המכללה האקדמית להנדסה

אורט בראודה

ORT BRAUDE COLLEGE

Software Engineering Department

ORT Braude College

Course 61771 - Extended Project in Software Engineering

Learning to Rank Short Text Pairs with CNN

In Partial Fulfillment of the Requirements for

Final Project in Software Engineering (Course 61771)

Karmiel – June 2020

Or Cohen

Eliran Menashe

Supervisor:

Dr. Renata Avros

CONTENT

1. INTRODUCTION

This project performs text analysis, thus introducing a model for text classification in English written messages through a Convolutional Neural Network. The messages analyzed are short and because there is not enough information to extract, it is difficult to classify them. In this case, the focus is on tweets from Tweeter. To accomplish this task, we use Convolutional Neural Network architecture to compare a pair of short texts by learning their optimal representation using a similarity function so that we can relate them in a supervised way from the training data set. There are only words in the input of the network, and this requires minimal processing. With this model, we can infer which classification the tweet represents.

Bots have become popular on social networks and especially on Twitter. Bots tweets are problematic because they manipulate information, distributing incorrect information, and promote unverified information, which can adversely affect public opinion on various topics, such as political campaigns and product sales. The task of detecting bot on Twitter is complex because many bots are actively trying to avoid detection.

Deep neural networks have shown great promises at capturing salient features for these complex tasks particularly successful for text classification were Convolutional Neural Networks (CNN), on which the project builds upon.

On this project, we describe a novel deep learning architecture for reranking short texts. The main building blocks of the architecture are two distributional sentence models based on convolutional neural networks. These underlying sentence models work in parallel, mapping sentences input to their distributional vectors, which are then used to learn the semantic similarity between them.

The main contributions of this project are the ability to identify bots on Twitter which is a very common issue in social media. We hope that our program will be able to help eradicate the phenomenon of offensive bots on Twitter.

1.1. Organization of the paper

Section 2 - we start by description of a CNN as the basic architecture for text analyzing. Section 3 - we present the model based on architecture for matching text pairs. Section 4 - we briefly discuss the expected result that we want to get. Section 5 - consists of preliminary software engineering documents - use case, class diagram, and initial GUI. Section 6 - consists of all the experiments we have performed on different models and their results.

2. BACKGROUND

2.1. Text Analyzing

Text analyzing task requires doing preprocessing on the data by cleaning and preparing the text for classification or for any analyzing [1]. The original text may consist of a lot of noise and irrelevant parts; hence there is a need to remove or realign them. At the word level, there might be many words that do not make much impact on the overall semantics of the textual context.

Text preprocessing includes a few steps, such as extraction, tokenization, stop words removal, text normalization with stemming and lemmatization. After these steps, the accuracy improves by converting the text to lower case, removing numbers, removing punctuation, removing white spaces and eliminating the sparse terms that are infrequent terms in the document.

2.1.1. Word Embedding

Word embedding [2] is some representation for each word, where words that have similar meanings have a similar representation too. In practice, word embedding includes a process that takes each word and mapping it to a one-hot vector, and by inserting it to neural network it produces a vector consist values that learned during the learning process. The key to the approach is the idea of using a densely distributed representation for each word. Each word is represented as vector of real numbers, often tens or hundreds of dimensions, contrast one-hot encoding representation that requires thousands or millions of dimensions.

2.1.2. Word2Vec

Word2Vec [3] is one of the most popular techniques to learn word embeddings using a shallow neural network, and it is implemented by a two-layer neural network for processing text. The input of the network is a text corpus and the output is a set of feature vectors that represent the words in that corpus.

The purpose and usefulness of Word2vec are to group the vectors of similar words together in vector space and to add the ability to do elementary operations on the vectors that represent the words. That is, it detects similarities mathematically. In addition, it can make a highly accurate guess about the meaning of the word based on past appearances. Those guesses can also establish the association of some words with other words, as demonstrated in Figure 1, or cluster texts and classify them by topic. These representation vectors help us to detect relationships between words. Measuring cosine similarity - no similarity is expressed as a 90-degree angle, while the total similarity of 1 is a 0-degree angle, complete overlap.

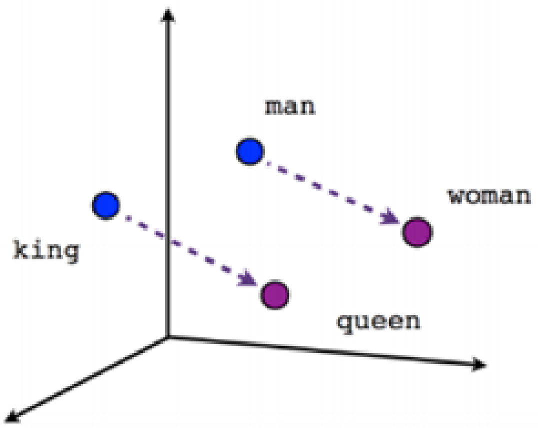


Figure 1 – Example of words association; Male-Female words vectors relations

2.2. Convolutional Neural Networks

An Artificial Neural Network is an information processing model that is inspired by the way biological nervous systems, such as the brain, process information. They are loosely modeled after the neuronal structure of the mammalian cerebral cortex but on much smaller scales. In simpler terms it is a simple mathematical model of the brain which is used to process nonlinear relationships between inputs and outputs in parallel like a human brain does every second.

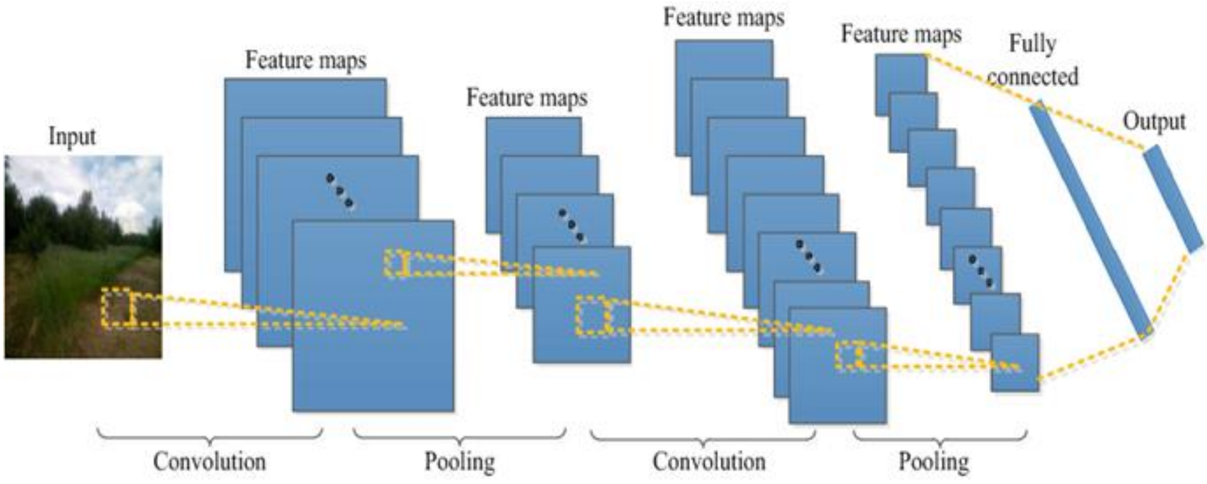


Figure 2 – Convolutional Neural Network for image classification

The Convolutional Neural Network [4] has a different architecture than normal Neural Networks. The input in normal Neural Networks pass through a series of hidden layers. Each layer containing groups of neurons and completely connected to all neurons of the previous layer. The final layer is the output layer that is fully connected and represents the predictions. The Convolutional Neural Network consists of an input layer, hidden layers, and an output layer. CNN hidden layers are usually composed of convolutional layer, an activation layer with some activation function, i.e.: The Rectified Linear Unit (ReLU) function, pooling layer, fully connected layers and normalization layers. CNN networks are mainly used for image classification problems. The input of these networks is a 2D array (for color images the input is 3 RGB matrices). Although CNN is commonly used to classify images, they can also be used for text analysis by representing words as vectors.

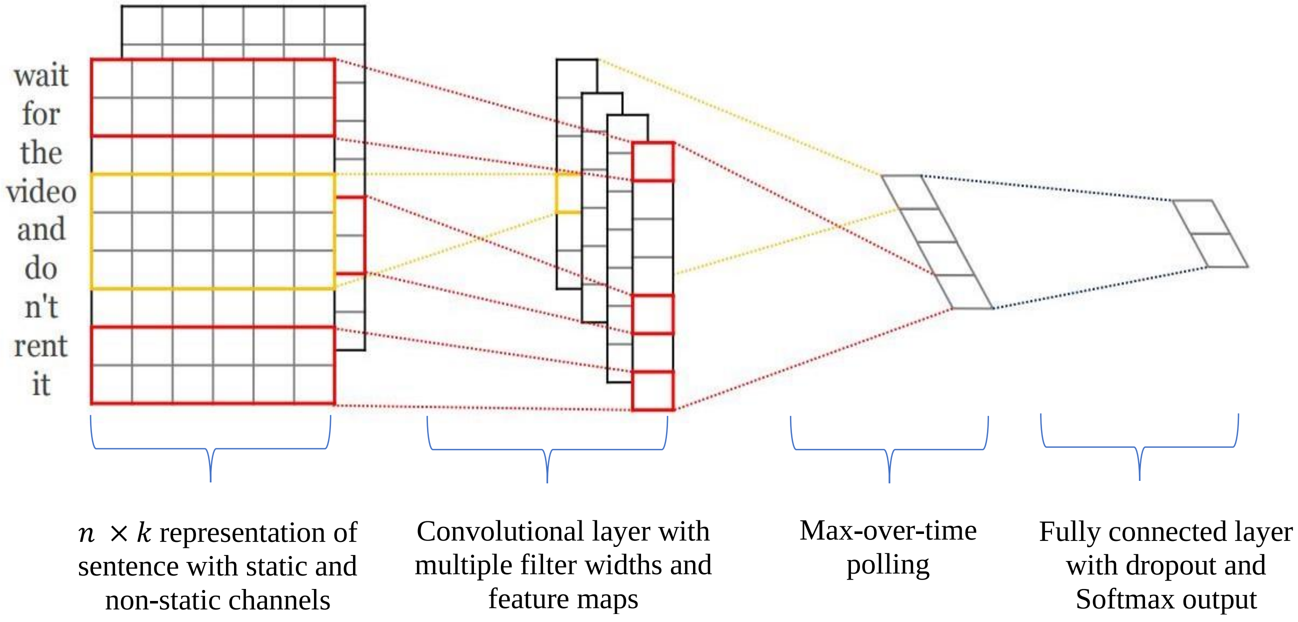

wait
for
the
video
and
do
n't
rent
it
$n\ \times k$ representation of sentence with static and non-static channels
Convolutional layer with multiple filter widths and feature maps
Max-over-time polling
Fully connected layer with dropout and Softmax output

Figure 3 – Convolutional Neural Network for text classification

2.2.1. Convolution feature maps

The convolution layer [5] is the primary layer of the CNN network. This layer is responsible for most calculations and performs the convolution operation on the input and passes the result to the next layer. The purpose of this layer is to extract patterns, i.e., discriminative word sequences found within the input sentences that are common throughout the training instances.

More formally, the convolution operation $*$ between two vectors $s \in \mathbb{R}^{|s|}$ and $f \in \mathbb{R}^{|m|}$, when s is the sentence vector and $|s|$ is the length of the sentence, and f is the filter vector of size m. The result of convolution operation is $c \in \mathbb{R}^{|s|+m-1}$ where each component is as follow:

$$c_i = (s * f)_i = s_{[i-m+1:i]}^{T} \cdot f = \sum_{k=i}^{i+m-1} s_k f_k \qquad (1)$$

The range of allowed values for i in Equation (1) indicates the type of convolution: narrow and wide. In the narrow convolution the window of the filter can move in the range: $[1, |s| - m + 1]$, which requires the filter width to be at most $|s|$. In the wide convolution, the window of the filter can move in the range $[1, |s|]$, so there is no limit on the size of m and s.

The advantages of the wide convolution over the narrow convolution is that it is better able to deal with the words at boundaries and it gives equal attention to all the words in the sentence, while in the narrow convolution, the words that are closer to boundaries are seen fewer times. In addition, the wide convolution always ensures valid values even when $|s|$ is smaller than $|f|$.

At the end of the convolution operation on vector s using filter f, we get a result vector c. In practice, we use more than one filter, so at the same time we perform a convolution operation on vector s using different filters, resulting in a collection of vectors, so each vector is the result of convolution by the different filters and the collection of vectors called feature map.

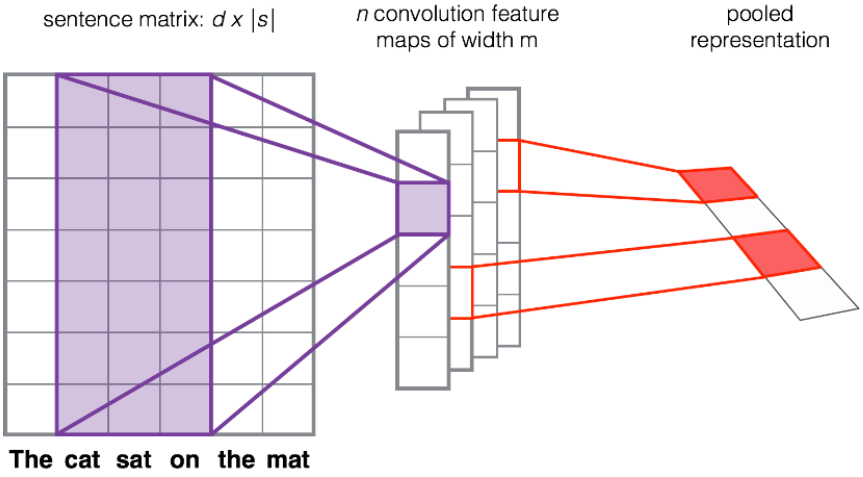

sentence matrix: d x $|s|$
n convolution feature maps of width m
pooled representation
The cat sat on the mat

Figure 4 - Convolutional Neural Network for text classification

2.3. Activation units

Activation units [6] use for learning the network non-linear decision boundaries, after each convolutional layer there is a non-linear activation function $\alpha()$ applied element-wise to the output of the previous layer. There are many choices of activation functions such as sigmoid, hyperbolic, and a rectified linear (ReLU) function defined as $\max(0, x)$ for positive feature maps. The activation function is important for getting the convergence rate and quality of obtained the solution.

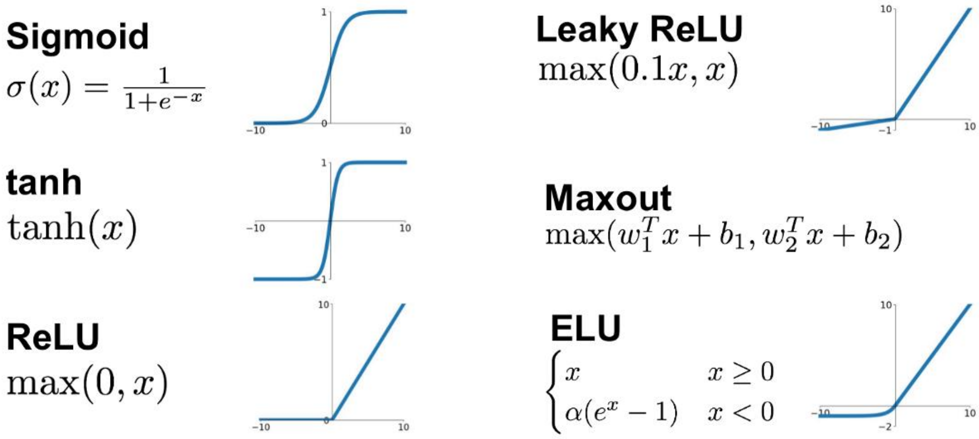


Figure 5 – Examples of Activation Functions

2.4. Pooling

The feature map is the output of the convolutional layer and it forwards to the activation function and then to the pooling layer. Their function is to progressively reduce the spatial size of the representation to reduce the number of parameters and computation in the network, and hence to also control overfitting. The result of the pooling operation is:

$$c_{pooled} = \begin{bmatrix} pool\big(\alpha(c_1 + \ b_1 * e)\big) \\ \dots \\ pool\big(\alpha(c_n + \ b_n * e)\big) \end{bmatrix} \qquad (2)$$

where c_i is a convolutional feature map including bias, which b_i is the bias that added to each element of c_i and e is a unit vector in the same size as c_i, and this is forward through the activation function $\alpha()$.

The pooling operation can be performed by average or by max. Both operations performed to the columns of the feature map matrix and mapping them to a single value, this is also demonstrated in Figure 4.

The extension of max-pooling is k-max pooling, where instead of taking a single max value, we take k values in their original order. This allows for extracting several largest activation values from the input sentence and allows using architectures with several convolutional layers.

2.5. Backpropagation

CNN uses backpropagation [7] as a learning algorithm. The algorithm compares desired outputs to the network actual results outputs, and it helps the network to adjust the connection weights and reduce the difference between the two as much as possible. This process of backwards propagation is repeated several times for each input until CNN's output matches the desired output.

Given a cost function to the CNN, which is an error metric, that gives an indicator of how much precision we lose, if we replace the real output by the actual output generated by trained neural network model, the backpropagation calculates the gradient of the cost function in order to minimize the error calculated by the cost function.

The "backwards" part of the name stems from the fact that calculation of the gradient proceeds backwards through the network, with the gradient of the final layer of weights being calculated first and the gradient of the first layer of weights being calculated last. Partial computations of the gradient from one layer are reused in the computation of the gradient for the previous layer. This backwards flow of the error information allows for efficient computation of the gradient at each layer versus the naive approach of calculating the gradient of each layer separately.

2.6. Softmax

Softmax is the output function of the last layer in neural networks, and that allows us to turn the score produced by the neural network into values that can be interpreted by humans.

The Softmax is a form of logistic regression and it gets as input value from the previous layer that is multiplied by some weights, and passed through an activation function and aggregated into a vector that follows a probability distribution and contains one value per every class of the model whose total sums up to 1. This allows the output to be interpreted directly as a probability. Similarly, Softmax functions are multi-class Sigmoids, meaning they are used in determining the probability of multiple classes at once. It is important to note that a Softmax layer must have the same number of nodes as the output later. It computes the probability distribution over the labels:

$$p(y = j \mid x) = \frac{e^{x^T \theta_j}}{\sum_{k=1}^{K} e^{x^T \theta_k}} \qquad (3)$$

where $\boldsymbol{\theta_j}$ is the sum of the score of each occurring element in the vector, $\boldsymbol{\theta_k}$ is the weight vector of the k-th class. x is the final representation of the input obtained by a series of transformations started in the input layer and then passes through a series of convolutional and pooling operations.

2.7. Hidden layers

The hidden layer's name derived from the fact that they are in between input and output layers, and they are not visible to others in the neural network. In the neural network could be zero or more hidden layers. For most networks, there is no need for more than one hidden layer and in most cases, each hidden layer contains the same number of neurons. The larger the number of hidden layers, the more time it takes for the network to provide the output and to deal with solving complex problems. The hidden layer computes the following transformation:

$$\alpha(w_h \cdot x + b) \qquad (4)$$

where w_h is the weight vector of the hidden layer, adding a bias b and $\alpha()$ is the non-linearity transformation.

2.8. Learning to Rank

The problem of reranking text pairs encompasses a large array of information retrieval (IR) tasks, such as answer sentence selection in question answering, microblog retrieval, etc. To perform the reranking accurately, effective representation of query-document pairs must be ensured.

2.8.1. Problem formulation

Learning to rank [8] made by getting a set of retrieved lists, where each query $q_i \in Q$ comes together with all potential documents $D_i = \{d_{i_1}, d_{i_2}, \dots, d_{i_n}\}$. These candidates set comes together with their labels $\{y_{i_1}, y_{i_2}, \dots, y_{i_n}\}$ that indicate how much the i-document relevant to i-query, where those that are relevant have labels equal to 1 (or higher) and 0 otherwise. The main goal is to build a model that generates an optimal ranking R for each query q_i and its potential documents D_i.

More formally, the task is to learn a ranking function:

$$h\left(w, \varphi\left(q_i, D_i\right)\right) \rightarrow R \qquad (5)$$

where function φ() maps query-document pairs to a feature vector representation where each component of the vector represents another type of similarity, e.g. lexical, syntactic, and semantic. The parameter w of the model is a weight vector and is learned during the training.

2.8.2. Learning to Rank approaches

For IR tasks, there are three approaches to learn the ranking function h: pointwise, pairwise and listwise.

The pointwise approach is the simplest and most common approach to build a reranker, where the training instance are triples $\left(q_i, d_{ij}, y_{ij}\right)$, and the result of the reranker is a binary classifier: $h\left(w, \varphi(q_i, d_{ij})\right) \rightarrow y_{ij}$, where φ maps query-document pair to a feature vector and w is a vector of model weights. The decision function $h(\cdot)$ is a linear form that computes the dot product between w and $\varphi(\cdot)$. After learning the model, it is can use to classify unseen pairs (q_i, d_{ij}). A more advanced approach to reranking is pairwise, and it is very similar to the pointwise approach, but it is more accurate because it takes advantage of more information on truth labeling of the input candidates. Therefore, this approach requires a larger set of data (potentially square size from the candidate document) which can lead to slow training times. The model is trained to score correct pairs higher than incorrect pairs with a certain margin:

$$h\left(w, \varphi(q_i, d_{ij})\right) \geq h\left(w, \varphi(q_i, d_{ik})\right) + \varepsilon \qquad (6)$$

where document d_{ij} is relevant and d_{ik} is not, still both pointwise and pairwise approaches ignore the fact that ranking is a prediction task on a list of objects. The third approach is listwise, treats a query with its list of candidates as a single instance in learning, thus it also takes advantage of more information on truth labeling of the input candidates. The pairwise and listwise approaches are yield better performance, but they are more complicated to implement and less effective for training.

3. APPROACH

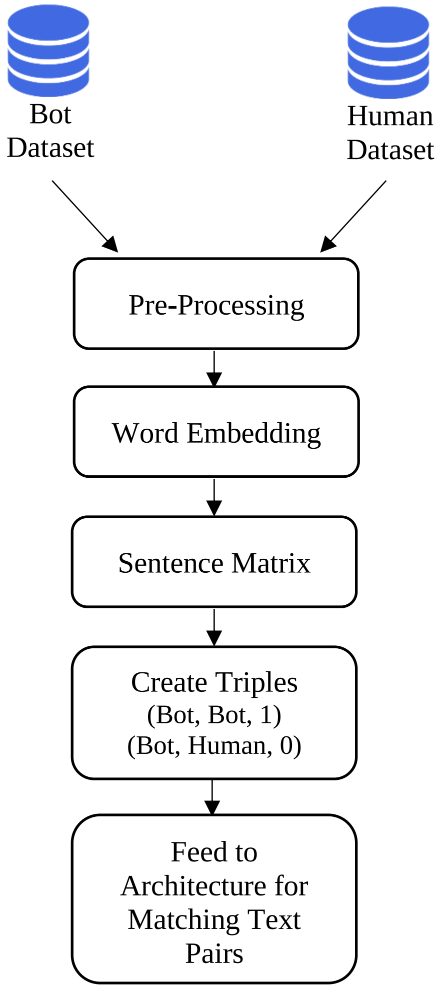


Figure 6 – Diagram of the training process workflow

The training process flow is as follows: There are two tweets dataset – bot tweets dataset $Q = \{q_1, q_2, \dots, q_n\}$ and human tweets dataset $D = \{d_1, d_2, \dots, d_m\}$. Then, for each tweet $q_i \in Q$ and $d_j \in D$, we perform the preprocessing process for removing irrelevant characters and words, and for each of them, we create a Sentence Matrix that contains its words in their embedding representation. At the end of this process, we get a collection of tweets in their Sentence Matrix representation $Q^* = \{Q'_1, Q'_2, \dots, Q'_n\}$ and $D^* = \{D'_1, D'_2, \dots, D'_m\}$, when Q'_i and D'_j are the Sentence Matrix that represents respectively the tweets q_i and d_j.

To enrich our dataset, we chose to divide the bot tweets into groups, so that each group represents the tweets written by the same user. After partition, for each user i, we get a group $Q^*_i = \{Q'_{i_1}, Q'_{i_2}, \dots, Q'_{i_m}\}$. In the next step, we create triples for training instances $(Q'_{i_j}, Q'_{i_k}, 1)$ in case of pairing two Sentence Matrix of bot tweets that written by the same user, and $(Q'_{i_j}, D'_k, 0)$ in case of pairing Sentence Matrix of bot and human tweets. In this method, the network can learn the semantic similarity better between bot tweets written by the same user and labeled by 1, and the difference between bots and human tweets labeled by 0.

At the end of the pre-processing steps, those triples instances are fed into the architecture for matching text pairs for training the network.

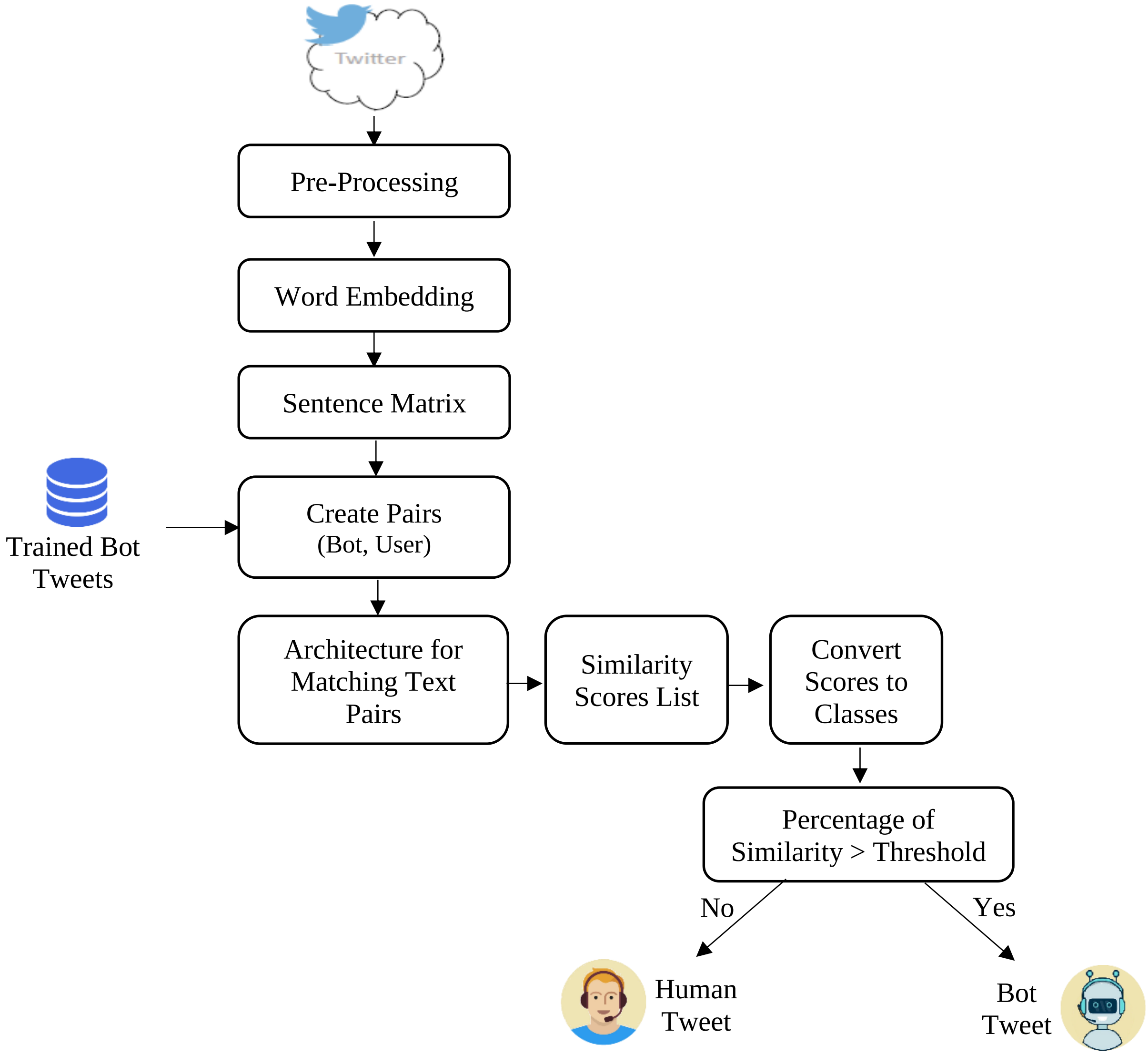


Figure 7 – Diagram of the predicting process workflow

The user inserts a tweet to find whether the tweet was written by a person or by a bot. Then, for this tweet, we perform the preprocessing process and we create a Sentence Matrix that contains its words embedding representation. As part of the training model process, we create a list of bot tweets in their Sentence Matrix representation, and we use them for binding the tweet of the user for each of them. This process allows us to create pairs of bot-user tweets, so we can insert the pairs into the architecture for matching text pairs for checking if this tweet is similar for every bot tweet.

As the output of the network, we will get the similarity scores list for each of the pairs we created, and then we convert the scores to classes (when zero means the two tweets have a different sentence structure, one means the opposite).

In order to determine if the tweet written by a bot or human, we calculate the percentage of tweets that are similar, by calculating the ratio of the number of pairs who received the classification 1 to the number of total pairs. If this ratio passes the threshold selected by the user, the model will identify the tweet as bot tweet.

3.1. Sentence Model

The architecture of ConvNet for mapping sentences to feature vectors is shown in Figure 4. The main goal of this model is to learn a good intermediate representation of the queries and documents, which are then used for computing their semantic matching.

The network consists of a single wide convolutional layer followed by a non-linearity activation function that followed by simple max pooling.

The input for the network is raw words that need to be mapped into real-valued feature vectors that processed by subsequent layers of the network.

3.1.1. Sentence Matrix

The sentence model requires a sentence s as the input and it presented as sequence of words: $[w_1, \ldots, w_{|s|}]$, where each word was taken from a vocabulary V. The words are represents by distributional vectors $w \in \mathbb{R}^d$, where d is the size of the vector, and are looked up in a word embedding matrix $W \in \mathbb{R}^{d\times|V|}$, which is performed by concatenating embeddings of all words in V.

The sentence matrix $S \in \mathbb{R}^{d\times|s|}$ performed for each input sentence s, where each column i represents a word embedding w_i at position i in a sentence (Figure 4).

$$\mathbf{S} = \begin{bmatrix} | & & | \\ \mathbf{w}_1 & \ldots & \mathbf{w}_{|s|} \\ | & & | \end{bmatrix}$$

Figure 8 – Sentence Matrix

The neural network of the sentence model applies a series of transformations to sentence matrix S using convolution, nonlinearity and pooling operations, to learn features of individual words in a sentence from low-level of word embeddings to higher-level semantic concepts.

3.2. The architecture for matching text pairs

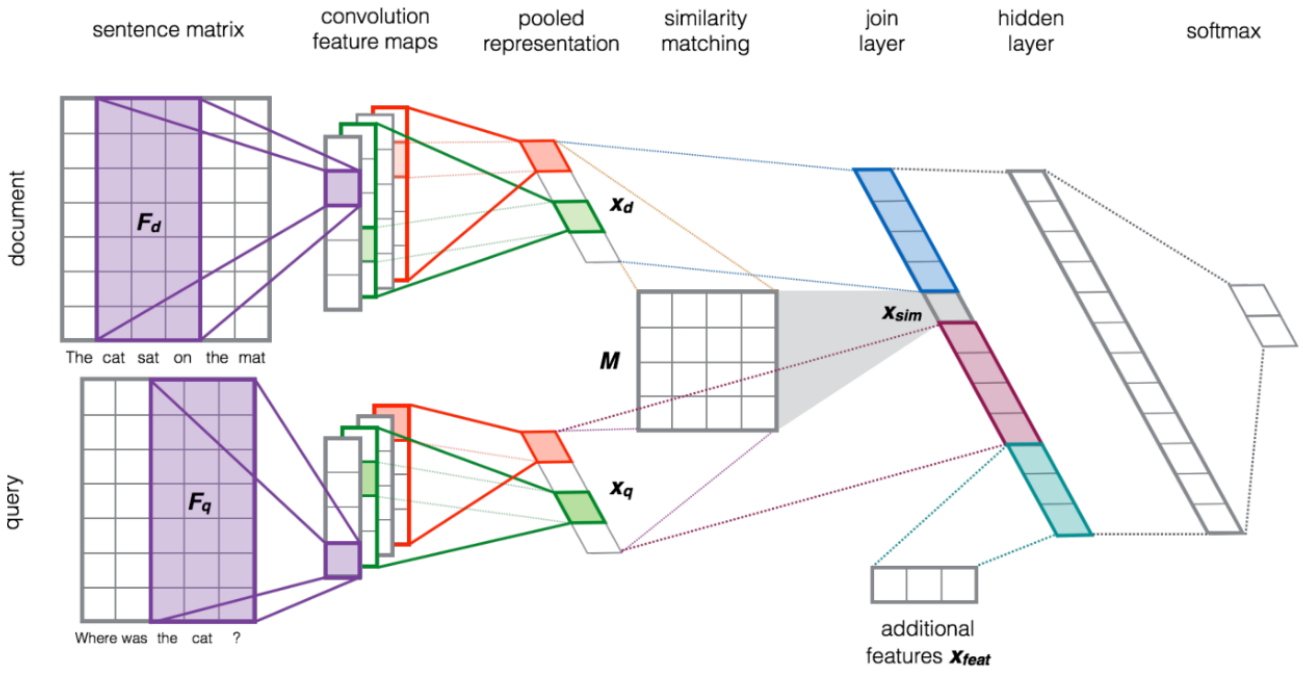


Figure 9 – Deep learning architecture for reranking short text pairs

The model based on architecture for matching query-document pairs as presented in Figure 9. The sentence model mapping the input sentence to vector representation, which also called intermediate representation, which can then be used to compute their similarity. Then, we used the vector representation to compute a query-document similarity score, which we take and combined with query and document vector representation into a single joint representation.

3.2.1. Matching query and documents

Sentence ConvNets model is processing queries and documents in parallel, which output the results as vector representations x_q (query vector) and x_d (document vector), and their representations can be used to compute a query-document similarity score. Then, the similarity between x_q and x_d vector as follows:

$$sim(x_q, x_d) = x_q^T M x_d \qquad (7)$$

where $M \in \mathbb{R}^{d \times d}$ in Equation 6 is a similarity matrix, and this equation has been widely used as a scoring model in information retrieval. In this model, we seek a transformation of the candidate document $x_d' = Mx_d$ that is the closest to the input query x_q. The similarity matrix M is a parameter of the network and is optimized during the training.

3.2.2. The information flow

In this section, we provide a full description of the deep learning network (shown in Figure 9) that maps input sentences to class probabilities. Then, the output of the sentence model is vector representation of a query $\boldsymbol{x_q}$ and a document $\boldsymbol{x_d}$ and known as vector representation. These are then matched using a similarity matrix M and according to Equation 6. Thus, it produces a single score x_{sim} that including various aspects of similarity (syntactic and semantic) between the input queries and documents. This architecture also allows us to add additional features x_{feat} to the model, such as word overlapping that gives information about overlapping words in a pair and provides a significant boost in accuracy and yields new state-of-the-art results.

The join layer concatenates all intermediate vectors, the similarity score x_{sim}, and any additional features into a single vector:

$$x_{join} = \left[x_q^T, x_{sim}, x_d^T, x_{feat}^T\right] \qquad (8)$$

Then, this vector is passed through a fully connected hidden layer, which allows for modeling interactions between the components of the joined representation vector. Finally, the output of the hidden layer is passed through to the Softmax classification layer, which generates a distribution over the class labels.

4 SOFTWARE ENGINEERING DOCUMENTATION

4.1 Use Case

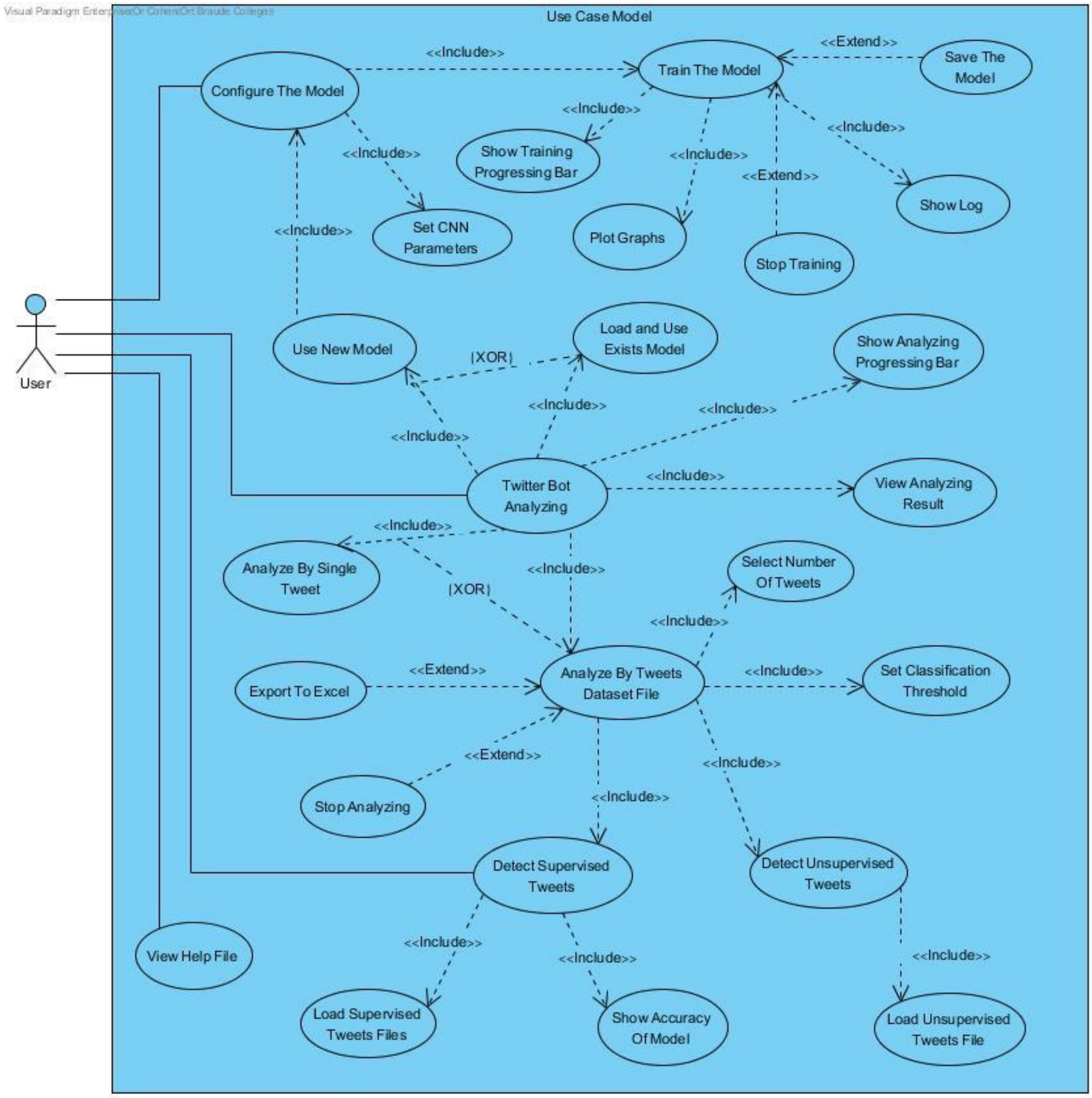


Figure 10 – Use Case diagram

4.2 Class Diagram

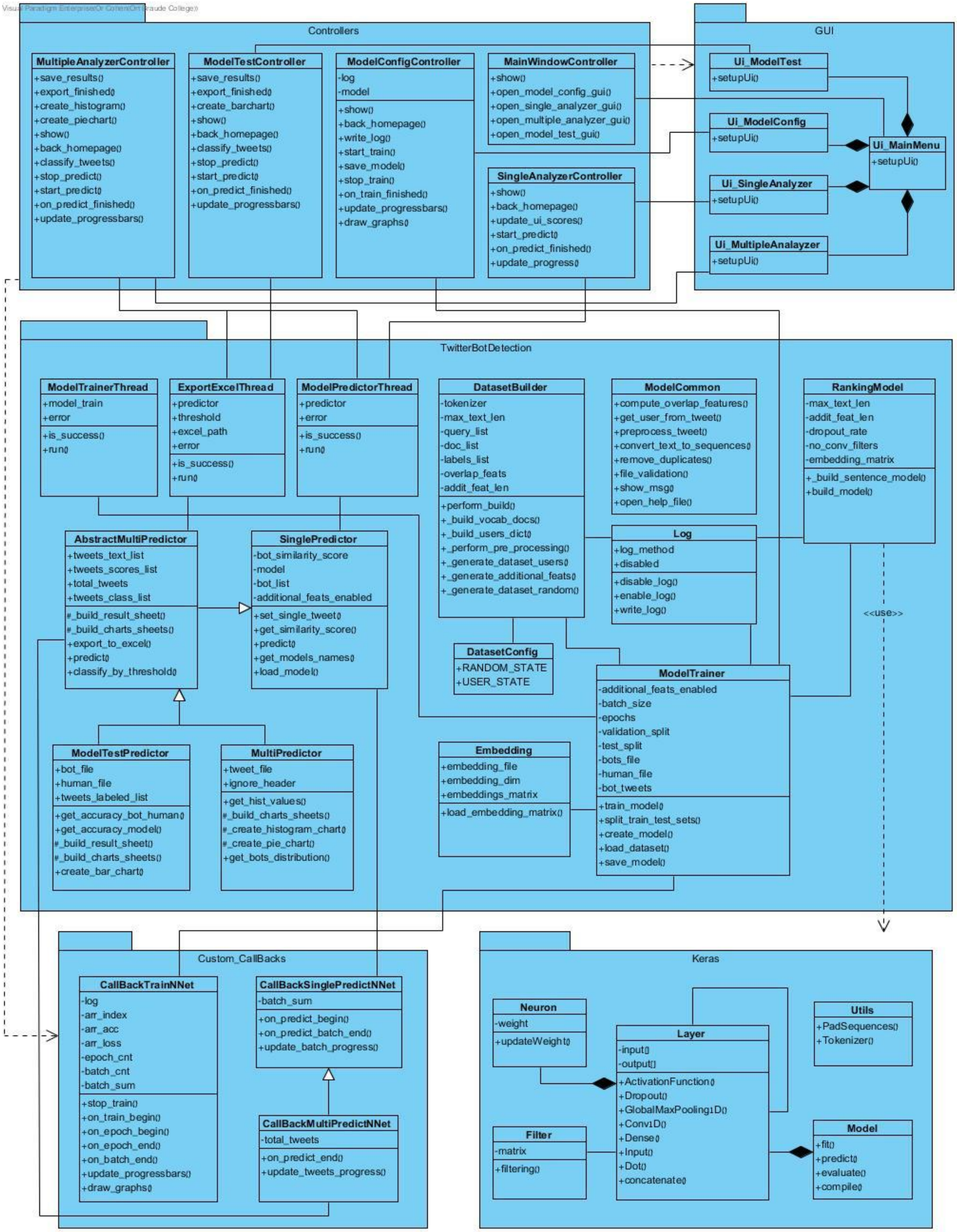
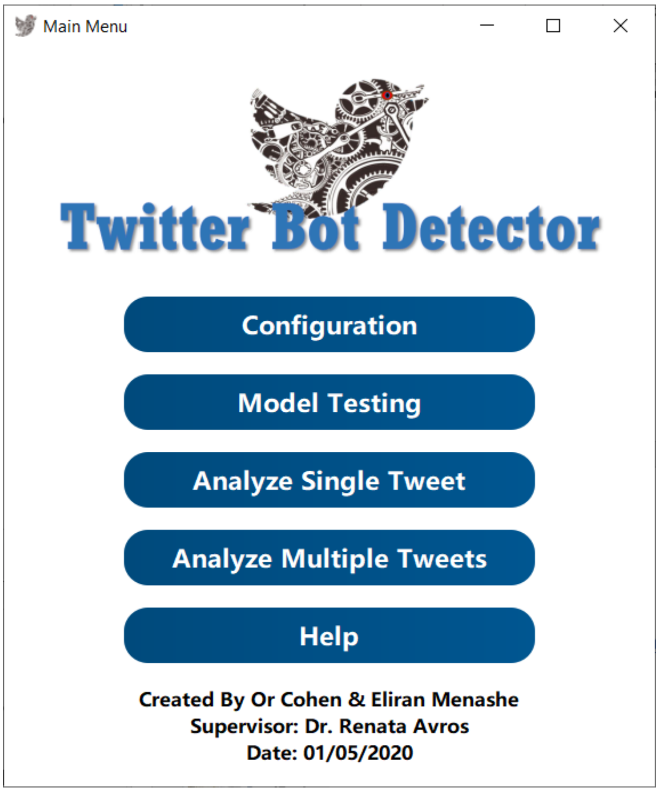


Figure 11 – Class diagram

4.3 User Interface

4.3.1 Windows

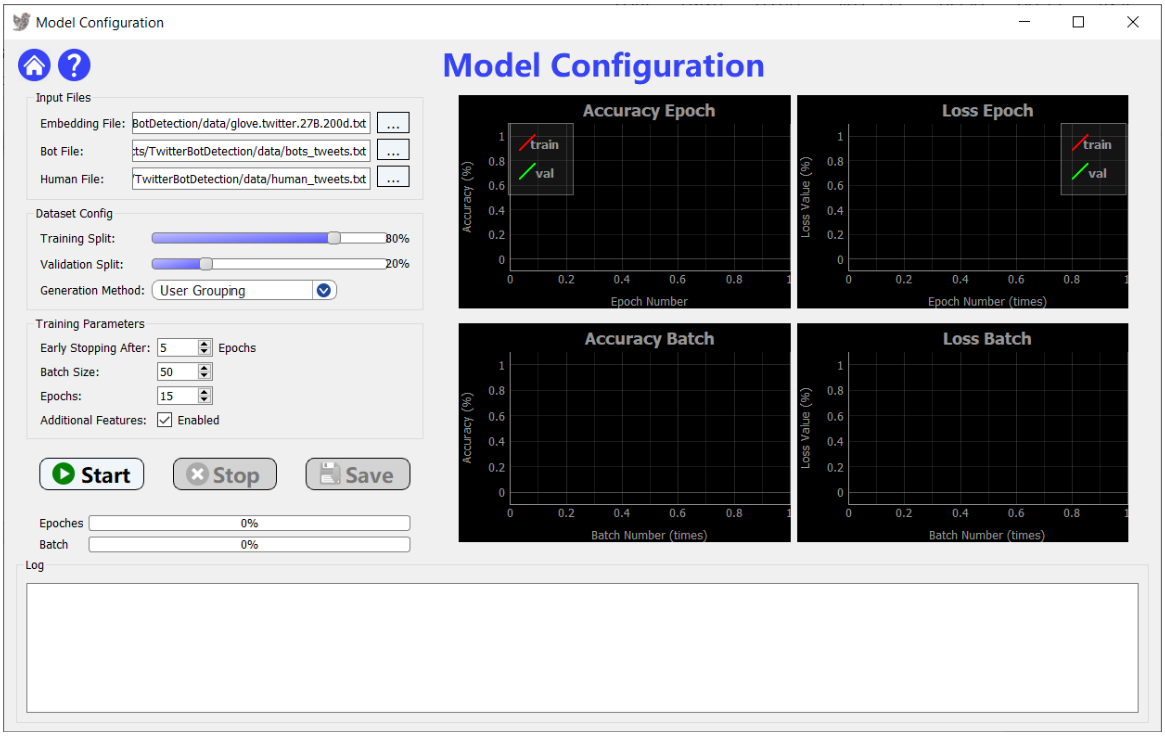
Main window – The first window that the user sees when he opens the application. This window contains all the options of the system. If the user selects in some option, it will open the relevant window for this option. In addition, he can press on “Help” button if he wants to see explanations about the system, and this current window.



Created By Or Cohen & Eliran Menashe
Supervisor: Dr. Renata Avros
Date: 01/05/2020

Figure 12 – Main window

If the user selects the "Configuration" option in the "Main Window", it will open the "Model Configuration" window. For training a new model, the user needs to load the embedding file, bot file, and human file. In addition, he needs to choose the training and validation split, the generation method for creating the dataset (User Grouping/Random Pairing), and CNN parameters (early stopping, batch size, epochs). If the user checks Additional Features, he can get additional info between pairs such as word overlapping for getting better accuracy. After choosing all those parameters, the user can start the training by clicking on the "Start" button. During the training process, all the graphs and logs are updated in real-time and the user can stop the process by clicking on the "Stop" button. At the end of the training, the user can save the final model by clicking on the "Save" button.



Embedding File: BotDetection/data/glove.twitter.27B.200d.txt

Bot File: ts/TwitterBotDetection/data/bots_tweets.txt

Human File: TwitterBotDetection/data/human_tweets.txt

Training Split: 80%

Validation Split: 20%

Generation Method: User Grouping

Early Stopping After: 5 Epochs

Batch Size: 50

Epochs: 15

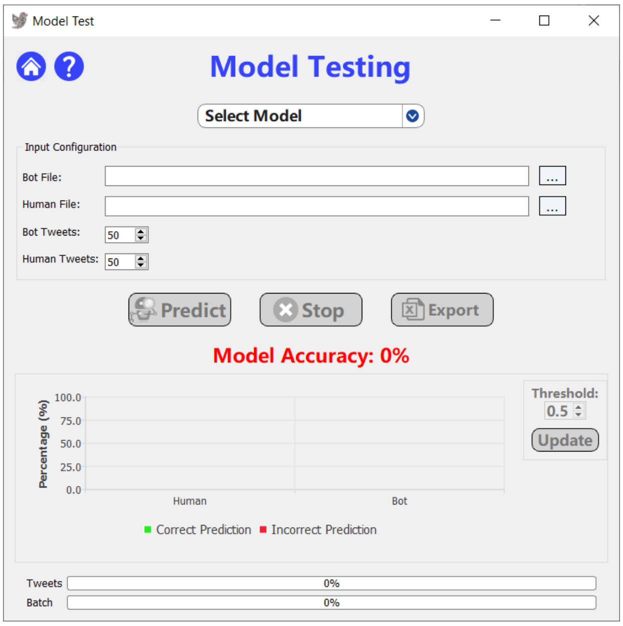
Additional Features: ☑ Enabled

Epoches 0%

Batch 0%

Figure 13 – Model Configuration window

If the user selects the "Model Testing" option in the "Main Window", it will open the "Model Testing" window. First, the user needs to select an existing trained model. In addition, he needs to choose a bot file and human file and define the number of tweets for bots and human analyzing. When the user clicks on the "Predict" button, the prediction process is starting, and the user can stop the process by clicking on the "Stop" button. At the end of the prediction, the user can show the results in the window: the model accuracy and bar chart that shows the correct/incorrect prediction for bot and human classification. In addition, he can update the results by selecting another threshold and clicking on the "Update" button. He also can save the results by clicking on the "Export" button.



Bot File:

Human File:

Bot Tweets: 50

Human Tweets: 50

Model Accuracy: 0%

Threshold: 0.5

Tweets 0%

Batch 0%

Figure 14 – Model Testing window

If the user selects the "Analyze Single Tweet" option in the "Main Window", it will open the "Single Tweet Analyzer" window. First, the user needs to select an existing trained model, and he needs to write or paste the tweet that he wants to predict. At the end of the prediction process, he can see the scores of the bot and human classification on the window.

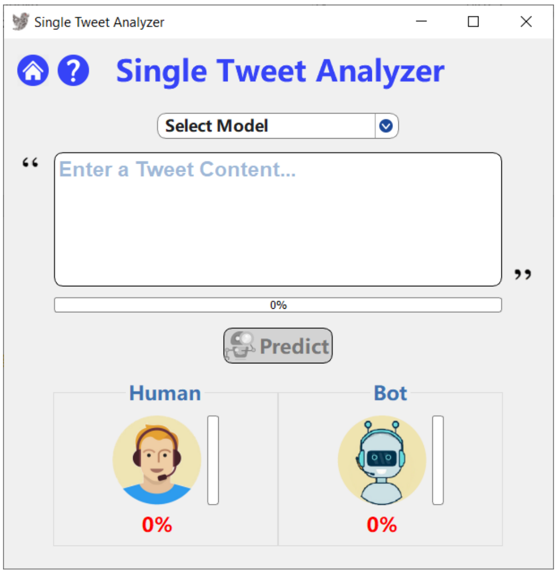
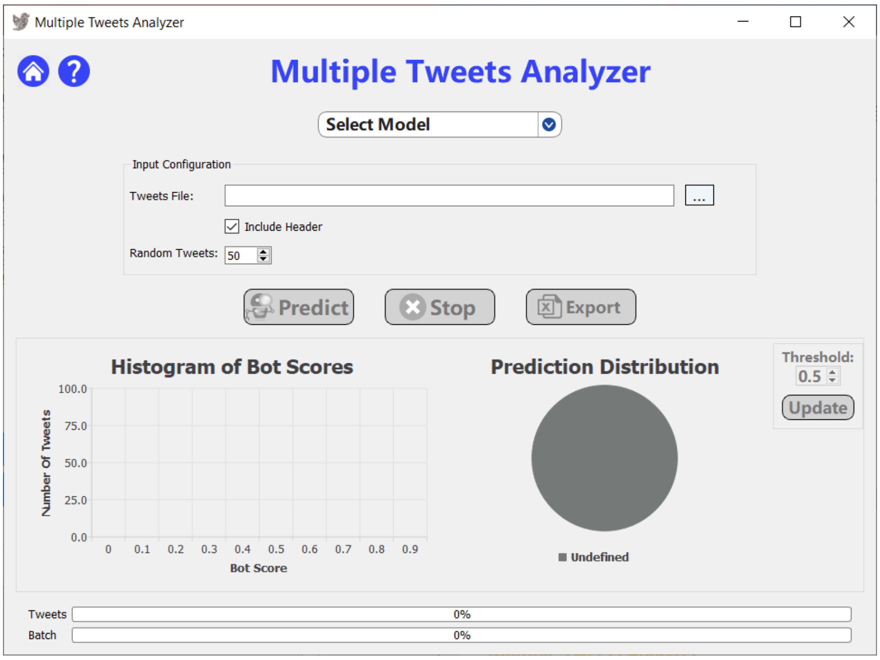


Figure 15 – Single Tweet Analyzer window

If the user selects the "Analyze Multiple Tweets" option in the "Main Window", it will open the "Multiple Tweets Analyzer" window. First, the user needs to select an existing trained model. In addition, he needs to choose a dataset file, and define the number of tweets for analyzing. When the user clicks on the "Predict" button, the prediction process is starting, and the user can stop the process by clicking on the "Stop" button. At the end of the prediction, the user can show the results in the window with the pie chart that show the classification distribution, and the histogram that shows the prediction scores of bots. In addition, he can update the results by selecting another threshold and clicking on the "Update" button. He also can save the results by clicking on the "Export" button.



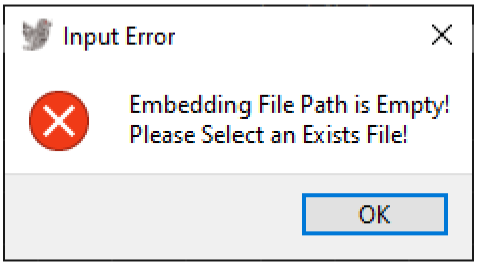
Random Tweets: 50

Threshold: 0.5

Figure 16 – Multiple Tweet Analyzer window

4.3.2 Errors and alerts

If the user clicks on the "Start" button on "Model Configuration" window and he does not choose an embedding file -

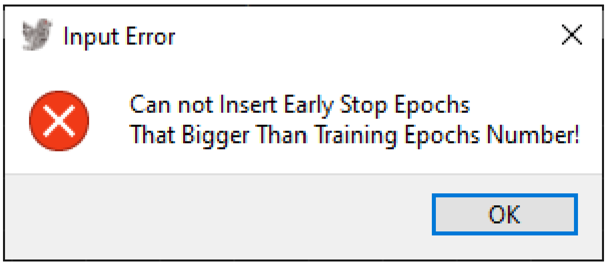


Input Error

Embedding File Path is Empty!
Please Select an Exists File!

Figure 17 – Missing file error – error message

If the user clicks on the "Start" button on "Model Configuration" window and he chooses early stopping that bigger than the number of total epochs for training -

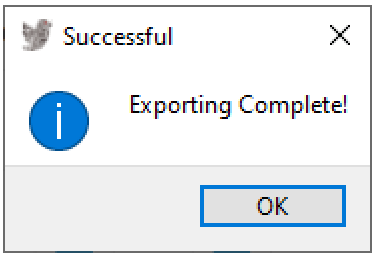


Input Error

Can not Insert Early Stop Epochs
That Bigger Than Training Epochs Number!

Figure 18 – Early stop epochs error – error message

If the user clicks on the "Export" button on "Multiple Tweets Analyzer" window and the process finished successfully –



Successful

Exporting Complete!

Figure 19 – Save successfully – alert message

4.4 Testing Plan

In order to check out the system performance, we run the program with some specific inputs to test its behavior.

Test Name	Description	Expected Result	Actual Result
Create a valid model	1) Choose an embedding file, bot, and human dataset files. 2) Choose 80% in training split and 33% in validation split and "User Grouping" method. 3) Insert early stopping after 5 epochs, 50 in batch size, 15 in epoch number. 4) Click on "Start".	Display a message in the log window: "Training Process Completed Successfully!"	Pass
Saving a model	When the model created successfully, save it to output folder.	Model file located on output folder in hard disk.	Pass
Invalid early stopping number	1) Insert 15 in epoch number. 2) Insert 25 in early stopping number.	Display error message: "Cannot Insert Early Stop Epochs That Bigger Than Training Epochs Number!"	Equal
File path is empty	Leave the text box of loading file as empty	Display error message: "File Path is Empty!"	Equal
Loading an empty file	Load an empty file	Display error message: "File is Empty!"	Equal
Loading file that does not exist	Load a file that does not exists in hard disk	Display error message: "File is Not Exists!"	Equal
Export results in model testing	1) Choose a model and the dataset files for bots and humans. 2) Click on "Predict". 3) After the process completed successfully, click on "Export".	Display alert: "Exporting Complete!"	Pass
Export results in multiple analyzer	1) Choose a model and the dataset file. 2) Click on "Predict". 3) After the process completed successfully, click on "Export".	Display alert: "Exporting Complete!"	Pass
Random tweets number overflowing	1) Choose a dataset file with 10 tweets. 2) Insert random tweets number as 50 tweets. 3) Click on "Predict"	Display error message: "The Random Bot Tweets You Choose is Bigger Than Number of Tweets in File!"	Equal

Table 1 – Testing plan

5. RESULTS AND CONCLUSIONS

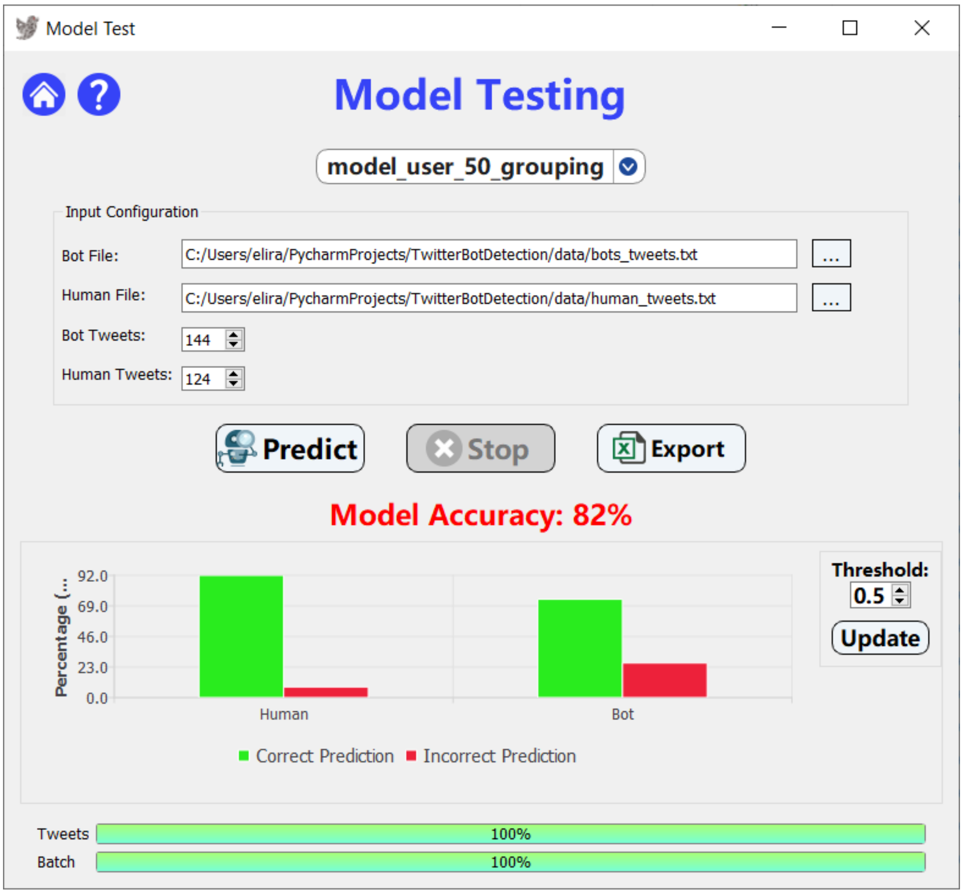
5.1 Results

5.1.1 Experiment 1

In this experiment, we check the accuracy prediction between the two categories – human and bot, when changing the threshold value, to draw conclusions about appropriate threshold selection for different tasks.

For this experiment, we trained the model with a user grouping method that generates the bot tweets pairs per each user. We also used with additional information feature that includes word overlapping between each pair, and with the following parameters: epochs number – 15, batch size – 50, training split – 80%, validation split – 33%, pre-trained word2vec – "glove.twitter.27B.200d".

We test this experiment with 268 tweets which contain 144 tweets written by bots and 124 tweets written by humans.



Model Testing

model_user_50_grouping

Input Configuration

Bot File: C:/Users/elira/PycharmProjects/TwitterBotDetection/data/bots_tweets.txt

Human File: C:/Users/elira/PycharmProjects/TwitterBotDetection/data/human_tweets.txt

Bot Tweets: 144

Human Tweets: 124

Model Accuracy: 82%

Threshold: 0.5

Tweets 100%

Batch 100%

Figure 20 – Predicting result on "Model Test" window

Test 1: Predict with threshold of 50%

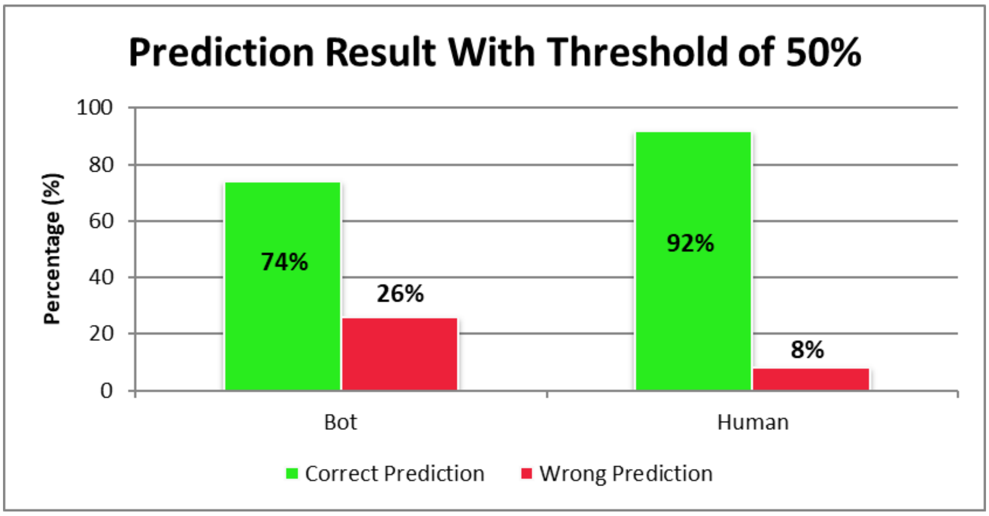


Figure 21 – Prediction diagram with threshold of 50%

In this test, we obtained a gap in the prediction results for the different categories, so the high accuracy in predicting human tweets comes at the expense of accuracy in bot tweets.

Predicted / Actual	Bot	Human
Bot	106	38
Human	12	114

Table 2 – Confusion matrix for threshold of 50%

The accuracy obtained is: 82%.

Test 2: Predict with threshold of 33%

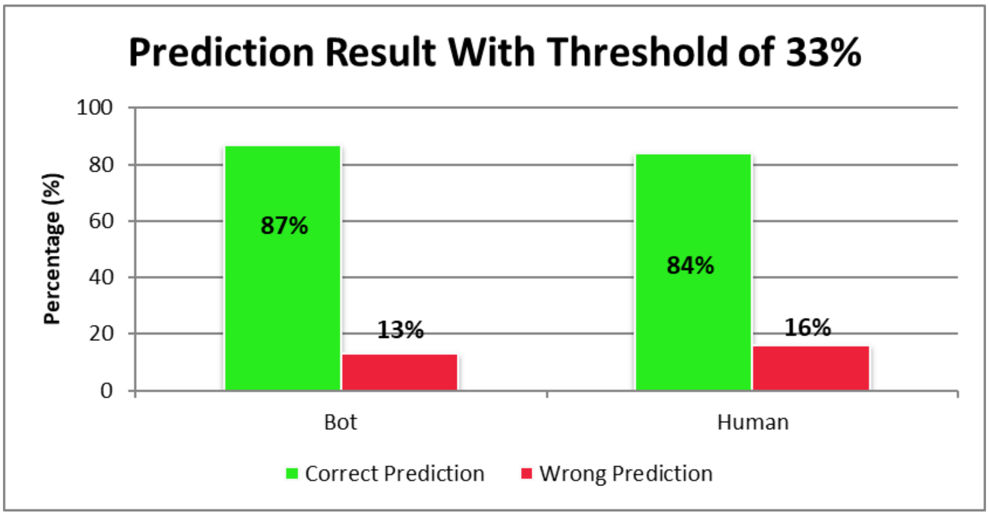


Figure 22 – Prediction diagram with threshold of 33%

Now, we can see that we got a balance in the prediction results for the different categories. In this case, we also got high accuracy for predicting human tweets and even higher for bot tweets. Therefore, we can conclude that this threshold is appropriate for the two tasks – for increasing the true-positive of human tweets and even more for bot tweets.

Predicted Actual	Bot	Human
Bot	125	19
Human	20	104

Table 3 – Confusion matrix for threshold of 33%

The accuracy obtained is: 85%.

Test 3: Predict with threshold of 75%

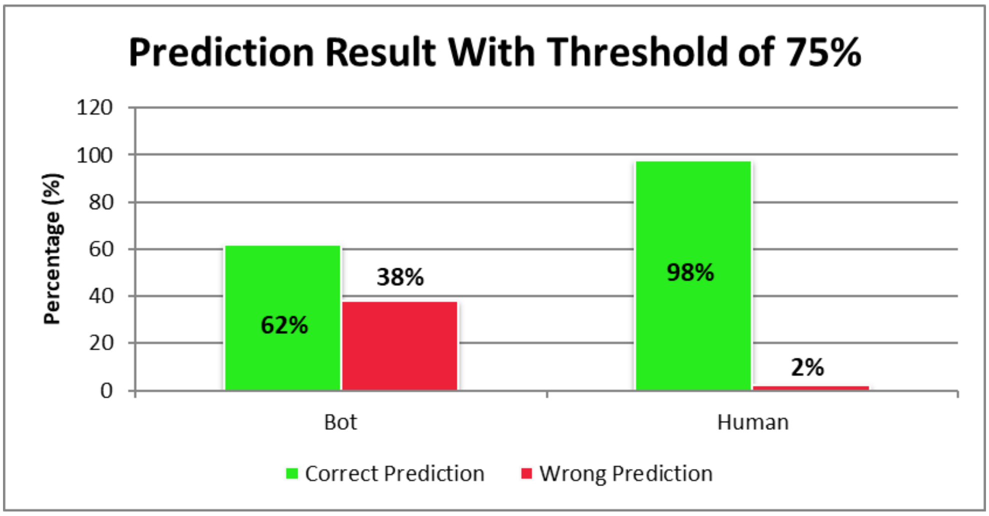


Figure 23 – Prediction diagram with threshold of 75%

In this test, we can see the same tendency as we got in the test with a threshold of 50%, but here the accuracy for human tweets has significantly increased, reaching 98%. Therefore, we can conclude that this threshold is much better for increasing the true-positive of human tweets.

Predicted Actual	Bot	Human
Bot	89	55
Human	3	121

Table 4 – Confusion matrix for threshold of 75%

The accuracy obtained is: 78%.

5.1.2 Experiment 2

In this experiment, we test the methods for pairs generation from the dataset of the bot and human tweets. The first method is "User Grouping" which means that the pairs of bot tweets are taken from the same user, and the second method is "Random Pairing" which means that the pairs have taken from random tweets.

For "User Grouping" method, we used the same model as the previous experiment, and for "Random Pairing" method, we trained a new model with the same parameters as the previous experiment but with the method of "Random Pairing". In addition, we test this experiment with the same 268 tweets from the previous experiment, by the threshold of 33%.

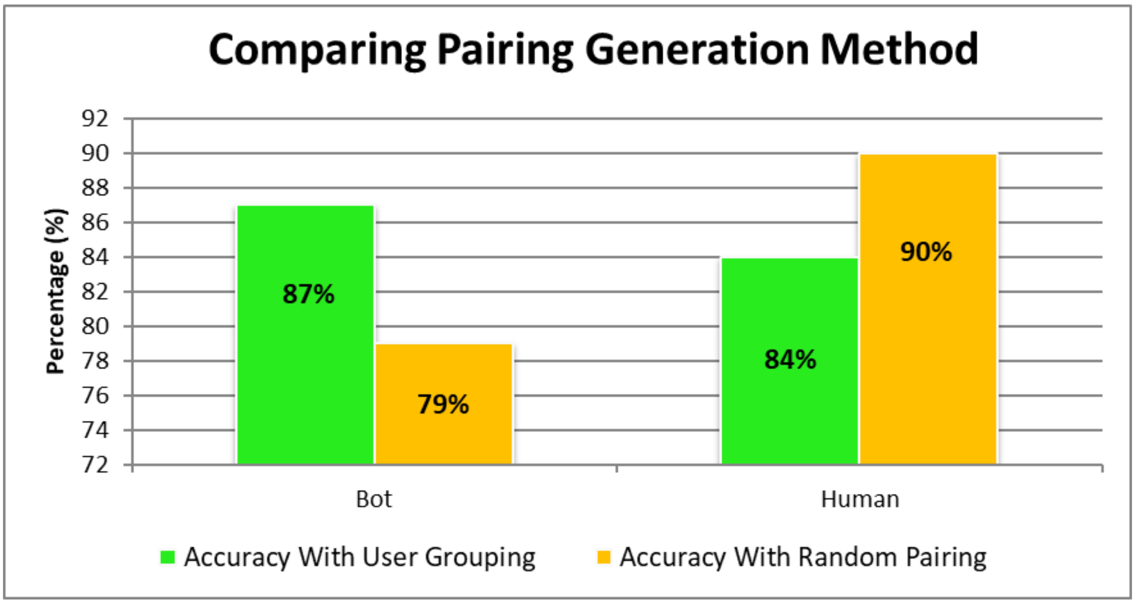


Figure 24 – Prediction diagram by random pairing with threshold of 33%

Predicted / Actual	Bot	Human
Bot	114	30
Human	12	112

Table 5 – Confusion matrix by random pairing with threshold of 33%

As we can see from the results, when using the model of "Random Pairing", the accuracy of bot detection decreased to 79% from 87% of the "User Grouping" method. On the other hand, the accuracy of human detection increased to 90% from 84%. Therefore, we can conclude that the "Random Pairing" method is more appropriate for increasing the true-positive of human tweets.

5.1.3 Experiment 3

In this experiment, we repeat testing the methods for pairs generation from the dataset, as we have done in the previous experiment, but now we trained models for each method without additional information between the pairs.

We want to examine for each dataset generation pairs method - User Grouping and Random Pairing, how adding or disabling additional features to model affects the accuracy.

For this experiment, we trained the models with the following parameters: epochs number – 15, batch size – 50, training split – 80%, validation split – 33%, pre-trained word2vec – "glove.twitter.27B.200d".

We test this experiment with the same 268 tweets from the previous experiments, by the threshold of 33%.

Test 1: "User Grouping" method

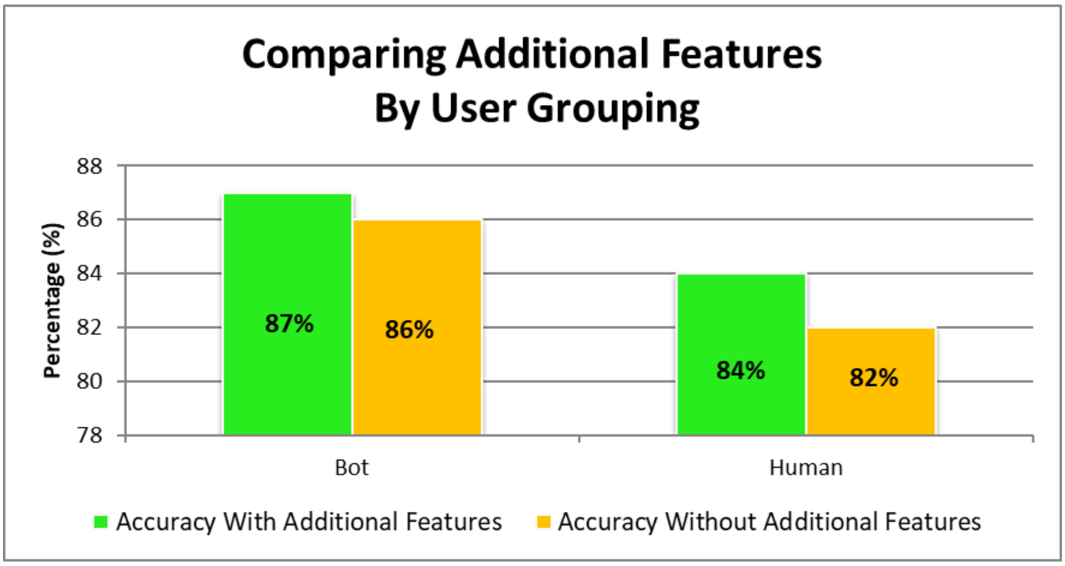


Figure 25 – Prediction diagram by user grouping without additional features

Predicted / Actual	Bot	Human
Bot	124	20
Human	22	102

Table 6 – Confusion matrix by user grouping without additional features

In this test, we can see that using the model without additional features cause to decrease in the accuracy of the bot detection and human detection.

Test 2: "Random Pairing" method

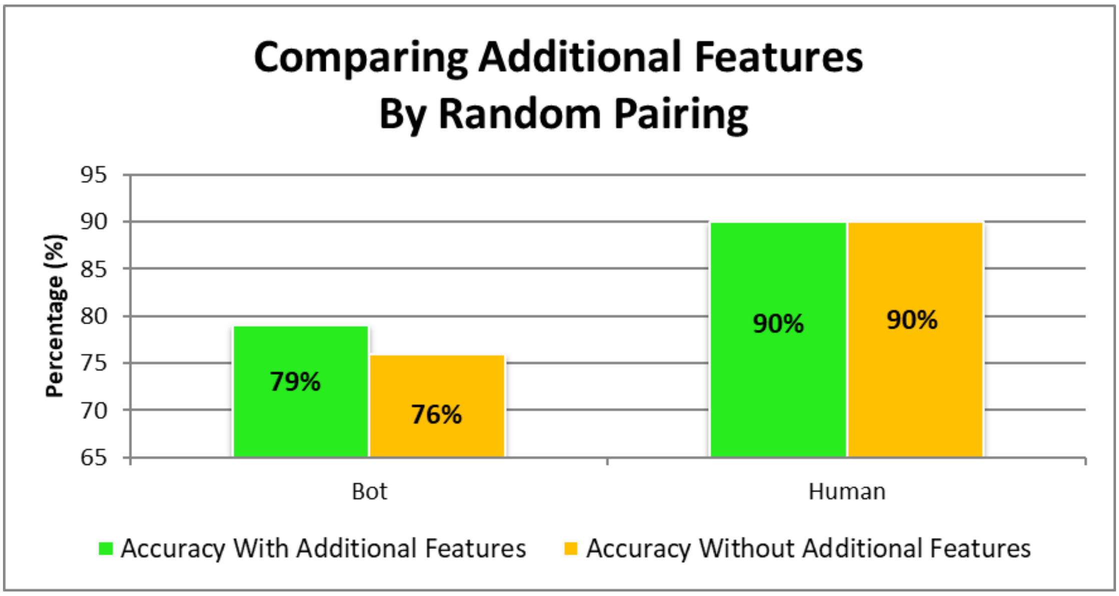


Figure 26 – Prediction diagram by random pairing without additional features

Predicted / Actual	Bot	Human
Bot	109	35
Human	13	111

Table 7 – Confusion matrix by random pairing without additional features

In this test, we can see that there is no change in the accuracy of human detection but using the model without additional features cause to decrease the accuracy of bot detection from 79% to 76%.

5.1.4 Experiment 4

In this experiment, we want to test 300 unclassified tweets about COVID-19 and examine the extent of bots in this data set. We used it as the same model that we trained for Experiment 1. In addition, we concluded in this experiment that using by the threshold of 33% yields the best accuracy for increasing the true-positive of both – human and bot tweets.

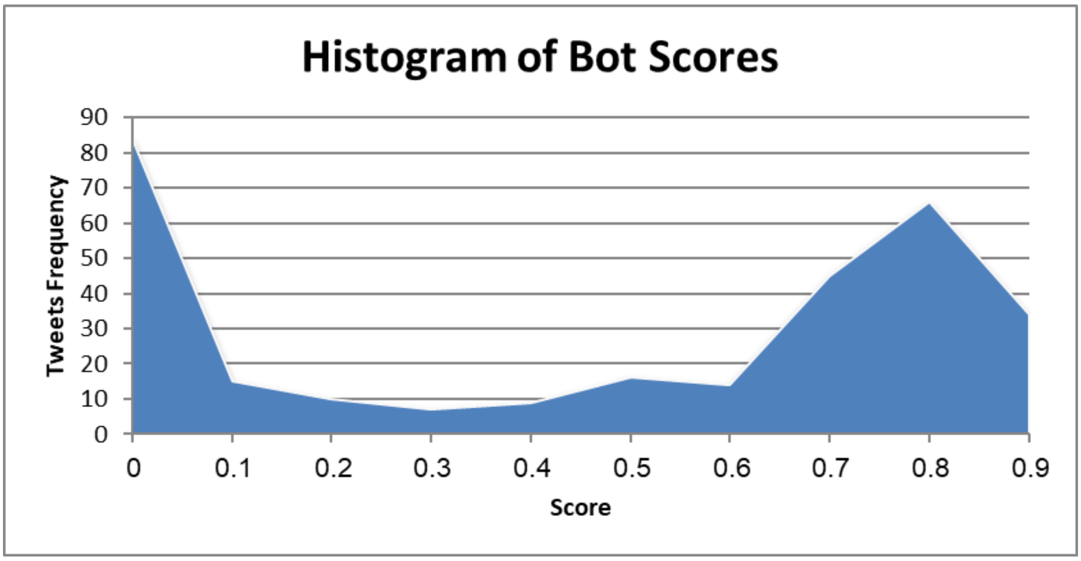


Figure 27 – Histogram of prediction scores across the dataset

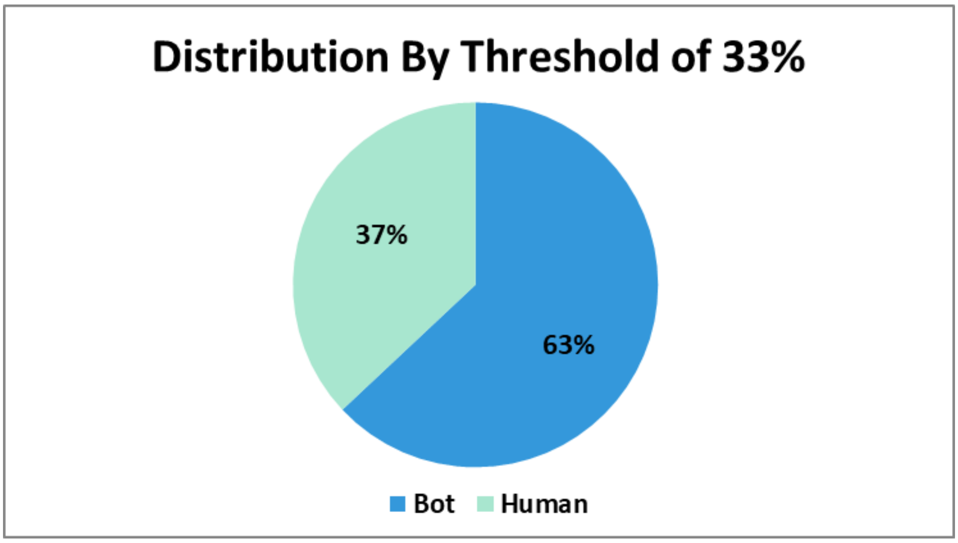


Figure 28 – Pie chart of classification distribution

In this experiment, as we can see from the histogram in Figure 27, we can conclude that there are a large number of tweets that have received a score higher than 0.6, which could indicate a plurality of bots in the test set. In addition, the pie chart in Figure 28, you can see that 63% of the tweets were classified as bots, compared to 37% that were classified as human tweets.

5.2 Conclusions

At the experiment stage, we examined how changing training parameters affects the accuracy of prediction. We learn from the training process that the model performance stops improving on a validation dataset after a small number of epochs.

From the first experiment, changing the threshold value results in different percentages of accuracy for human or bot-written tweets. We have concluded that different threshold values are appropriate for different tasks, for example, for a 75% threshold, a significant increase in human-written tweets is made at the expense of bot tweets, and a 33% threshold has the best true-positive balance for Human tweets and also for bot tweets.

From the second experiment, we learned that using the "User Grouping" method, which the bots of tweets are taken from the same user, yields the best accuracy percentages for both - human and bot tweets, while the "Random Pairing" method adds noise and significantly reduces the true-positive of identifying bots tweets.

In the third experiment, we wanted to examine the extent to which the use of additional features affects the accuracy percentages, which in this feature for each pair also considers the number of overlapping words. From this experiment, we found that the accuracy percentages were lower for both categories and especially for bot tweets.

In the last experiment, we wanted to examine the presence of bots in tweets written on COVID-19. we performed this test with the model we tested in the first experiment by a threshold value of 33%, and we concluded that a significant portion of 63% of tweets was written by a bot.

In the future, there is an option to expand the data set of tweets written by bots, which can assist to improve the classification results. In addition, considering the possibility of a new bot generation, we suggest repeating the training process periodically.

We hope that our program will be able to help eradicate the phenomenon of offensive bots on Twitter.

6. REFERENCES

[1] Welbers, K., Van Atteveldt, W., & Benoit, K., "Text Analysis in R", *Communication Methods and Measures*, 2017.

[2] Y. Liu, Z. Liu, T.S. Chua and M. Sun, “Topical Word Embeddings”, *International Journal of Computer Applications in Technology,* 2015.

[3] X. Rong, “word2vec Parameter Learning Explained”, *arXiv preprint arXiv: 1411.2738,* 2016.

[4] D. Gupta, "Architecture of Convolutional Neural Networks (CNNs) demystified," *Analytics Vidhya,* 2017.

[5] N. Kalchbrenner, E. Grefenstette, and P. Blunsom, "A convolutional neural network for modelling sentences", *Proceedings of the 52nd Annual Meeting of the Association for Computational Linguistics*, June 2014.

[6] C. E. Nwankpa, W. Ijomah, A. Gachagan, and S. Marshall, "Activation Functions: Comparison of Trends in Practice and Research for Deep Learning", *arXiv preprint arXiv: 1811.03378*, 2018.

[7] Y. Chauvin (Ph. D.), D.E. Rumelhart, *Backpropagation: Theory, Architectures, and Applications*, Psychology Press, 1995.

[8] A. Severyn and A. Moschitti, "Learning to Rank Short Text Pairs with Convolutional Deep Neural Networks", *ACM SIGIR Conference on Research and Development in Information Retrieval*, 2015.