences.

Header columns in the dataset (train.csv/test.csv) are as follows:

**textID**: unique id for each row of data

**text**: this contains text data of the tweet sentence .

**sentiment**: this column classifies the sentiment of the text (positive/negative/neutral)

**selected\_text:** contains certain phrases /words from the text that strongly supports the tweet sentiment

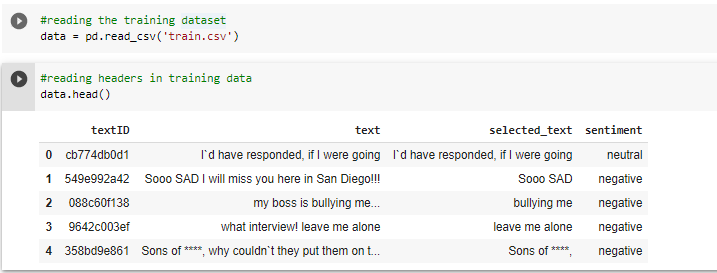
 

Fig 1: Headers in train.csv Fig2: Headers in test.csv

1. **Data Pre-processing**

There are two csv files, train.csv and test.csv, the data from these files is used for training and testing respectively. The text column is the column on which training is done. The text is converted into tokens using various embedding techniques and converted to a format which is accepted by the algorithm. The sentiment column contains three categories which define the sentiment of the text in their respective text column. Using some kind of label encoding these categories are converted into numbers. 0 is assigned to neutral, 1 is assigned to negative and 2 is assigned to positive.

1. **Performance Metrics**

Accuracy and f1 score metrics are used to evaluate the overall performance of the solutions. The F1 score can be interpreted as a weighted average of the precision and recall, F1 score reaches its best value at 1 and worst score at 0. It is calculated by the formula F1 = 2 \* (precision \* recall) / (precision + recall). The F1 score is less impacted even when there is an imbalance in the classes. Accuracy is the proportion of true results among the total number of cases examined and is mainly used in classification problems. Accuracy is determined using the formula Accuracy (TP+TN)/(TP+FP+FN+TN)

1. **Recurrent Neural Network techniques**

**RNN(Recurrent neural networks) :**

RNN is a type of Artificial neural network which can be used for building a powerful model sequence, which solves natural language or speech processing tasks. This model works extensively with sequential or contextual data and gathers text information in each iteration to predict the upcoming scenario.

**Simple RNN (Recurrent neural networks):**

Simple RNN architecture (subset of RNN) from Keras library multiplies Input and the output of the previous layers and passes it through Tanh activation function. No gates are present in this architecture.

**ARCHITECTURE**

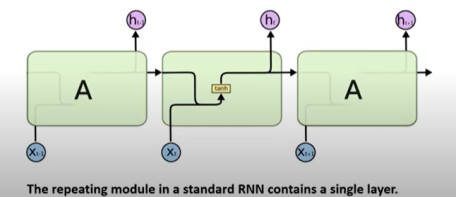


Fig 3:RNN Architecture

**RNN** comprises of an input X, token t, activation A,hidden output h, Output y, time t, tanh (activation). The current layer is fed with output of the previous layer and input of the current layer which then passed through the activation function tanh, to produce the output of the current layer.

**Pros**

(i)They are capable of processing inputs of any length.

(ii) The model size remains the same irrespective of the inputs.

(iii)Shared weights across the time steps.

(iv)Internal memory present.

**Cons**

(i)RNN’s leads to vanishing gradient and explosion problems while learning long term dependencies due to the usage of back propagation over time.

(ii)Training can be difficult.

(iii)Naïve RNN forgets old information when the number of tokens is more.

**LSTM (LONG SHORT TERM MEMORY)**

LSTMs are explicitly designed to avoid the long-term dependency problem. Recurrent Neural Networks suffer from short-term memory. While trying to process a paragraph of text for predictions, RNN’s might omit the vital information from the beginning. During back propagation, recurrent neural networks suffer from the vanishing gradient problem. LSTM’s overcame these challenges with the use of internal mechanisms called gates, which regulates the flow of information. The working of LSTM’s is related to RNN’s but there are some differences in way the operations occur in LSTM’s cells.

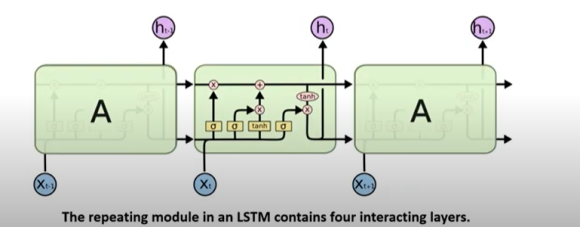
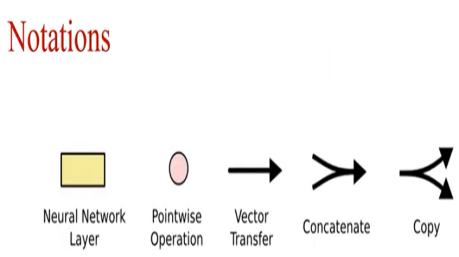
 

Fig 4: Lstm architecture Fig 5: Notations used

**Architecture**

The components of LSTM are input X, token t, activation A, hidden output h, Output y, Current time t, previous time t-1, Sigmoid(activation), tanh(activation), pointwise multiplication and vector addition. The key to LSTM is the cell state represented by the horizontal line at the top of the diagram .This cell state acts as long term memory. (Refer Fig 4 and 5)

**Sigmoid activation**

The sigmoid activation function is also known as logistic activation is mainly used in models where the probability prediction is done .This activation function maps the real values into a small range which lies between 0 and 1 or -1 and 1 from which the probability is defined.(Refer Fig:6)

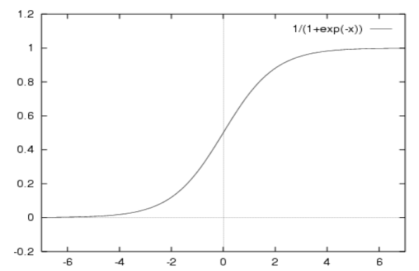


Fig 6: Sigmoid Function

**Tanh activation**

Tanh activation function is mainly used for classification in feed-forward networks. The input value lies in the range -1 to 1 prior training. (Refer Fig :7)

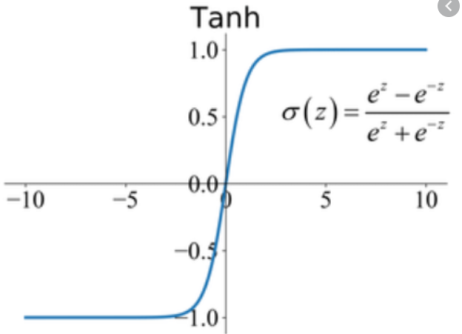


Fig 7: Tanh activation

**Pointwise multiplication/addition**

Element-wise multiplication/addition operations on vectors are known as Pointwise multiplication operation .In this multiplication operation we apply multiplication operator on first item of each vector to get the first item of the output and so on.

**Vector concatenation:**

This is used for concatenation of output from the different activation functions or binary operations.

**Forget gate**

This gate decides what information should be thrown away or kept. Information from the prior hidden state and information from the current input is passed through the sigmoid function. The values closer to 0 means to forget and the closer to 1 means to retain.

**Input gate**

The function of Input Gate is to update the cell state. First, the previous hidden state and current input is passed to a sigmoid function that decides which values will be updated by transforming the values to be between 0 and 1. 0 means not important and 1 means important. The hidden state and current input can be fed into the tanh function, which is multiplied with Sigmoid output to select which information is important to keep from the tanh output.

**Cell state**

First, the cell state gets pointwise multiplied by the forget vector. This has a possibility of dropping values in the cell state if it gets multiplied by values near 0.Then; we take the output from the input gate and do a pointwise addition which updates the cell state to new values that the neural network finds relevant. That gives us our new cell state.

**Output gate**

Finally we have the output gate. The output gate decides what the next hidden state should be. The hidden state contains information on previous inputs. The hidden state is also used for predictions.

**Pros**

(i)Faster than simple RNN

(ii)Retains long term information

**GATED RECURRENT UNIT (GRU)**

GRU is a type of RNN similar to LSTM with a forget gate and reduced parameters. It uses hidden state to transfer the required information. Sigmoid, tanh and pointwise functions are similar to LSTM.

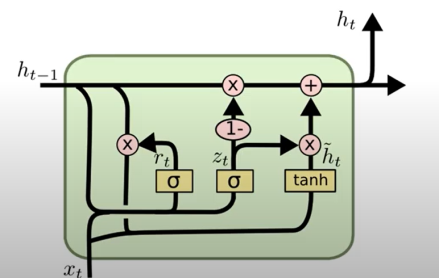


Fig 8: GRU

**ARCHITECTURE**

The components of GRU are input X, token t, activation A, hidden output h, Output y, Current time t, previous time t-1, Sigmoid(activation), tanh(activation) , pointwise multiplication, Vector addition, Relevance gate r, Update gate z, h~ temporary information. (Refer :Fig 8 and 5)

**Update gate**

The function is similar to forget and input gate in LSTM and is useful in retaining the relevant information.

**Relevance gate**

The reset gate determines the amount of past information to forget.

**Pros**

Parameters are less and no internal memory

**Cons**

No non linearity while computing

1. **Baseline model and architecture**

A sequential LSTM model with a combination of dense and dropout layers is used as baseline. A sequential model is one in which there is one input tensor and one output tensor. The model is trained and validated for 40 epochs.

**Architecture**

The baseline model, for Tweet sentiment extraction has the following layers: (Refer : Fig 9 )

(i) An LSTM layer, with shape 64 and 29440 parameters is used

(ii) A dropout layer with 0.5 input

(iii) A dense layer with units =3 and activation=’Softmax’

**Lstm layer**

LSTM layer is used for classifying, processing and making predictions on the sentiments based on the time series data. This layer deals with the vanishing gradient problem and explosion, thereby improving the performance.

**Dropout layer**

Dropout layer is used for regularization. In this layer, every neuron (including the input neurons, but always excluding the output neurons) from the model is switched off randomly at different iterations, to make it more robust. This technique has been implemented to enable the neuron to learn more features and generalise better.

**Dense layer**

Dense layer (Fully connected Neural Network) consists of neurons which are connected to its input layer as well as the output layer. In a fully connected neural network, every neuron in a layer is connected to every neuron in the previous layer as well the following layer. Multiple dense layers have been used in this model.

**Model Compilation:**

The following components are used in compilation of the model,

**Nadam optimizer**

Nadam is an optimization algorithm with improved convergence speed. Our model utilises the momentum technique used in this algorithm to get the information from recent past and apply it on determining the current step.

**Significance:** Increases the efficiency of learning rate of the model

**Sparse Categorical Cross Entropy Loss**

In Sparse categorical cross entropy, uses integer labels to classify the sentiment type

**Significance:** Uses a single integer as class label rather than a vector, hence it saves time and computational memory.

**Accuracy Metrics**

The Accuracy metrics is beneficial in determining the ratio of correct predictions on the balanced dataset. As the MNIST dataset has equal number of sample for each of its class, this metrics is highly beneficial in determining the training versus validation accuracy and loss of the model.

**Significance:** Helps in identifying the true positive and true negative rates with greater accuracy on a balanced dataset

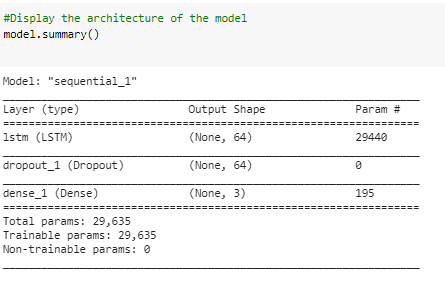


Fig 9 : Baseline model summary

1. **Modifications to baseline**

Experiments were done on the baseline model by replacing the LSTM layer with Simple RNN and GRU. The weights and biases were adjusted to improve the performance of the models. The results were recorded and analysed thoroughly. The overfitting of baseline model was reduced using kernel and bias regularizer in addition bidirectional layers were also used. It is observed that the presence of dense layer is significant to this model as it elevates the performance to a greater extent, however the dropout layer doesn’t impact much. Bidirectional RNN model with GRU cell and LSTM has the highest f1 score and accuracy score of 0.69 , whereas the SimpleRNN exhibits the worst case performance with f1 score and accuracy score of 0.19 and 0.40 respectively . The f1 and accuracy metrics of the other models ranged between 0.65 to 0.67.(Refer : Fig 10)

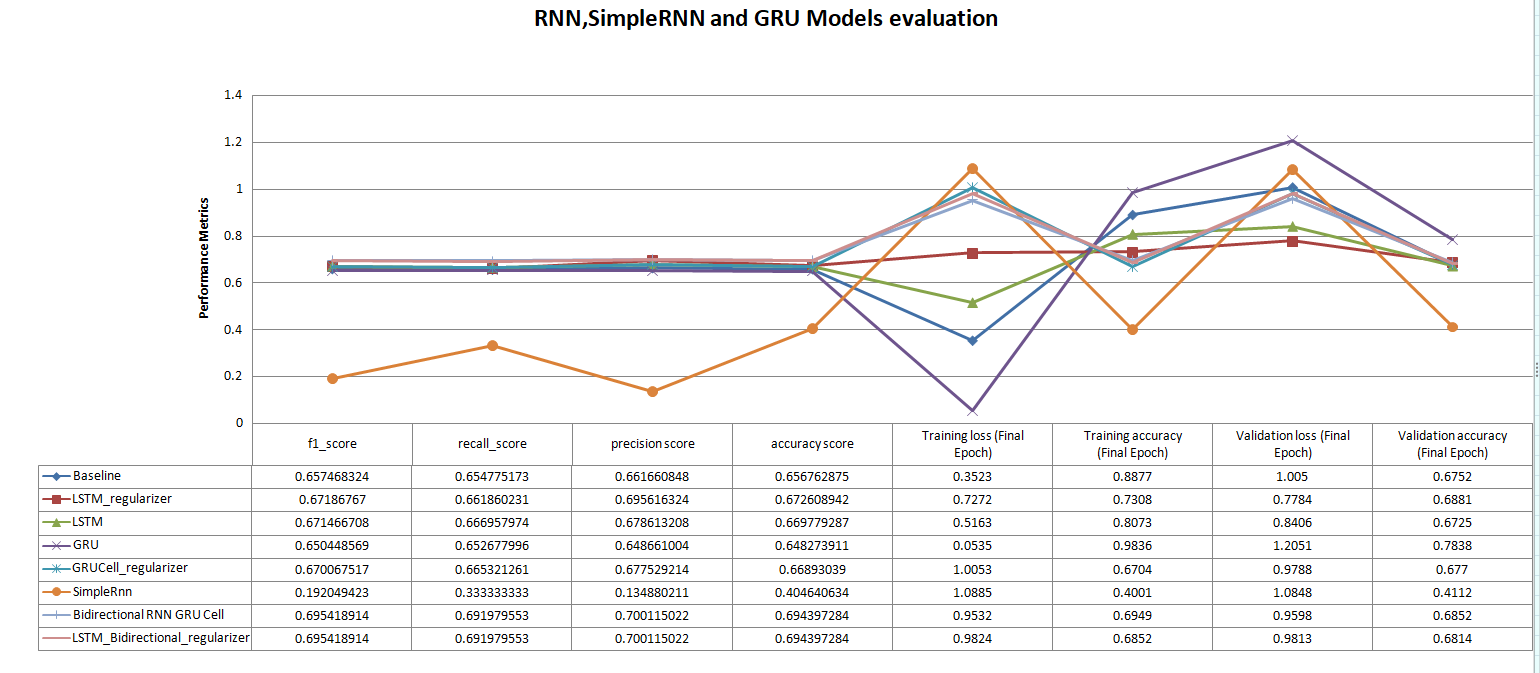


Fig 10 : Graphical representation of Model evaluation

1. **Testing and training process**

The test and training data stored are processed at first before being used in the model .The read data is then split and embedded to covert the high dimensional data into vectors. Then the model is trained and validated for 40 epochs using different adjustments in weights and bias. About 20 % of the training set is taken for validation .The model fitting is then done to evaluate the training and validation accuracy /loss. These values are then plotted on a graph and the trend of the Validation and testing curve is observed. To avoid the overfitting loss, regularizer was used on the kernel and bias level .The deviation in testing loss was reduced after introducing the regularizer. (Refer Fig 11 and 12)

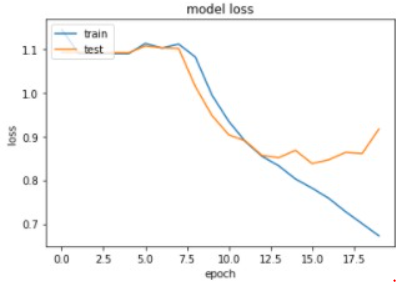
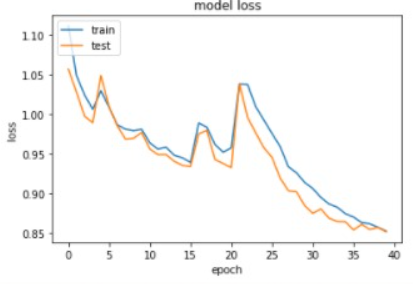
 

Fig 11: Before regularization Fig 12 :After regularization

1. **Machine learning techniques to solve the problem**

Machine learning algorithms like logistic regression, linear SVC and Random Forest classifier were implemented on the existing model. To convert text into tokens we have used the count vectorizer provided by the Scikit learn library. To make sure that the training data and the testing data have the same shape we first combine the training and testing datasets then after converting them into tokens we used the train test split method to convert the data set into training and testing datasets.

The shape of training and testing datasets were (18411, 26439) and (9069, 26439) respectively. We used accuracy and F1 score as our metrics to calculate the performance of these machine learning algorithms.

Out of the three algorithms the random forest classifier performed the worst with the accuracy of about 40% and the F1 score of only 0.23.The linear SVC model scored  65% accuracy and about 0.65 F1 score.The logistic regression got approximately 68 percent accuracy and 0.68 f1 score which was similar to our LSTM model.

1. **BERT Model to solve the problem**

Bert stands for Bidirectional Encoder Representations from transformers.

**Bidirectional**: To understand the text, the model looks in forward as well as backward direction.

**Transformer**: Unlike LSTM’s which read sequentially, the transformer, reads the entire data at once. This allows the model to learn contextual relationships (for example, him as referred to a particular person).

Bert was trained by masking 15 percent of the tokens and the goal of the model was to guess them. Another goal was to predict the next sentence. We are using the ‘Bert-base-uncased' model which is one of the many Bert models provided by Google.Bert-base has a pre-trained stack of 12 transformer encoders.

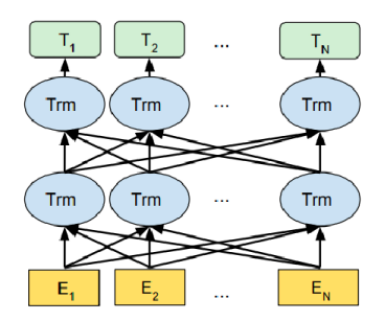


Fig 13 :BERT architecture

A word starts with it’s embedded representation from the embedding layer. At every layer, some multi headed attention computation is done on the word representation from the previous layer. This results in some intermediate representation. All these intermediate representations are of the same size. In the case of Bert-base there are 12 intermediate representations.

For the implementation of BERT, we are using the documentation and coding examples by Hugging Face Library and also using the transformers library provided by them. Trained model is fine tuned according to our needs. We give our text to Bert tokenizer, which gives us the encoded text called token ids along with attention mask. Attention masks consists of array of 0s (pad token) and 1s (real token). Attention mask is of the same length as the token. We have also selected a maximum token length of 256 characters for our model and if two tokens are less than the maximum length the rest of them are padded. We have used the AdamW optimizer care suggested by the Hugging Face documentation. We are using her learning rate of 1e-5.Model is configured for five epochs but we're getting the best accuracy at the second epoch itself with an accuracy and f1 score of about 79 percent.

1. **Results And Efficiency Analysis**

The below table (Table 1)represents the metrics f1 score and accuracy score obtained by using different approaches of Deep learning and machine learning .BERT solution outperforms all the other techniques with a max accuracy and f1 score of 0.79 with SimpleRNN and Random Forest being the least

|  |  |  |
| --- | --- | --- |
| **Model Name** | **f1\_score** | **accuracy score** |
| **LSTM** | 0.69 | 0.69 |
| **SimpleRNN** | 0.19 | 0.40 |
| **GRU** | 0.69 | 0.69 |
| **BERT** | 0.79 | 0.79 |
| **SVM** | 0.65 | 0.65 |
| **Random Forest** | 0.25 | 0.40 |
| **Logistic Regression** | 0.68 | 0.68 |

Table 1:Efficency Anlaysis

1. **Efficiency Improvements and future scope**

In the future, we can improve the performance by, using a larger model of Bert, we could not use it because it was computationally extensive for the time frame of the project, but we can improve this in future. Getting a larger dataset so that, the model has more tokens to learn from. Considering the RNN techniques we can improve their performance by using learning rate and adjusting the hyper parameters yield better outcome

1. **Conclusion**

In this project, the authors have implemented Deep learning and machine learning approaches to provide solution for Tweet sentiment extraction to classify the text sentiments from the tweet data. RNN’s models with glove algorithm were applied to the dataset and Accuracy and F1 score was extracted. BERT and Machine learning techniques were also experimented and comparative analysis was done. The solutions provided extract the sentiment and classify them under the label positive negative or neutral. BERT is by far the best solution implemented as it is more efficient and displays high metric scores.

1. **References**
2. Bert github repository: <https://github.com/google-research/bert>
3. Tensorflow :<https://tfhub.dev/google/bert_uncased_L-12_H-768_A-12/1>
4. Hugging Face github repository: <https://github.com/huggingface/transformers>
5. Tutorial from Hugging Face: <https://huggingface.co/transformers/training.html>
6. LSTM and GRU:<https://towardsdatascience.com/illustrated-guide-to-lstms-and-gru-s-a-step-by-step-explanation-44e9eb85bf21>
7. <https://cs224d.stanford.edu/reports/YuanYe.pdf>
8. <https://machinelearningmastery.com/diagnose-overfitting-underfitting-lstm-models>
9. <https://arxiv.org/ftp/arxiv/papers/1601/1601.06971.pdf>