Stance Classification using Recurrent Neural Network

INTRODUCTION

Sentiment analysis and opinion mining have become a popular research topic in natural language progressing to gain information from an online text and classify text into sentiments or opinions. Stance classification is a subcategory of opinion mining, which aims at automatic detection of the stance expressed in the text towards a specific target. It has widespread applications in information retrieval, text summarization, and textual entailment which is known to employ automated detection systems. Stance detection is related to, but not the same as sentiment analysis. In sentiment analysis, we are interested in whether a piece of text is positive, negative, or neutral based on just the content of the language used. However, the stance of a piece of text is defined with respect to a target topic and can be independent of whether positive or negative language was used.

Our task was to develop a deep learning model to determine whether the tweeter is in favour of a particular target or against it or doesn't have any strong opinion against the target. The given tweets were about five politically charged targets: "Atheism", "the Feminist Movement", "Climate Change is a Real Concern", "Legalization of Abortion", or "Hillary Clinton". The main aim of this project was to build a model which can consume the given the text of a tweet and a target, and in return, classify the tweet as either FAVOUR or AGAINST, or NONE.

LITERATURE REVIEW

Stance classification is a popular research area where many researchers in the field of Machine learning have published their work on similar classification problems. The problem of the stance detection is a supervised learning problem to detect whether a tweet is in favour or against a target which can be solved using multiple approaches. In contrast to the dataset provided in SemEval-2016, (Saif M. Mohammad, 2016) created a tweet-target paired with stance and sentiment containing tweets about the 2016 US Elections featuring Donald Trump and Hillary Clinton. These inputs had an extra feature called target and the stance classes were annotated in "AGAINST", "FAVOR" and "NONE" classes in comparison to the SemEval-2016 dataset which had only 2 classes "AGAINST" and "FAVOR" There were some efforts to extract features from tweets using a SVM classifier by (Saif M. Mohammad, 2016).

Another paper published, where (Guido Zarrella, 2016) used the dataset from SemEval-2016 dataset which contained only 2 labels in their Stance Target variable: "FAVOR" and "AGAINST". Their model achieved the highest weighted average F1-Score of 67.8. They employed a recurrent neural network which was a many to one input network. In their approach, input was converted using one-hot-encoding into a sequence of 30 elements. The sequence of inputs then fed to an architecture that had a 256-dimension embedding layer followed by 128-dimensional LSTM units. The final layer had a Fully Connected Layer of 128 nodes with ReLU Activation and 90% dropout. The unique aspect of their modeling process was that they performed feature training using weak supervision from unlabelled datasets. They employed transfer learning using pre-trained vectors like word2vec and GloVe to improve the performance and reduce the training time. They also used 5-fold cross-validation as a way to tune their hyper-parameters. Their main limitations were that there was still some overfitting left and variation in the F-1 scores across many classes. The majority class had always dominated the minority classes and skipping their embedding layer slightly decreased the weighted average F1 -Score.

Another inspiring work was done by (Gayathri Rajendran, 2018). They demonstrated a detection system to classify the stance against the new headline. Each news headlines was paried with multiple bodies. Basic pre-processing was applied to the datasets such as removing stop words, special characters and tokenization using pandas and numpy libraries. The embedding matrix of 200-dimensional was given as an input which learned from the pre-trained weights of word2vec vector. For their specific problem,

they found word2vec to perform better than GloVe vector. Different models like LSTM, Bi-directional LSTM, GRU, MLP and RNN were employed, where, Bi-directional LSTM units outperformed all the other models by achieving an accuracy of 83.5%.

(Gui, Du, Xu and He, 2020) claims that stance is highly dependent on target and RNNs cannot pay attention to the salient parts of the text. To solve this problem, they applied a novel attention-based mechanism to detect stances from tweets to combine tweets with the target to form a target augmented embedding which is then fed into their RNN model. Their main work was determining which part of text in the tweet was highly dependent on the target. They also evaluated their model on Chinese and English datasets to achieve decent performance of weighted F1 score. Recently, researchers have started to use Transformer networks to gain better results for stance detection (Slovikovskaya, 2020).

DATA EXPLORATION

The source of the dataset given is Semeval-2016 Task 6: Detecting Stance in Tweets (Saif M. Mohammad, 2016). However, the provided dataset was the modified version of the original dataset. The dataset contained a collection of 2814 tweets for training and 1249 tweets for testing the stance detection model. This was a multi-class dataset where stance annotations are expressed using three classes namely 'FAVOUR', 'AGAINST', and 'NONE. Fig. 1 shows a breakdown of tweets pertains to each topic by their stances. Based on the distribution, the stance classification task appeared quite challenging as not only the dataset was small but also it was significantly class imbalanced. For example, the target "Climate change is a concern" has major samples (212 tweets) classified as "favour" and a negligible percentage classified as "against". In contrast to this, target "atheism" has a larger percentage of its samples classified as "against". We also checked the maximum length of a tweet sequence which came out to be of 33 words. It became critical to develop a deep learning model that can produce efficient results with this limited amount of imbalanced training data.

METHODOLOGY

Data Pre-processing

The Tweets can contain a number of arbitrary non-English characters, so we have done pre-processing to some extent on the raw data before feeding it to our model. Almost every social networking platform is recognised for the subject it portrays in the form of hashtags. Particularly, in our case, there could be some strong signals in the hashtags that can be used as an input to the model. However, it was not that straightforward as hashtags usually contain multiple words together (Without spaces) and some of them would not even make sense. Therefore, we first initiated the process by finding the hashtags and mentions present in the tweet data and putting them into another column. This was done to keep the information saved which could later be used to compare the model performance with or without hashtags information. All the hashtags with more than one word had to be segmented into k-words and we did that using ekphrasis library, which has a word information specifically from Twitter. Later we merged the segmented hashtags back to the tweets so that they can also be used for learning. In addition to this, we had one more information that could be merged with the tweets. We merged the targets against their corresponding tweets in a hope that the model will try to learn and build the context around the target value present in the text. Based on our literature review, we came across another approach where (Gui, Du, Xu and He, 2020) used an attention-layer to learn the targets and their relationship against the tweets in a model that had two input layers. The output of these layers was getting concatenated later to build another tensor. However, due to the time constraints, we decided to just concatenate the target values with the tweets.

We also performed a few operations to make the data better for analysis and modeling. We used a prebuild pre-processing library for tweet data available in python which identify and remove text like URLs, (#) hashtags, (@) Mentions, Emojis and Smileys. This library helped us to perform most of the heavy lifting in pre-processing. Despite that, the tweets needed little more attention to deal with the

digits, mixed-case characters and stop words. Therefore, the operations we did in pre-processing were to remove the digits, punctuations and stop words. Words are considered as stop words if they do not add much meaning to the sentence and which can be ignored without losing the meaning of the sentence. Words like “is”, “at”, “which”, “of” etc. Since the stop words can occur in abundance and increase the number of unwanted words in our corpus, it was necessary to remove them from the tweets. We used the NLTK library, a popular toolkit, to remove the stop words from the dataset. This also helped us to reduce the model’s training time as we decreased the significant word from our dataset without losing much unique information.
Next, to deal with words of the same meaning but written in different tenses, Stemming and lemmatization are two popular techniques in NLP. For our task, we used lemmatization, a dictionary-based approach, for converting each word to its base form (Analytics Steps, 2020). For example, the word “Caring” will be converted to “care” as lemmatization can correctly identify the base form of “Caring” word. Another popular approach, stemming, doesn’t retain the meaning of the word and instead just remove the last few characters, which often leads to incorrect or different meanings. For the same example, stemming would have given “Car” as the output which has a different meaning from the original word. Therefore, for us, lemmatization was more suitable and accurate for our project goal.

With the cleaned and lemmatized data, we tried to visualize the word frequency using word cloud for each target value. The most frequent words were from targets that we included in the tweets. If we ignore the targets, it was observed that there was a moderate frequency of words like “SemSt”, “god”,” faith”,” lord” etc.

Building Vocabulary

To create an internal vocabulary, we started by creating a corpus, which is a collection of texts or words, by appending all the words in tweets into a single list. Then, we used fit_on_texts function of the tokenizer Library from Keras on this list to create a vocabulary index based on word frequency in the corpus. Tokenizer’s job was to assign every word a unique integer value where 0 was reserved for padding. The words with lower integer value convey the high frequency words in the corpus. Once we had our internal vocabulary ready, we used another function known as texts_to_sequences of the same library to transform each text in texts to a sequence of integers. In other words, it takes each word in the text and replaces it with its corresponding integer value from the word_index dictionary. At last, we configured the maximum length of a sequence to 50 and used pad_sequences function to ensure that all the sequences in a list have the same length. By default, it uses 0 at the beginning of each sequence for padding.

MODELING

Performance Metrics

To evaluate the classification performance, we used the weighted-average F-score, which is calculated by taking the harmonic mean of precision and recall. Here, precision measures the exactness of a classifier and recall measures the sensitivity of a classifier. The combination of these two, F1-score, has proven (Saif M. Mohammad, 2016) to be a good and acceptable metric for similar classification problems. Another major explanation for using the weighted-average F1-score is that the class imbalance is taken into account. While evaluating our models, we used 'Categorical Cross -entropy' as error metric and weighted F1-score as our success measure.

Model Architecture

Recurrent neural networks are known to use the sequential information for making predictions as these networks have the ability to retain information in temporary memory. However, RNN works well only for short sentences because of the vanishing gradient problems. On the other hand, Long-Term-short-Memory (LSTM) is a variant of RNN which overcome the drawbacks of simple RNN and solve the problem of long-term dependency.

Our base model architecture consists of 4 weight layers. The topmost layer contains an embedding layer where each input token is in a form of a sparse vector. The sequence of these sparse vectors is passed through our internal word vocabulary as the input to the model. The embedding layer is trainable with no initialized weights. This feeds into a Long Short-Term Memory (LSTM) layer with 32 units. We have also included a dropout rate of 0.5 and a recurrent dropout rate of 0.25 in this LSTM layer. The recurrent layer is densely linked to the 64-dimensional fully connected layer with a rectified linear unit (ReLU) as an activation function. At last, there's a 3-dimensional output layer. We have used ADAM as the optimizer for adaptive learning and to make the model computationally efficient.

EXPERIMENTS & HYPER-PARAMETER TUNING

Regularization

Our base model was highly overfitting; therefore, it needed some type of regularization. In the next model we used two types of regularization in the LSTM unit. One of them was the dropout_rate and another was the recurrent_dropout. After an experiment with recurrent dropout for values [0.1,0.25 and 0.3], 0.25 performed the best to control overfitting. When we looked at the work of Team MITRE, who won the twitter stance classification in 2016, we found out that they did use a dropout of 0.9. Therefore, we experimented with [0,0.5, 0.6 and 0.9] dropout rates. The dropout_rate of 0.6 performed the best for us, with no significant improvement with a dropout rate of 0.9. Another reason for not using 0.9 was the depth of our network. It was not as deep as the one used by Team MITRE, causing the model to lose most of the information in the tweets, which was not feasible.

Transfer Learning

Transfer learning is a method that uses the information gained from a fully trained network to train another network using the weights previously learned. Transfer learning has the advantage of reducing a neural network model's training time and can result in lower generalisation errors. In our case, we used transfer learning by mapping our internal vocabulary to the words represented in the pre-trained network. GloVe and word2vec are the two popular pre-trained word vectors that we considered for this task. As we were dealing with the twitter data, we choose a pre-trained 100-dimensional twitter vector containing 27B tokens of almost 1.2M vocabulary as our embedding layer. This pre-trained vector was downloaded from an external source (https://nlp.stanford.edu/projects/glove/). While training, we freeze the embedding layer to avoid training on the pre-trained weights and included few layers at the head of the architecture allowing the model to learn problem specific relationships and context from the input. Fig. 3 and Fig. 4 shows the contrast between the results of two models. We can observe that the model with transfer learning has significantly overcome the problem of overfitting and a slight improvement in the performance too.

Batch Normalization

We also added Batch Normalization in one of our models to normalize the input before the fully connected layer. Adding more dense nodes in the fully connected Layer increases the number of trainable parameters, hence, adding more capacity. Batch normalization sometimes improves the network substantially in such cases.

Holdout & K-Cross Validation

We had a very small training set for our model and split it into training and validation set would have left us with an even smaller dataset. With only 2914 instances distributed across 3 classes was challenging to generate any real conclusion or generalization. Therefore, as an experiment, in addition to Train (80%): validation (20%) hold-out validation, we also tried K-cross validation to build K different models. Using this, we were able to train and validate on all of our data. We made an assumption that if our K-models end up giving a similar performance on different parts of the train data, then, the performance will be similar we train on the complete data at once. Both approaches were then compared using the F1-score on the testing dataset.

RESULTS

We built 7 different models in a process of finding the most efficient and stable one. We improved the performance of our base model by using LSTM memory units of different sizes, different dropout rates, increasing or decreasing model capacity by fitting fully connected layers of different sizes, employing more advanced RNN variants like Bi-directional LSTM and GRU. Our model7 (Bi-Directional LSTM with K-cross validation) performed better than other models by achieving a weighted F1-score of 0.56 on unseen testing data. Bi-Directional LSTM, which is an extension of traditional LSTM trains twice on a single input sequence (forward & backward) to extract additional context from the input. However, other models like model6 (LSTM) and model5 (GRU) also performed equally well with slightly lower F1-score. The results of all the models can be referred from Table1.

INDEPENDENT EVALUATION

To investigate the performance under a real-world setting, an independent dataset was created using some of the real tweets available online for the given targets and some were written by us. We tried to collect at least 5 tweets per target. We managed to create a dataset with 33 tweets in total along with their stance and target. Creating tweets with NONE targets was a challenging task as it was not clear what kind of language or words to be used in the tweets to make it differential. Referring to the existing data was not helping either as doesn't hold any specific pattern for those. Due to our lack of understanding on this, our model suffered a bit while giving predictions and end up mislabelling majorly for NONE class (Fig. 2). Once we had our dataset, the same pre-processing steps were applied to make it compatible with the model's expectations. The criteria to pick the best model for this problem were based on this dataset as it simulates the real word settings more closely. As shown in Table1, there was a drastic change in F1-score for few models like model3, model5 and model7. Model7, despite giving the best performance on testing data, didn't perform well when applied to real world settings. Using this result, we can conclude that model7 is not suitable for generalization. On the other hand, model6 came out to be more stable in generalization by achieving the highest weighted F1-score of 0.49 (Fig. 2) on the independent dataset and 0.55 on the testing dataset.

ULTIMATE JUDGEMENT

We are recommending our model6 as the best model to be used in a real-world setting. We were dealing with a very small dataset that was highly imbalanced and still, achieving a weighted F1-score of 55% on testing and 49% on an independent dataset is quite decent and satisfactory. Referring to (Saif M. Mohammad, 2016), we can say that our model's performance was among the most of the classifiers used for this problem. Transfer learning was a great aid to the model because, despite having less data, provided a powerful shortcut to perform well. Also, 3 fold K-cross validation approach turned out to be better than the hold-out validation and therefore, we recommend that too for the real-world setting. However, we strongly believe that the limitation of this model can be overcome if we feed the model with more data distributed equally against each target class. Employing multi-input model with an attention layer could also help to increase the performance.

APPENDICS

Model Name	RNN Unit	Data Split	Transfer Learning	Weighted F1-score	
				Testing Data	Independent Data
model1	LSTM	Hold-out	No	0.45	0.41
model2	LSTM	Hold-out	No	0.44	0.39
model3	LSTM	hold-out	Yes	0.54	0.32
model4	Bi-directional LSTM	hold-out	Yes	0.53	0.45
model5	GRU	hold-out	Yes	0.53	0.3
model6	LSTM	K-cross (k=3)	Yes	0.55	0.49
model7	Bi-directional LSTM	K-cross (k=3)	Yes	0.56	0.39

Table1. Overview of all the results

Target	Stance	
Atheism	AGAINST	304
	FAVOR	92
	NONE	117
Climate Change is a Real Concern	AGAINST	15
	FAVOR	212
	NONE	168
Feminist Movement	AGAINST	328
	FAVOR	210
	NONE	126
Hillary Clinton	AGAINST	393
	FAVOR	118
	NONE	178
Legalization of Abortion	AGAINST	355
	FAVOR	121
	NONE	177

Fig 1. Train data distribution by target and stance

```
Evaluation on Independent Dataset using model: 6
Confusion Matrix
[[8 3 2]
 [6 7 2]
 [3 1 1]]
Classification Report
              precision    recall  f1-score   support

     AGAINST       0.47      0.62      0.53        13
       FAVOR       0.64      0.47      0.54        15
        NONE       0.20      0.20      0.20         5

    accuracy                           0.48        33
   macro avg       0.44      0.43      0.42        33
weighted avg       0.50      0.48      0.49        33
```

Fig 2. Confusion matrix from model6 on Independent dataset

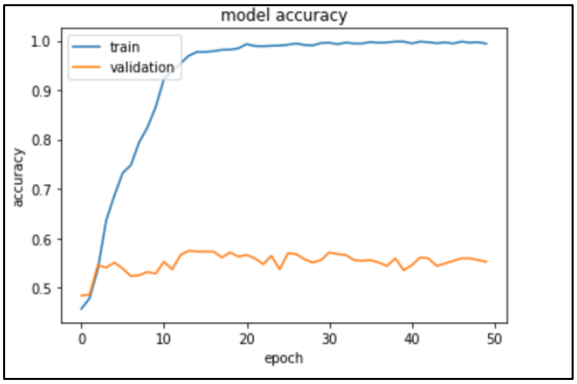

Fig. 3 Accuracy metrics from model2 (No TL)

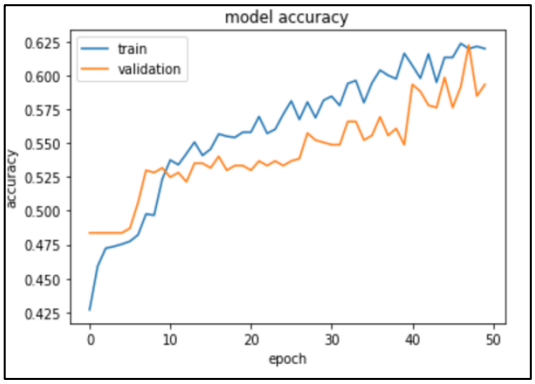

Fig. 4 Accuracy metrics from model3 (TL)

REFERENCES

[1] Saif M. Mohammad, S., Sobhani, P. and Kiritchenko, S., 2017. Stance and Sentiment in Tweets. ACM Transactions on Internet Technology, 17(3), pp.1-23.

[2] Guido Zarrella, A. M. (2016). MITRE at SemEval-2016 Task 6: Transfer Learning for Stance Detection. arXiv:1606.03784v1 [cs.AI]

[3] Rajendran, G., Chitturi, B. and Poornachandran, P., 2018. Stance-In-Depth Deep Neural Approach to Stance Classification. Procedia Computer Science, 132, pp.1646-1653.

[4] Gui, L., Du, J., Xu, R. and He, Y., 2020. Stance Classification with Target-Specific Neural Attention. [online] Doi.org. Available at: <https://doi.org/10.24963/ijcai.2017/557> [Accessed 12 October 2020].

[5] Slovikovskaya, V., 2020. Transfer Learning From Transformers To Fake News Challenge Stance Detection (FNC-1) Task. [online] arXiv.org. Available at: <https://arxiv.org/abs/1910.14353> [Accessed 13 October 2020].

[6] Analyticssteps.com. 2020. What Is Stemming And Lemmatization In NLP? | Analytics Steps. [online] Available at: <https://www.analyticssteps.com/blogs/what-stemming-and-lemmatization-nlp> [Accessed 13 October 2020].