Classifiying Sincerity Using Machine Learning

Rachana Chittari
Department of Computer Science
California State University, Fullerton
Fullerton, USA
chittari.rachana@csu.fullerton.edu

Doina Bein
Department of Computer Science
California State University, Fullerton
Fullerton, USA
dbein@fullerton.edu

Abhishek Verma
Department of Computer Science
California State University, Northridge
Northridge, USA
abhishek.verma@csun.edu

Marian Sorin Nistor
Department of Computer Science
Universität der Bundeswehr München
Neubiberg, Germany
sorin.nistor@unibw.de

Stefan Pickl
Department of Computer Science
Universität der Bundeswehr München
Neubiberg, Germany
stefan.pickl@unibw.de

***Abstract*— Quora is an online platform that empowers people to learn from each other. On Quora, users can post questions and connect with others who contribute unique insights and quality answers. But as with any other social media or online platform, there is the potential for misuse. A key challenge in maintaining the integrity of such an online platform is to classify and flag negative content. On Quora, the challenge is to identify insincere questions. Insincere questions could be those founded upon false premises, are disparaging, inflammatory, intended to arouse discrimination in any form, or intend to make a statement rather than look for helpful answers. We propose to develop a text classification model that correctly labels questions as sincere or insincere on the Quora platform. For this purpose, we used the Quora Insincere Questions Classification dataset, which is available on Kaggle. We first trained classical machine learning models such as Logistic regression and SVMs to establish a baseline on the performance. However, to leverage the large dataset, we used neural network-based models. We trained several models including standard neural networks, and LSTM based models. The best model that we obtained is a two-layer Bidirectional LSTM network that takes word embeddings as inputs. The classification accuracy and F1-score for this model were 96% and 0.64, respectively.**

Keywords—text classification, Quora dataset, LSTM model.

I. Introduction

In addition to the search engines such as Google and Bing, there are websites known as question forums that one can use to gain knowledge. A question forum is an online discussion site where people can hold discussions in the form of posted messages. They have gained a lot of popularity due to their easy to use and understand methodology. Quora, Stack Overflow, Yahoo Answers are some examples of question forum websites.

Quora is a popular online platform for users to ask simple, personal, professional questions and generally obtain well thought-out answers to [1]. While most of the questions are asked in good faith, there are several instances where people tend to ask questions that are inappropriate or inflammatory. They may be targeted at a specific group of people, intend to make a false statement, or sometimes make no sense. These questions tend to create havoc and threaten the integrity of the platform. Questions like these are termed "insincere" and they deviate from the main purpose of an online forum, which is simply to help users share knowledge.

To ensure the safety of the users and the integrity of the forum, it is thus extremely important to classify, flag and remove such insincere questions before they can cause any significant damage. In the past, Quora made use of human reviewers to label questions as sincere or insincere. However, due to the tremendous growth in the number of users and the number of questions posted daily, it becomes intractable to maintain a human based review system. Thus, it becomes important to develop an automated system to perform this classification task. One viable approach is to use a machine learning based text classification approach. We propose to build a supervised machine learning model that can classify questions as sincere or insincere.

The paper is organized as follows. In Section II we present background work, followed by our proposed system architecture in Section III. Results from applying various machine learning algorithm are presented in Section IV. Concluding remarks and future work are given in Section V.

II. Background Work

Detecting inappropriate and negative content online is a highly relevant problem today. This is relevant not just to Quora, but to all social media platforms and online forums. Examples of classifying negative content include analyzing movie reviews on IMDB, entries on Wikipedia, and tweets on Twitter. These are essentially problems in the domain of natural language processing.

A lot of work has been done using machine learning and deep learning models. The basic idea is that supervised machine learning algorithms can be trained on a labeled set of content to correctly identify negative examples. To this end, several companies have shared large pools of data on platforms such as Kaggle [2], and challenge machine learning researchers to develop state of the art classification models.

In the past, traditional machine learning approaches were used to tackle these classification problems. The important consideration with these approaches is how to represent a sequence of words. Once a good set of features have been extracted, they can be used with almost any classification

model. Typically, representations such as bigrams, n-grams, and bag-of-words, have been combined with classifiers such as Logistic Regression and Support Vector Machines. A few examples of these are the following. Vectorization using bag-of-words was combined with Logistic Regression and Naïve Bayes classifiers for tweet classification in [3], [4]. A Support Vector Machine model was used to classify BBC documents into five categories in [5], and n-grams were combined with SVMs for emergency event detection on social media in [6].

Deep learning [7] is a broad family of machine learning models that use artificial neural networks [8]. The term "deep" refers to the use of multiple layers of neurons in the network. The availability of massive data sets and high-speed computational resources such as Graphics Processing Units have enabled us to train deep neural networks to perform a wide variety of machine learning tasks.

Deep learning models have achieved near human performance in various tasks such as object recognition and speech recognition. They also provide the best results for natural language processing tasks such as machine translation, text classification and sentiment analysis.

One of the important advantages of using neural network-based models for natural language processing is that they can be used to learn word embeddings. A word embedding is a mapping from a word to a real-valued vector that is thought to encode the meaning of the word. While several methods exist to generate such embeddings, deep learning methods have provided some of the most widely used word embeddings, such as GloVe (Global Vectors for Word Representation) [9]. These word embeddings can be thought of a representational layer in a deep learning architecture.

Several types of deep learning architectures have been considered for text classification. These include convolutional neural network models, sequence models such as recurrent neural networks, and attention-based models. [10] provides a comprehensive review of more than 150 deep learning models for text classification that were developed in the last few years. In this project, I have mainly focused on using recurrent neural networks and their variants for insincere question classification.

We used an open dataset available on Kaggle [2]. There are about 1.3 million examples in this dataset. Each example consists of 3 fields: q_id, question_text and target:

q_id: Unique question ID assigned to each question

question_text: a question posted on Quora in the English language

target: takes the values 0 or 1. The values 0 and 1 correspond to sincere and insincere questions, respectively.

Here are a couple of example questions and their target values:

Sincere (0): "How did Quebec nationalists see their province as a nation in the 1960s?"

Insincere (1): "Which babies are more sweeter to their parents? Dark skin babies or light skin babies?"

Many of the insincere questions are discriminatory in nature, inflammatory, politically polarizing, and potentially harmful to raise on the Quora platform. It is thus crucial to correctly identify and weed out these questions.

We analyzed was the distribution of the target values. We found that the Kaggle dataset is highly imbalanced. Only about 6.2 % of all the examples are labeled as insincere. This implies that we should not use classification accuracy as a metric to evaluate our model. For instance, a naïve model that always outputs 0 as its prediction would already achieve a classification accuracy of ~ 93.8%, but importantly it would never be able to correctly identify an insincere question (with target value 1). Therefore, it is important to use metrics such as precision and recall to correctly evaluate the performance of a model on this dataset.

We applied the following preprocessing steps:

- Convert all text into lowercase
- Remove all special characters and non-English characters that appear in the dataset
- Remove punctuation characters
- Remove stop words or common English filler words like "and", "an", "the"
- Split the question strings into word tokens

We utilized the Natural Language Toolkit (NTLK) [11] to perform several of these preprocessing steps. NTLK is an open-source platform for building Python programs to work with natural language data. It provides a corpus of English stop words and tokenization functions that I used to clean my dataset. Alternatively, the Torchtext library of PyTorch can also perform the same set of preprocessing steps. PyTorch is an open-source machine learning framework that can be used to quickly build deep learning models.

The next stage was to extract features from the cleaned dataset that can be used to train the machine learning models. For text classification, we must first convert the text into a vector representation. Word vectorization is the process of converting text into a numerical representation and is an important first step in any Natural Language Processing task.

We have considered the following two approaches for feature extraction:

a) Word count vectors. A word count vector of a question or sentence is a representation that specifies how many times each word in a given vocabulary appears in the sentence. This is essentially a bag-of-words representation, which disregards word order but keeps multiplicity. To implement the word count representation, we first built a dictionary of all the unique words that appear in my dataset. Each question was then converted into a vector of word counts. The length of this vector is equal to the number of words in the dictionary. Consider the following example for illustration. Suppose that a dictionary comprises the following words: {"hello", "who", "how", "are", "you", "me", "doing", "today"}. Let us see what the word count vector for a sample question, "Hello, hello, how are you?" would like. Since the dictionary has 8 words, the sample question is represented by an 8-dimensional vector, with each element corresponding to the count of a particular word in the dictionary. The word count vector for our example would be: (2, 0, 1, 1, 1, 0, 0, 0). While word count vectors are useful in some cases, they do have a few limitations. When the vocabulary is large, the word count vectors are sparse and high dimensional. The Quora dataset, for instance, has about

100,000 unique words. And each question in the dataset typically has only 10-20 words. Further, the word count vectors ignore word order, which is certainly important for assessing the nature of a question.

b) Word embeddings are mappings from words to N-dimensional real-valued vectors such that any two words that are closer in the vector space are similar in meaning [12]. Various methods can be used to generate these mappings, including deep learning techniques. Further, in deep learning models, the embeddings can be thought of a representational layer in the architecture. There are several pre-trained word embeddings that are now available. A few examples are wiki-news-300d-1M, GoogleNews-vectors-negative300, paragram_300_sl999, glove.6B.100D, and glove.840B.300D. For this project we have used the GloVe embeddings. GloVe or Global Vectors for Word Representation [9] is an unsupervised learning algorithm for obtaining vector representations for words. The glove.840B.300D embeddings output a 300-dimensional vector for each word in the vocabulary. To combine the embedding vectors for words in a question, I considered 3 methods:

1. Computing the mean of embeddings vectors of all words in a sentence
2. Concatenation of the embedding vectors of all the words in a sentence
3. Use these embeddings as inputs to the sequence models

The first approach is obviously very lossy. In the second approach, the dimensionality of the feature vector increases with the number of words in the question. We explored using the concatenated features as inputs to a few basic classifiers.

III. Machine Learning Models Applied

To obtain a baseline on the performance we started with Naïve Bayes Classifier, Logistic Regression, and Support Vector Machines.

A Naïve Bayes classifier is a probabilistic classifier based on Bayes theorem, which makes the strong assumption that the features are independent of each other [13]. Given an input feature vector $x = (x_1, x_2, \ldots, x_N)$, we must compute the probability of every class given $\boldsymbol{x}$. A vector x is assigned to the class C_i which has the highest probability.

Logistic regression is a binary classification algorithm, used to model the probability of a certain class given a set of input features. In logistic regression, the probability of the target given input features is computed as follows:

$$P(y = 1|x) = \sigma(w^T x + b)$$

where w is a weight vector, b is a bias term and σ is the sigmoid function given by

$$\sigma(z) = \frac{1}{(1 + e^{-z})}$$

Let us call the prediction $\hat{y} = P(y = 1|x)$. We can then compute the cross-entropy loss function for one example as follows:

$$L(y, \hat{y}) = -y \log(\hat{y}) - (1 - y) \log(1 - \hat{y})$$

The weight vector and bias term are then learnt such that they minimize the average cross-entropy loss over all the training examples. This optimization is typically performed using gradient descent-based methods.

For the logistic regression classifier, we used a concatenation of 300-dimensional Glove word embedding vectors as inputs. We first computed the histogram of the number of words in each question for the entire dataset. Based on this histogram, I chose the maximum number of words, N_{max}, to represent each question. Then, each question was represented by a feature vector of size $300\ N_{max}$. For questions with fewer words than N_{max}, we zero padded the feature vector to ensure they are of the same length for all examples.

A Support Vector Machine (SVM) is a binary classification algorithm that tries to find the separating hyperplane/decision boundary with the maximum margin [14]. The SVM is a linear classifier, but they can also perform non-linear classification by implicitly mapping the input features into higher-dimensional feature spaces. This is known as the kernel trick. We applied SVM for question classification with a concatenation of word embedding vectors as input features.

We implemented the Naïve Bayes, Logistic Regression, and SVM classifiers using the scikit-learn, which is an open-source machine learning library in Python.

One of the crucial limitations of the methods described previously is that they do not explicitly consider the sequence of words in a question. A family of artificial neural networks known as Recurrent Neural Networks (RNNs) can handle sequential inputs and are perfectly suited for the question classification task. A Recurrent Neural Network (RNN) is a class of neural networks with self or recurrent connections between nodes. They are derived from feed-forward neural networks, which have only forward connections between nodes in different layers of a network. The hidden layer in RNNs can be thought of as memory states, which are continuously updated by the inputs. Further, RNNs can be used to handle input sequences of variable lengths. This makes RNNs ideally suited for handling applications with sequential inputs such as machine translation, time-series prediction, speech recognition, and indeed text classification. Next, let us look at how a RNN can be trained.

Backpropagation is a widely used algorithm for training neural networks [16]. In supervised learning, training a model requires us to update its weights or parameters such that they minimize some loss criterion. This requires us to compute the gradient of the loss function with respect to all the model parameters.

Backpropagation is an algorithm for computing the gradient of the loss function with respect to the weights of a feed-forward neural network for a single input-output example. In contrast to a naïve direct computation of the gradient with respect to each individual weight in the network, backpropagation uses the chain rule of calculus to compute the gradients one layer at a time. It iterates backwards from the output layer to avoid redundant computations of intermediate terms. This results in a highly efficient algorithm for computing gradients in a neural network. This efficiency makes it feasible to use gradient methods for training multilayer neural networks.

The backpropagation algorithm was originally designed for feedforward neural networks. But they can be adapted to train recurrent neural network as well. A recurrent neural network can essentially be unfolded in time to obtain a feedforward network with tied weights. This intuition was used to develop the Backpropagation Through Time algorithm (BPTT) [17].

Given an input sequence, BPTT works by unrolling all input timesteps. Each timestep of the unrolled recurrent neural network can be considered as an additional hidden layer, which also receives as input the hidden state from the previous time step. BPTT calculates and accumulates errors across each time step to compute the gradient of the loss function with respect to the network's weights.

BPTT is a useful algorithm for training recurrent neural networks. However, it can become computationally expensive as the number of time steps in the input sequence increases. Further, when the input sequences are long, the number of derivatives required for a single weight update will be high. This can cause the gradients of the weights to become diminishingly small depending on the nature of the activation functions used in the network. This problem is known as the vanishing gradients problem. For some activation functions, we can also observe the exploding gradient problem, which would result in numerical overflow problems.

The vanishing gradient problem is a major drawback of the BPTT algorithm especially for standard recurrent neural networks that use sigmoidal activation functions. The vanishing gradients make learning extremely slow. Another consequence of the vanishing gradient problem is that is does not allow learning of long-range dependencies within input sequences. Learning dependencies among different parts of a sequence is vital for several tasks such as machine translation and text classification.

Variations of BPTT such as Truncated BPTT were developed to solve the vanishing gradient problem. But perhaps the best solution was networks that used gating mechanisms, such as Long Short-Term Memory networks [18].

Long Short-Term Memory (LSTM) is a type of recurrent neural network architecture that uses gating mechanisms to regulate the flow of information through a neuron. A basic LSTM unit is composed of a memory cell c, hidden state h, and input, output, and forget gates that allow the unit to remember values over arbitrary times by regulating the flow of information in and out of the unit.

In theory, standard RNNs could keep track of arbitrarily long-term dependencies in the input sequences. However, the problem with standard RNNs is computational in nature. While training RNNs with BPTT, we run into the issue of vanishing gradients. RNNs using LSTM can alleviate this problem because LSTMs can remember values across several time-steps if necessary. This would allow the gradient to flow unchanged during backpropagation. This makes LSTM based networks well-suited to problems involving data where there can be lags of unknown duration between important events in the input sequence.

There are other types of architectures that use gating mechanisms such as Gated Recurrent Units (GRU) [20] that are based on the same intuition.

Recurrent neural networks are ideally suited for processing sequential inputs. However, in normal RNN architectures the computation at any time step depends only on the inputs up to the current time. However, for several tasks it might also be important to consider input data in future time steps as well.

This is crucial in applications such as natural language processing. In many cases, to understand the context in which a word is used, it is important to know what comes before and after it in the sentence. Consider the following two sentences:

a) She said, "Teddy bears are for sale"

b) She said, "Teddy Roosevelt was the 26th president of the United States"

A recurrent neural network processing the two sentences would perform the exact same computations for the first three words. However, in the first sentence the word "Teddy" refers to a toy, and in the second it refers to a person. This information can be obtained only by using the words that follow "Teddy" in the two sentences. Bidirectional RNNs were developed to address exactly this issue.

A bidirectional RNN is simply an RNN with two groups of neurons in the hidden layer. One group serves to process the input sequence in the forward direction, and the other for the backward direction (see Figure 1). The two groups of neurons are then connected to the same output neurons, thereby conveying relevant information from both directions of processing. Bidirectional RNNs can be trained very similarly to unidirectional RNNs, because the forward and backward states do not directly interact with each other.

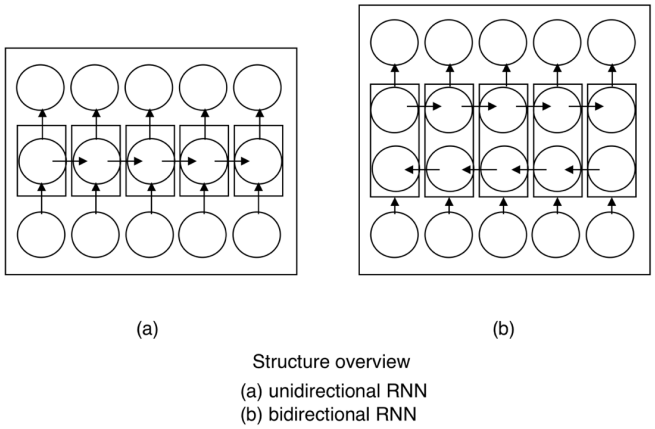

Figure 1. Comparison of a unidirectional vs. bidirectional RNN [21].

Bidirectional RNNs are especially useful when the context of an input is needed like in the Quora insincere question classification task. We decided to use a bidirectional LSTM (Fig. 2) for this classification task. This combines the advantages of both a bidirectional architecture and LSTM units.

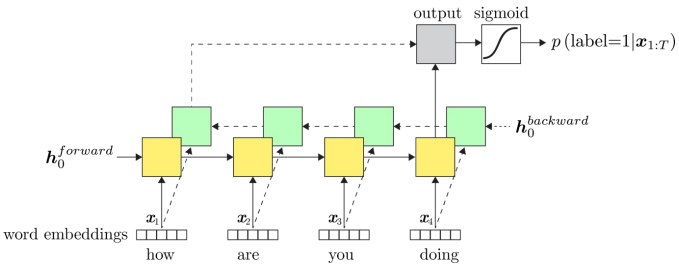

Figure 2. Bidirectional LSTM network architecture

At each time step, the embedding vector for a word in the question text is provided as input to the network. The embeddings can be considered as an additional representational layer in the architecture. These embeddings can be learnt while training the network. Pre-trained embeddings can also be used (i) by keeping them fixed, or (ii) by fine-tuning them during training. The embedding vectors at each time step act as input to the forward and backward hidden states. The hidden states are usually initialized using random values. After the entire sequence has been processed in both directions, the final hidden states in both forward and backward directions are fed as inputs to an output neuron. The activity of this neuron is then passed through a sigmoid function to obtain the probability of the given question being insincere.

Fig. 2 illustrates an architecture with one hidden layer. But this could be generalized to having multiple hidden layers. RNNs with multiple hidden layers are known as stacked RNNs. In a stacked RNN architecture, the hidden states in a given layer are fed as inputs to the hidden layer above it. Increasing the number of layers or depth of the network enhances its representational power, similar to conventional feedforward networks. It is also thought that stacking allows the hidden states at each level to operate at different timescales, and thus allow the network to learn both short-range and long-range dependencies in a sequence [22].

IV. Results of Various Machine Learning Algorithms

We used Google Colaboratory, or "Colab" or short, to develop and train the bidirectional LSTM model. Colab is a product from Google research that allows one to write and execute Python code through the browser. Colab also allows one to use Graphics Processing Units (GPUs). GPUs can be used to train deep learning models much faster than on CPUs.

Since the Quora Insincere Questions dataset is highly imbalanced, the classification accuracy is not a good metric to evaluate a model's performance. A confusion matrix is a better way of summarizing the results. We can use the entries of the confusion matrix to compute three metrics: precision, recall, and F1-score. For our classification task, it is very important to correctly flag all the insincere questions. Therefore, recall is probably the more important metric to consider. When we need a balance between precision and recall, we can use the F1-score.

We used a subset of the full data to train the classical models. The full dataset has roughly 1.3 million examples. Using word count vectors or concatenation of word embeddings results in very high dimensional features. For instance, even for a subset of 200,000 examples, the size of the input matrix of word count vectors for all examples was 200,000 x 85,903. The number of unique words in the dataset is 85,903. As the number of examples used increases, we can expect the number of unique words to grow as well. Such high dimensional features result in memory constraints. Thus, to obtain a feasible implementation as well as a rough lower bound on the performance, I trained the models using small subsets of data.

We used a subset of size 50,000 for Naïve Bayes, and size 20,000 for Logistic Regression and SVM. In each case, we split the data into train and test with a split ratio of 0.8, i.e., 80% of the data was used for training and the remaining 20% of the data was used for evaluating the model performance. Just for the Naïve Bayes classifier, we managed to train the model using a subset of size 200,000.

We implemented the Bidirectional LSTM using PyTorch on Colab. PyTorch is an open-source machine learning platform for building and training deep neural network models. One of the most powerful features of PyTorch is Autograd, which is an automatic differentiation engine that powers neural network training. In PyTorch, we just need to implement the forward pass of the neural network model and provide the loss criterion. The Autograd engine can automatically compute the gradients of the loss function with respect to all the weights in the network. This makes the training of a neural network using backpropagation almost trivial.

The torchtext package in PyTorch consists of data processing utilities for natural language. We used torchtext to preprocess my data and created a PyTorch dataloader.

We used the GloVe.840.300D embeddings to map input words to vectors. This is treated as an embedding layer in the neural network architecture. Our implementation also has the option to fine-tune the embeddings if required. We used cross-entropy as the loss function with the Adam optimizer for training my model. Adam [23], derived from Adaptive Moment Estimation, is a gradient-based optimization algorithm that computes adaptive learning rates for each model parameter. Finally, for training my neural network model, we split the dataset into training and validation sets with a split ratio of 0.9.

We started with 500,000 examples for training the model. We used a 2 hidden layer architecture throughout. For the first pass, we used 64 units in each hidden layer, and the default learning rate of 0.1 for the Adam optimizer.

Figures 3 and 4 illustrate the cross-entropy loss and F1-scores vs. no. of training epochs. One epoch corresponds to iterating through all the mini-batches of the dataset.

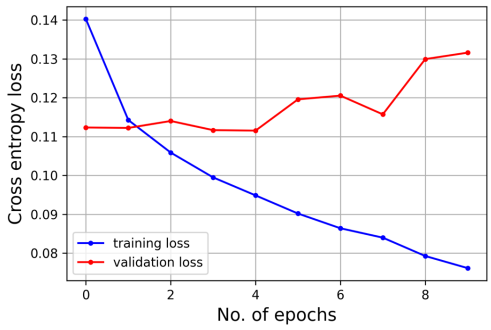

Figure 3. Cross entropy loss with 64 hidden units

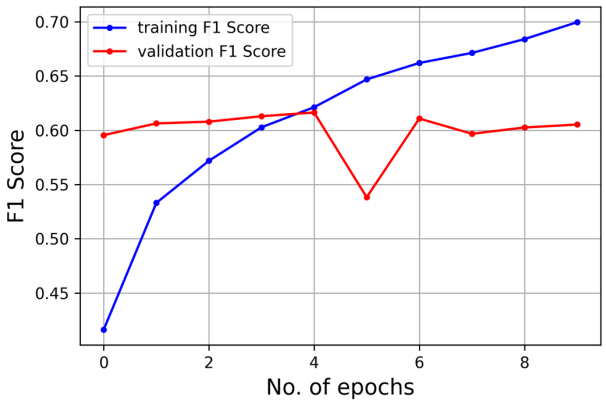

Figure 4. F1-scores with 64 hidden units

In Fig. 3, although the mean training loss (blue) was continuously decreasing with the number of epochs, the validation loss increased after 4 or 5 epochs. This suggested that the model was in the overfitting regime. We obtained similar overfitting effects for different values of the number of hidden units. In order to address overfitting, we plan to use Dropout regularization.

V. Conclusion and Future Work

The neural network model that we used for the Quora Insincere question classification task was a Bidirectional LSTM network with two hidden layers. We obtained better recall and F1 score than the baseline set by the traditional ML methods.

The model most likely benefits from the advantages of the bidirectional architecture and the LSTM. This could be analyzed by comparing the performance of our model with that of a unidirectional architecture.

While the performance of my model is quite good, there are several things that could be done to potentially improve the classification performance. The first of these is to perform a precision-recall analysis to find the best threshold. In classification problems such as these, there is always a trade-off between precision and recall. The precision-recall analysis would allow us to pick a threshold based on the criteria we want to satisfy.

ACKNOWLEDGMENT

This research was sponsored by the NATO Science for Peace and Security Programme under grant SPS MYP G5700.

References

[1] "Quora," [Online]. Available: https://www.quora.com/.

[2] "Kaggle," [Online]. Available: https://www.kaggle.com/..

[3] O. Aborisade and M. Anwar, "Classification for authorship of tweets by comparing logistic regression and naive bayes classifiers.," in IEEE International Conference on Information Reuse and Integration (IRI)., 2018https://www.fidelity.com/learning-center/trading-investing/trading/pairs-trading. (n.d.). From Fidelity.

[4] S. T. Indra, L. Wikarsa and R. Turang, "Using logistic regression method to classify tweets into the selected topics.," in International Conference on Advanced Computer Science and Information Systems (ICACSIS)., 2016.

[5] A. W. Haryanto and E. K. Mawardi., "Influence of word normalization and chi-squared feature selection on support vector machine text classification," in nternational Seminar on Application for Technology of Information andCommunication, 2018.

[6] L. Yanfang, J. Niu, Q. Zhao, J. Lv and S. Ma, "A novel text classification method for emergency event detection onsocial media," in IEEE SmartWorld, Ubiquitous In-telligence Computing, Advanced Trusted Computing, Scal-able Computing Communications, Cloud Big Data Comput-ing, Internet of People and Smart City Innovation, 2018.

[7] "Deep learning," [Online]. Available: https://en.wikipedia.org/wiki/Deep_learning.

[8] "Artificial neural network," [Online]. Available: https://en.wikipedia.org/wiki/Artificial_neural_network.S. Selvarani, S. J. (2014). Automatic Identification and Detection of Altered. International Conference on Intelligent Computing Applications,Coimbatore, 239-243.

[9] J. Pennington, R. Socher and C. D. Manning, "Glove: Global vectors for word representation," in Proceedings of the 2014 conference on empirical methods in natural language processing , 2014.

[10] S. Minaee, N. Kalchbrenner, E. Cambria, N. Nikzad, M. Chenaghlu and J. Gao, "Deep Learning--based Text Classification: A Comprehensive Review.," in ACM Computing Surveys (CSUR), 2021.

[11] S. Bird, E. Klein and E. Loper., Natural language processing with Python: analyzing text with the natural language toolkit., O'Reilly Media, Inc, 2009.

[12] "Word embedding," [Online]. Available: https://en.wikipedia.org/wiki/Word_embedding.

[13] "Naive Bayes Classifier," [Online]. Available: https://en.wikipedia.org/wiki/Naive_Bayes_classifier.

[14] "SVM," [Online]. Available: https://en.wikipedia.org/wiki/Support-vector_machine.

[15] "Unfolded basic recurrent neural network," [Online]. Available: https://en.wikipedia.org/wiki/Recurrent_neural_network#/media/File:Recurrent_neural_network_unfold.svg.

[16] D. E. Rumelhart, G. E. Hinton and R. J. Williams, "Learning representations by back-propagating errors," Nature, vol. 323, no. 6088, 1986.

[17] P. J. Werbos, "Backpropagation through time: what it does and how to do it," Proceedings of the IEEE, vol. 78, no. 10, 1990.

[18] S. Hochreiter and J. Schmidhuber., "Long short-term memory," Neural Computation, vol. 9, no. 8, 1997.

[19] "LSTM," [Online]. Available: https://en.wikipedia.org/wiki/Long_short-term_memory#/media/File:The_LSTM_Cell.svg.

[20] C. K, V. M. B, G. C, B. D, B. F, S. H and B. Y, "Learning phrase representations using RNN encoder-decoder for statistical machine translation," 2014.

[21] "Bidirectional RNNs," [Online]. Available: https://en.wikipedia.org/wiki/Bidirectional_recurrent_neural_networks.

[22] R. Pascanu, C. Gulcehre, K. Cho and Y. Bengio, "How to construct deep recurrent neural networks," 2013.

[23] D. P. Kingma and J. Ba, "Adam: A method for stochastic optimization," 2014.

[24] N. Srivastava, G. Hinton, A. Krizhevsky, I. Sutskever and R. Salakhutdinov, "Dropout: a simple way to prevent neural networks from overfitting," vol. 15, no. 1, 2014.