T2-10 Ablation Study: i have a feeling trump will win..... Forecasting Winners and Losers from User Predictions on Twitter

Anand Kamat
McGill ID: 260773313
Email: anand.kamat@mail.mcgill.ca

Dipanjan Dutta
McGill ID: 260779557
Email: dipanjan.dutta@mail.mcgill.ca

Daniel Lutes
McGill ID: 260609221
Email: Daniel.Lutes@mail.mcgill.ca

I. Introduction

People use social media daily on a rampant scale for every event or activity in their lives. They express emotions, opinions and information via social media platforms. It is thus useful to study the tone and emotion attached to a social media post by someone. The emotion and tone conveyed by a person is very useful in understanding the inclination of a person towards an entity. This information is especially useful for predicting winners of contests, elections and sports events. Twitter is the most popular social media platform for dissemination of opinion and emotion. Twitter users use 280 characters to express what they intend to, so usually tweets are precise and on-point. So Twitter statements, or *"Tweets"* can be analyzed to identify the person's emotion and desire towards an entity and event. However, Tweets vary in the extent to which the author wants to express their desire for the event to happen. For example,

In this paper, we explore the model proposed by Swamy, Ritter and Marie-Catherine de Marneffe on "Forecasting Winners and Losers from User Predictions on Twitter", where they used veridicality to annotate a corpus of tweets and and train a log-linear classifier to detect positive veridicality with high precision. Along with exploring the veridicality model, we created a sentiment analysis and Recurrent neural network model to compare the performance to the veridicality classifier. We create the models on the data set used by the authors to keep parity between the performance metrics. We also attempt to implement the veridicality model on a non-contest dataset to evaluate the robustness of the model and how effective it is in a different domain..

II. VERIDICALITY MODEL

We first briefly present the author's implementation of veridicality and the methodology they used. The focus of the paper was specifically on contests which they defined as events in their paper (Swamy et al. 2016). The event was planned to occur on specific date, where a number of contenders compete and a single winner is chosen.

A. Creating a corpus

The first step was to create a corpus of texts or tweets relating to an event. Relevant messages were collected by querying tweets from the Twitter search interface which included the name of the contender for a given contest in conjunction with the keyword *win*. The tweets were all collected before the event in context occurs to keep the opinions of users unbiased. Next, they required the data annotated to compute and compare the performance metrics of their veridicality algorithm. The veridicality annotations on the data was acquired through Amazon Mechanical Turk. Annotators were asked to judge the veridicality of each tweet towards the candidate's winning as expressed in the tweet as well as the author's desire towards the event's outcome. The inclusion was the author's biased proved very effective in limiting the user's bias as they were forced to make the distinction between veridicality and desire and write it down. The following tweet represent varying degrees of veridicality with very little or no desire expressed by th author. The tweets are obtained from Swamy et al. 2016.

- Natalie Portman is gonna beat out Meryl Streep for best actress
- La La Land doesnt have lead actress and actor guaranteed. Natalie Portman will probably (and should) get best actress
- Adored #LALALAND but its #NataliePortman who deserves the best actress #oscar #OscarNoms $\geq$ superb acting

This seems like a effective way to get better annotations. Each text's final annotation was averaged over at least seven author's annotations to reduce the variance and aim to get a more accurate veridicality score.

B. Veridicality Classifier

The approach presented in Swamy et al. 2016 was defined around identifying components of the text and creating features which represent various attributes that relate to the veridicality of the text. The first focus was directed towards identifying the various components of the text which would be used to create the features. These were:

- Target(t) = A target is a named entity that matches a contender name from our queries.
- Opponent (O). For every event, along with the current TARGET entity, they also kept track of other contenders for the same event. If a named entity in the tweet matches with one of other contenders, it is labeled as opponent.
- Entity (e): Any named entity which does not match the list of contenders.

Once the above keywords were identified and labeled, the features can now be processed. The paper Swamy et al.

2016 presented five feature templates: context words, distance between entities, presence of punctuation, dependency paths, and negated keyword. Explaining the relevance and importance of these features are vital to this ablation study as we focus on determining the importance of each feature in learning the veridicality of the tweets. Following are some ways the context of the text was captured by the authors:

- Target and opponent contexts: For every target and opponent entities in the tweet, four words to left and right of them were considered as context words.
- Keyword context: For target and opponent entities, words between these entities and the specific keyword used (like *win*) were also stored.
- Pair context: For the election type of events, in which two target entities are present (contender and state. e.g., Clinton, Ohio), words between these two entities were stored.
- Distance to keyword. The distance of target and opponent entities to the keyword were also computed.
- Punctuation: Presence of question marks and exclamation marks were recorded as well as whether the tweet ends with an question mark, exclamation mark or a period. These help understanding how sure the author of the tweet is regarding his/her claim.
- Dependency paths: Tweeboparser (Kong et al. 2014) was used to retrieve dependency paths between the two target entities and between the target and the keyword (eg. win) after normalizing the paths in the tree (e.g., 'doesn't' becomes 'does not')
- Negated keyword: Tweeboparser's (Kong et al. 2016) normalized dependency paths were used to check whether the keyword was negated.

C. Training and Results

Once the entities in the dataset are labeled and features extracted, the paper talks about training the model on this data. Having a dataset of only recent tweets without including past tweets regarding similar contests/contenders, the model assumes training would not require historical data. In the paper Swamy et al. 2016, the authors modeled the conditional distribution over a tweet's veridicality towards a candidate c winning a contest against a set of opponents, O using a log-linear model. The model is defined as:

$$P(y = v|c, tweet) \propto exp(\theta_v * f(c, O, tweet)) \quad (1)$$

where v is the veridicality (positive, negative or neutral). The algorithm predicts the veridicality scores for each of the tweets. Figure 1 is acquired from Swamy et al., 2016 representing the weights given to the features indicating their relevance and importance towards training.

Once all the tweets have been classified into one of five classes (PY, DY, UC, PN and DN), the next step would be predicting the overall outcome of the contest by poling predictions from all the tweets. The paper Swamy et al. 2016 defines a prediction scoring formula used to predict the outcome of the contest by analyzing the prediction labels given to all the tweets. Equation 2 shows how the final score was calculated:

Positive Veridicality Feature Type	Feature	Weight
Keyword context	TARGET *will* KEYWORD	0.41
Keyword dep. path	TARGET → *to* → KEYWORD	0.38
Keyword dep. path	TARGET ← *is* → *going* → *to* → KEYWORD	0.29
Target context	TARGET *is favored to win*	0.19
Keyword context	TARGET *are going to* KEYWORD	0.15
Target context	TARGET *predicted to win*	0.13
Pair context	TARGET1 *could win* TARGET2	0.13
Distance to keyword	TARGET *closer to* KEYWORD	0.11

Fig. 1. High weight features for positive veridicality

$$score = (|T_c| + 1)/(|T_c| + |T_O| + 2) \quad (2)$$

where $|T_c|$ is the set of tweets mentioning positive veridicality predictions toward candidate c, and $|T_O|$ is the set of all tweets predicting and opponent will win. For each contest the model simply predicts the winner based on the contender with the highest score.

The paper shows how the veridicality classifier as implemented in TwiVer (Swamy et al. 2016) outperforms the sentiment baseline (Mohannad et al. 2013) and the frequency baseline.

III. Algorithms

A. Sentiment Analysis

Sentiment Analysis is the process of determining whether a piece of writing is positive, negative or neutral. Its also known as opinion mining, deriving the opinion or attitude of a speaker. A common use case for this technology is to discover how people feel about a particular topic. It is a way to evaluate written or spoken language to determine if the expression is favorable, unfavorable, or neutral, and to what degree. We can use any classification algorithm for sentiment analysis. For our ablation study, we used Random forest classifier and Fully connected Neural Networks.

1) Support Vector Machines: The support vector machines (SVM) (Boser et al., 1992; Cortes and Vapnik, 1995) require the solution of the following optimization problem:

$$\begin{gathered} \min_{\mathbf{w},b,\boldsymbol{\xi}} \ \frac{\mathbf{1}}{\mathbf{2}}\boldsymbol{w^T w} + C\sum_{\boldsymbol{i=1}}^{\boldsymbol{n}} \boldsymbol{\xi_i} \\ \text{subject to } \ \boldsymbol{y_i(w^T \phi(x_i) + b) \geq 1 - \xi_i}, \\ \xi_i \geq 0 \end{gathered} \quad (3)$$

Support Vector Machines find a linear separating hyperplane with the maximal margin in the higher dimension space. C is the cost or penalty term for misclassification. The kernel function $K(x_i, y_i) = \phi(x_i)^T \phi(x_j)$ is used maps the input to a different dimension. The two types of kernel functions we used are :

- Linear kernel : $K(x_i, x_j) = x_i^T x_j$

- Radial Basis Function (RBF) kernel : $\exp(-\gamma \left\| x_i - x_j \right\|^2), \gamma > 0$

2) Random Forests: Random Forest was developed as an ensemble approach based on many decision trees. Random Forest uses the Majority Vote method and returns the class with most votes. Random Forest uses the Bagging approach in building classification models. For a dataset, D, with N instances and A attributes, the general procedure to build a Random Forest ensemble classifier is as follows. For each time of building a candidate Decision Tree, a subset of the dataset D, d, is sampled with replacement as the training dataset. In each decision tree, for each node a random subset of the attributes A, a, is selected as the candidate attributes to split the node. By building K Decision Trees in this way, a Random Forest classifier is built. In classification procedure, each Decision Tree in the Random Forest classifiers classifies an instance and the Random Forest classifier assigns it to the class with most votes from the individual Decision Trees.

3) Feedforward Neural Networks: Artificial Neural Network (ANN) is inspired by biological neuron has the capability of recognizing patterns effectively. Among all the neural networks, feed forward neural networks with back propagation has the highest rate of convergence. The neural network model can be described as a series of functional transformation. Activations are defined as:

$$a_j = \sum_{i=1}^{D} w_{ji}^{(1)} x_i + w_{j0}^{(1)} \tag{4}$$

where j = 1, 2, ...,M and the superscript (1) indicates the corresponding parameters are in the first layer of the network. Each of the activation is then transformed using a differential, non-linear activation function h(.) to give

$$z_j = h(a_j) \tag{5}$$

These quantities correspond to the outputs of the basis functions, that in the context of neural networks are called hidden units. The activation functions mainly used are sigmoid, tanh and relu. Example of the sigmoid activation function is shown:

$$y_k = \sigma(a_k) \tag{6}$$

$$\sigma(a_k) = 1/(1 + \exp(-a)) \tag{7}$$

Backpropagation is a method used to calculate gradient that is needed to calculate the weights used in the network. Backpropagation is very common in deep neural networks. The gradient calculated is usually that of an error function (loss function).

4) Recurrent Neural Networks: Recurrent Neural Networks take the previous output or hidden states as inputs. The composite input at time t has some historical information about the happenings at time $T < t$. RNNs are useful as their intermediate values (state) can store information about past inputs for a time that is not fixed a priori. Figure 3 shows the vanilla RNN architecture.

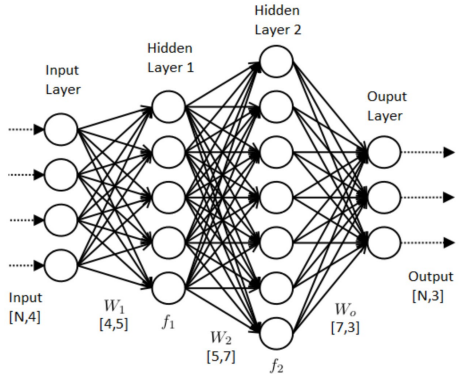

Fig. 2. Feed Forward Neural Net Architecture

5) Long Short Term Memory RNN: The LSTM contains special units called *memory* blocks in the recurrent hidden layer. The memory blocks contain memory cells with self-connections storing the temporal state of the network in addition to special multiplicative units called gates to control the flow of information. Each memory block in the original architecture contained an *input* gate and an *output* gate. The input gate controls the flow of input activations into the memory cell. The output gate controls the output flow of cell activations into the rest of the network. Later, the *forget gate* was added to the memory block. This addressed a weakness of LSTM models preventing them from processing continuous input streams that are not segmented into subsequences. The forget gate scales the internal state of the cell before adding it as input to the cell through the self-recurrent connection of the cell, therefore adaptively forgetting or resetting the cells memory. In addition, the modern LSTM architecture contains *peephole connections* from its internal cells to the gates in the same cell to learn precise timing of the outputs.

An LSTM network computes a mapping from an input sequence $x = (x_1, \ldots, x_T)$ to an output sequence $y = (y_1, \ldots, y_T)$ by calculating the network unit activations using the following equations iteratively from t = 1 to T:

$$i_t = \sigma(W_{ix} x_t + W_{im} m_{t-1} + W_{ic} c_{t-1} + b_i) \tag{8}$$

$$f_t = \sigma(W_{fx} x_t + W_{fm} m_{t-1} + W_{fc} c_{t-1} + b_f) \tag{9}$$

$$c_t = f_t \odot c_{t-1} + i_t \odot g(W_{cx} x_t + W_{cm} m_{t-1} + b_c) \tag{10}$$

$$o_t = \sigma(W_{ox} x_t + W_{om} m_{t-1} + W_{oc} c_t + b_o) \tag{11}$$

$$m_t = o_t \odot h(c_t) \tag{12}$$

$$y_t = \phi(W_{ym} m_t + b_y) \tag{13}$$

where the W terms denote weight matrices (e.g. W_{ix} is the matrix of weights from the input gate to the input), W_{ic}, W_{fc}, W_{oc} are diagonal weight matrices for peephole connections, the b terms denote bias vectors (b_i is the input gate bias vector), is the logistic sigmoid function, and i, f, o and c are respectively the input gate, forget gate, output gate and cell activation vectors, all of which are the same size as the cell output activation vector m, $\odot$ is the element-wise product

of the vectors, g and h are the cell input and cell output activation functions, generally and in our implementation $tanh$, and ϕ is the network output activation function, $tanh$ in our implementation.

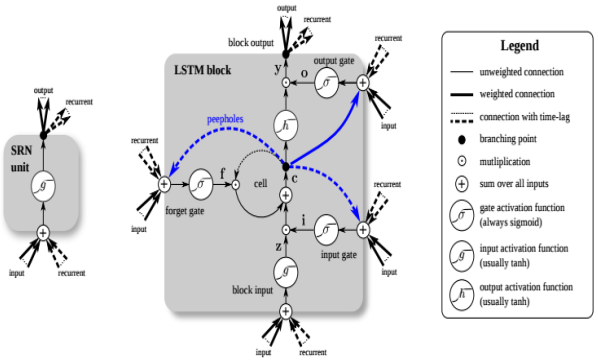

Fig. 3. Detailed schematic of Simple Recurrent Network (SRN) unit (left) and a Long Short-Term Memory block as used in the hidden layers of a Recurrent Neural Network

IV. ABLATION METHODOLOGY

In this section we explore the model defined in Swamy et al., 2016, perform a critical analysis of several points raised in the paper and analyze its performance and efficacy. We have carefully studied every component of their methodology and their application TwiVer (Swamy et al., 2016) and acquired a critical analysis of the same. We discuss several branches of their research such as the corpus, codebase, non-contest datasets, feature selection and ablation, other contest datasets and more. We had access to the original codebase from the github repository of one of the authors. We began with replicating the results described in the paper (Swamy et al., 2016).

A. Corpus

Obtaining an identical dataset was a crucial first step in replicating the results. We were unfortunately not given access to the entire corpus which was used in the paper(Swamy et al., 2016). The files present in the GitHub repository contained the tweet ids for the tweets they acquired and the corresponding labels. Having only access to the tweet ids we were needed to extract the tweets from the Twitter Search API ourselves. The python package *tweepy* eased our efforts with the convenience of direct call methods. While attempting to load the tweets, we were inconvenienced having realized that several of the ids didn't match any tweets. One possible explanation could be that they referenced deleted tweets or were simply erroneous. Due to this, we were able to only acquire 807 tweets instead of the 997 ids provided. Another shortcoming we noticed was that there didn't seem to be enough data (tweets) relating to each event. There were events with only four tweets relating to it. Obviously, the model trains on all the annotated tweets from all events and then makes predictions, but intuitively training the model on multiple tweets all pertaining to one event might seem like a constructive way to increase accuracy. Another shortcoming of the model was seen during the annotation stage. The tweets collected related to events which transpired in the past (e.g. Donald Trump winning the election). However the annotation process was conducted after the results of the contests came out. Knowing the actual results of the contests, like the presidential election, it would naturally follow that annotators would induce a little bias towards the contender who, in the real contest won. For example knowing that Donald Trump would win the election might make a annotator label an potentially uncertain veridical text to be favoring Donald Trump. The logical and intuitive solution which follows would be to test the model on contests which are yet to transpire. This might make the annotations unbiased. After all, the annotations here represent the gold standard.

B. Codebase

We attempted to replicate the results to ensure the code's compliance with the results published in Swamy et al., 2016. We were disappointed to see a haphazard codebase which hindered our process for quite some time. The github repository consisted of about twenty files with no direction and sensible arrangement, and analyzing the worth and use of a file proved to be an exhaustive process. We came across several files which, we presume to be of no use to the application. Another disappointment was the fact that the code files were not commented and seemed like a vat of inexplicable lines of code. It took our team a while to understand the process and replicate the results. There were also certain dependencies associated with the algorithms, such as TweeboParser which didn't provide a direct effective installation and took us a while to get it running. Although the codebase was a huge disappointment, the repository also contained the trained model which could be directly loaded and used to predict the labels. Given this, we successfully managed to replicate the exact results published in the paper.

C. Non-Contest Datasets

One way to test the model's versatility of the model is to directly test it using data from different domain. Veridicality, being analogous to sentiment to some degree should be able to score text from relating to other topics just as directly as it did for contests and events. We explored one just dataset unique to this algorithm. We obtained a dataset from Kaggle containing movie reviews about several movies and the corresponding sentiment associated with it. The dataset seemed quite comparable to different contests and events scored by the veridicality classifier in TwiVer (Swamy et al., 2016). Being an ablation study, we attempted to retain the dynamics of the model and limit our modifications to the TwiVer algorithm to enable compliance with the new dataset. We changed the keyword from *win* to *like* and attempted to train the model on reviews regarding the movie '*The Da Vinci Code*' which became the target. Unfortunately, the model fperforms poorly with only one entity and no contender. Moreover, the model operates on the presence of

a keyword like "*win*". For a non-competitive dataset, there is no clear keyword to operate on, as they are mostly feelings towards one entity and not a preferential keyword between two entities. This seems like a shortcoming of the model. Given having the algorithm designed for a contest, it seems like it could have been generalized to some degree. For example not having a contender mentioned in the text seems like a common sight in tweets regarding events (e.g. "Donal Trump will not win the election"). Hopefully we see the algorithm more inclusive of other domains of data in the future.

D. Feature Ablation

Labeling different entities in the text and creating features was a crucial element of the methodology used in Swamy et al., 2016. The success attributed to this model can be traced to the features created. The features very efficiently capture the relations between the entities and the context. It became imperative to test these features and understand how crucial the feature is to the success of the model. As presented in Table 1, we attempted to reinforce this weighted importance of each feature using the ablation methodology. Our approach involved commenting out lines of code that involve a particular feature in the training process. We then attempt to continue with the normal training and prediction approach as stated in the paper Swamy et al., 2016. We tested two different datasets in this ablation study. We also continue the discussion of the results in subsection E. We notice a big reduction in performance for features such as **Target Context** and **Opponent Context**. This backs the values in table 1, as features given high weights affect and reduce the performance of the model proportionally. We continue this ablation study in subsection E. There, we introduce two datasets involving two other contests and perform feature ablation on the same.

E. Other Contest Datasets

A natural study in ablation would involve how the model performs in conditions not explicitly represented in the paper. We follow this thought by testing the model using two additional datasets each representing contests. These datasets were provided by the authors, Swamy et al., 2016 through their GitHub repository. The reason we chose to work with these datasets is because they have been directly implemented in the paper Swamy et al., 2016. This way, we could avoid surprises in terms of implementation roadblocks.

- Dataset 1: The first dataset represents the 2016 Wisconsin Primary election between Clinton and Sanders. The data contained 100 tweets ids corresponding to tweets related to this election. The dataset however, lacked labels (gold standard) limiting our performance testing
- Dataset 2: The second dataset represents the 2016 Ballon D'or contest between Ronaldo and Messi. The data contained 100 tweets ids corresponding to tweets related to this election similar to the previous dataset. Again, we are deprived of the gold standard labels for this dataset.

Not only did we train the model using the TwiVer algorithm unchanged, we subjected the new datasets to feature ablation discussed in subsection D. Since we didn't have access to labels we come off short with all the performance metrics, but we can still demonstrate the confidence the veridicality inspired in the label. The tables **mention table names here** show the performance in terms of veridicality score for tweets from the above mentioned datasets. The tables show samples of data, the entire dataset has been provided along with the code. The features mentioned in the tables are explained here:

- **Feature 1** = Target Context - Consider the context of the target in the text.
- **Feature 2** = Pair Context - Consider the words between two target entities to gain context.
- **Feature 3** = Opponent Context - Consider the context of the opponent in the text.
- **Feature 4** = Keyword Context - Consider the words between opponent and keyword entities to gain context.
- **Feature 5** = Distance to keyword - Find the distance between the target and the keyword

We notice a very assuring and familiar trend from the Fig1. The absence of a feature given a high weight in the paper Swamy et al., 2016, affects the veridicality score more than a feature given a lower weight. This shows that the weights obtained by training the model seems more or less on point.

F. Performance against other Models

We extended our ablation study to include comparative study of the veridicality algorithm against other models for sentiment analysis. The paper Swamy et al., 2016 states the TwiVer algorithm is one of the best performing sentiment analysis algorithms yet. To validate this claim we move to implement two such machine learning algorithms which have proven to be of common use in this field. We use the same dataset used by the TwiVer algorithm and estimate the same performance metrics (precision and recall) as used by the TwiVer algorithm. The following two approaches were considered and implemented to perform this comparative study. The code for these algorithms are submitted along with the code.

Before passing the text through the model, we preprocessed the data. This involved:

- Obtain the raw text data from the tweet status object returned by tweeny.
- Removal of URLs from the tweet text
- Switching the text to lower case
- Removal of any punctuations
- Removal of hashtags (as most of them dont relate to viable words)

Once preprocessed we needed to tokenize the text. Tokenizing involves creating features required for training using individual words. Sequential models such as RNNs require data to be tokenized before training. We used the tokenizer involved in the keras' preprocessing package. The results obtained from both the classifiers are shown in the Table IV-F:

TABLE I

FEATURE ABLATION ON THE TWEET "MESSI, NOT RONALDO WILL WIN BALLON D'OR ···"

Feature Missing	Predicted Label	Prediction Confidence
None	1	0.78
Target Context	1	0.60
Pair Context	1	0.63
Opponent Context	1	0.72
Opponent Context	1	0.78
Distance to keyword	1	0.76

TABLE II

FEATURE ABLATION ON THE TWEET "TROLL__FOOTBAL LAST PLAYER TO WIN THE BALLON D'OR NOT NAMED MESSI OR RONALDO. CAN'T REALLY ···"

Feature Missing	Predicted Label	Prediction Confidence
None	0	0.73
Target Context	0	0.66
Pair Context	0	0.73
Opponent Context	0	0.72
Keyword Context	0	0.73
Distance to keyword	0	0.70

TABLE III

FEATURE ABLATION ON THE TWEET "CHECK IT OUT WISCONSIN!! YOU GUYS #FEELTHEBERN POLL: SANDERS LEADS CLINTON IN WISCONSIN AHEAD OF MUST-WIN PRIMARY ···"

Feature Missing	Predicted Label	Prediction Confidence
None	1	0.63
Target Context	1	0.47
Pair Context	1	0.63
Opponent Context	1	0.49
Keyword Context	1	0.62
Distance to keyword	1	0.63

TABLE IV

FEATURE ABLATION ON THE TWEET "#BERDIESANDERS: LEADS CLINTON IN NEW WISCONSIN POLL ···"

Feature Missing	Predicted Label	Prediction Confidence
None	0	0.75
Target Context	0	0.57
Pair Context	0	0.74
Opponent Context	0	0.66
Keyword Context	0	0.74
Distance to keyword	0	0.71

TABLE V

PERFORMANCE METRICS FOR VARIOUS CLASSIFICATION MODELS

Classification Model	Precision	Label	F1-Score
TwiVer Veridicality Classifier	80.1%	44.3%	57.0487%
RNN (LSTM)	69.766%	32.077%	37.21044%
Sentiment Analysis (Random Forest)	42.73%	45.69%	44.1604 %
Sentiment Analysis(SVM)	38.94%	51.68%	44.414%
Sentiment Analysis (Neural Network)	42.73%	45.69%	44.1604%

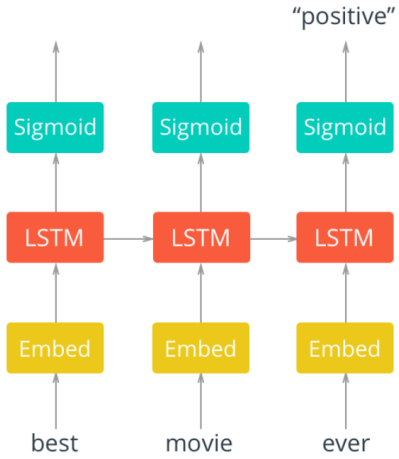

Fig. 4. A schematic of the Recurrent Neural Network Architecture with LSTM used for sentiment analysis

1) Recurrent Neural Network (RNN): Recurrent Neural Networks (RNNs) have been very popular with natural language processing tasks due to its ability to handle sequential data efficiently, keeping the context as a priority. Sentiment analysis is very popular field where RNNs architecture fits right in. Figure IV-F.1 shows how the RNN architecture along with LSTM can be used for sentiment analysis.

Once the text is preprocessed we entered it in the RNN model. The RNN model was configured to handle this data along with adding LSTM details. We used the keras platform to define and train the RNN network. The following were the configurations the RNN network was set to:

- Max features were set to 2000
- Added an embedded layer with input dimension set to 128
- Added a spatial dropout layer with 0.4 of the input units set to drop
- Added an LSTM layer with 196 units
- Batch size was set to 32
- The number of epochs were set to 20

The RNN model performs quite well given that it wasn't provided with any prior context or specifically engineered features like the Twiver algorithm had. The RNN model produced a testing accuracy of 71.233%. The precision achieved by the RNN classifier is 69.766% and recall was 32.077%. The Twiver agorithm surpassed the RNN classifier significantly by having a 80.1% precision and 44.3%.

2) Sentiment Analysis: The classic sentiment analysis algorithm was the other algorithm we used to compare the TwiVer algorithm. The classic sentiment analysis approach was tested on three models, all of which are mentioned in Section III. The models used in this approach are:

- Random Forest Classifier: The random forest classifier was implemented using the sklearn's package with default hyperparameters (besides setting setting max depth to 20). The performance metrics obtained were lacking as compared to the TwiVer with a precision of 42.73% and recall of 45.69%.
- Support Vector Machine (SVM): The support vector machine classifier was also implemented through sklearn with default parameters. The SVM classifiers also fails to compete with the TwiVer algorithm with a precision of 38.94% and recall of 51.68%.
- Fully Connected Feed Forward Neural Network: The neural network was also implemented using sklearn with the following hyperparameter values: 100 hidden layer sizes, tanh activation function, alpha value being 0.0001 with a constant learning rate of 0.0001 and a tolerance of 0.0001. The feed forward neural network demonstrated a precision of 42.73% and recall of 45.69%. TwiVer again seems to have the upper hand in these performance metrics.

As observed the TwiVer algorithm outperforms the RNN classfier as well as the sentiment analysis classifier on all three models. The Model by itself performs exceptionally when it comes to sentiment and veridicality as long as the dataset relates to a contest. This observation can be credited to the feature creation process used by the TwiVer algorithm. The features capture the importance, dependence as well as the context within a text enabling a more efficient training and prediction process.

V. CONCLUSION

The veridicality model performs much better than the LSTM and the Sentiment Analysis model, as claimed by the authors in the paper. The feature selection for the veridicality also seems to be on point as the prediction confidence of a tweet as positive or negative reduces with the removal of one or more features from the model. It outperformed LSTM and Sentiment Analysis model on the original dataset. However, there is an overhead to annotate a significant number of tweets beforehand to train the veridicality classifier. We extensively tuned the models to perform as well as possible in classifying the tweets as positive or negative veridicality but the veridicality classifier does a better job than any other algorithm. Thus, this new approach of veridicality for predicting winners of a contest using Twitter data is indeed an effective one.

VI. DISCUSSION

Even though the veridicality classifier does a really good job of labeling the veridicality of authors from their tweets, there are still some aspects of the classifier that needs to be inspected further.

- **Non-contest datasets** : In a non-competitive setting, where classification is based on a single entity and whether people have a positive or negative outlook to it, veridicality classifier performs poorly. We tried to implement the veridicality classifier on a movie review dataset where people tweet abut their likes or dislikes about a movie and the problem is to predict whether the movie is accepted among the people as a good movie or a bad movie. The veridicality classifier was

unable to label the tweets correctly. The essence of the veridicality classifier is in the feature set that is used to train the classifier. Most of the important features of the veridicality model is based on the presence of an opponent and the distance of the entity from the target and opponent. However, in a non-competitive dataset classification problem, there is no opponent. Hence most of the features are invalidated for the veridicality classifier. Moreover, there is no clear keyword to operate on, as they are mostly feelings towards one entity and not a preferential keyword between two entities. Thus, this is a highly specific classifier that can only be used for predicting contest winners. Implementing veridicality in non-competitive domains requires further research into features that are relevant to that domain.

- **Annotation challenge** : The important aspect of the veridicality classifier is to annotate the tweets for training the classifier. The authors used Amazon Mechanical Turk to annotate 3,543 tweets to train the model. The Turkers were asked to judge veridicality toward a candidate winning as expressed in the tweet as well as the authors desire toward the event. For veridicality, they asked Turkers to rate whether the author believes the event will happen on a 1-5 scale (Definitely Yes, Probably Yes, Definitely No). They also added a question about the authors desire toward the event to make clear the difference between veridicality and desire. Uncertain about the outcome, Probably No, It cost them approximately $1,000 to annotate those tweets. However, we did not have such funding or means to annotate a new dataset and train the veridicality classifier on the new dataset. Thus, we were constrained to use their trained model to predict labels on other datasets which hindered our ablation capability.

- **Reproducibility challenge** : The biggest challenge we faced was trying to reproduce their code. The code is available in the GitHub repository. However, the code present in GitHub was not well organized and did not have a lucid instruction as to the execution of the model. Moreover, the annotated dataset, present in the GitHub repository for training the veridicality classifier, had only the Tweet IDs and the labels. On attempting to extract the tweets and create the dataset, we found out that several of the tweets are dead or deleted by the authors. Also, the annotated tweets were 998 in number only. This severely restricted out ability to explore the components of the model with respect to other datasets as the training was restricted to 998 tweets (with some of them dead).

- **Annotator bias** : The brilliant aspect of using Amazon Mechanical Turk for annotation is that it can be used to label data quickly, easily and at a relatively affordable price. The data is aggregated over several Turkers to reduce bias in annotations. However, there is still a level of bias injected into the annotations, solely because of the fact that these annotations are done after the event. Since the maximum amount of Twitter traffic occurs a few days to a week before an event or contest, it is only possible to get annotations after the event has taken place. Thus, given the fact that the Turkers already know the outcome of the contest. This bias stays in the data, even after aggregating, and cannot be removed by any known means. Thus, even though Mechanical Turk is an efficient way to annotate data, it might not be as accurate as we want it to be.

VII. SUMMARY OF CONTRIBUTIONS

The project was a joint effort involving equal participation and contribution of all three members. Progress in each stage was cross-checked as well as constructively critiqued by other team members. All members were present in all team meetings and participated in a healthy discussion regarding the progress of the project. Daniel was in charge of replicating the results mentioned in the paper Swamy et al. 2016. He also participated in the feature ablation study by implementing the code and generating the results. Anand took the responsibility of implementing the RNN and Sentiment analysis classifier and generating the datasets from Twitter with help from Dipanjan. Dipanjan critiqued the various areas of the paper, Swamy et al. 2016. The report and executive summary writing was taken care of Dipanjan and Anand jointly with generous feedback and proofreading. The presentation was planned and designed by all members of the group and was presented by everyone in the team. In all, the project could be classified as a combined effort from all the team members.

REFERENCES

[1] Swamy, Sandesh and Ritter, Alan and de Marneffe, Marie-Catherine: "i have a feeling trump will win.....": Forecasting Winners and Losers from User Predictions on Twitter, 2017

[2] Andranik Tumasjan, Timm O. Sprenger, Philipp G. Sandner, and Isabell M. Welpe. 2010. Predicting elections with Twitter: What 140 characters reveal about political sentiment. In Proceedings of the Fourth International AAAI Conference on Weblogs and Social Media.

[3] Michael J Paul, Mark Dredze, and David Broniatowski. 2014. Twitter improves influenza forecasting. PLOS Currents Outbreaks, 6.

[4] Lei Shi, Neeraj Agarwal, Ankur Agarwal, Rahul Garg, and Jacob Spoelstra. 2012. Predicting US primary elections with Twitter. In Social Network and Social Media Analysis: Methods, Models and Applications, NIPS

[5] Pedregosa, F. and Varoquaux, G. and Gramfort, A. and Michel, V.and Thirion, B. and Grisel, O. and Blondel, M. and Prettenhofer, P. and Weiss, R. and Dubourg, V. and Vanderplas, J. and Passos, A. and Cournapeau, D. and Brucher, M. and Perrot, M. and Duchesnay, E. , Journal of Machine Learning Research ,2011

[6] https://github.com/tweepy/tweepy

[7] https://github.com/DocNow/twarc

[8] Frdric Branchaud-Charron, Fariz Rahman, Taehoon Lee, Keras Application, https://github.com/keras-team/keras

[9] Sushmita Das, Aleena Swetapadma and Chinmoy Panigrahi *Building Occupancy Detection Using Feed Forward Back-Propagation Neural Networks*, International Conference on Computational Intelligence and Networks, 2017.

[10] https://techblog.viasat.com/using-artificial-neural-networks-to-analyze-trends-of-a-global-aircraft-fleet/

[11] Chih-Wei Hsu, Chih-Chung Chang, and Chih-Jen Lin, *A Practical Guide to Support Vector Classification*, Department of Computer Science, National Taiwan University, Taipei 106, Taiwan
[12] Hasim Sak, Andrew Senior, Francoise Beaufays, *Long Short-Term Memory Recurrent Neural Network Architectures for Large Scale Acoustic Modeling*, Google, USA
[13] Arun Mallya, *Introduction to RNNs*
[14] Yun Wan, Dr. Qigang Gao, *An Ensemble Sentiment Classification System of Twitter Data for Airline Services Analysis*, 2015 IEEE 15th International Conference on Data Mining Workshops
[15] https://deeplearning4j.org/lstm.html
[16] https://towardsdatascience.com/sentiment-analysis-using-rnns-lstm-60871fa6aeba