Software Engineering Department

ORT Braude College

Course 61766: Extended Project in Software Engineering

Bot Detection using Sentence Structure

In Partial Fulfillment of the Requirements for

Final Project in Software Engineering (Course 61771)

Karmiel – July 2019

Itai Tal 305781601

Ariel Yaakov 308155308

Supervisor:

Prof. Valery Kirzner

Advisor:

Prof. Zeev Volkovich

Contents

Abstract. *In our project, we present an approach to detect bots on Twitter based on sentence structure identification.*

A successful semantic matching algorithm needs to adequately model the internal structures of language objects and the interaction between them. In order to achieve this goal, we propose convolutional neural network models for matching two sentences, by adapting the convolutional strategy. Our models are generic and requiring no prior knowledge on language. In addition, the proposed models not only represent the hierarchical structures of sentences with their layer-by-layer composition and pooling, but also capture the rich matching patterns at different levels.

Based on the assumption that tweets written by the same generator have the same sentence structure, we created a convolution neural network model that capable to detect bots on Twitter. We hope that our program will be able to help eradicate the phenomenon of offensive and inflammatory bots on Twitter.

Keywords: Convolution Neural Network, Twitter bots detector, Fully Connected, Word Embedding.

1. INTRODUCTION

Bot on Twitter are semi- automated or automated programs that use the normal functions on Twitter, such as posting and tweeting. Unlike other social media websites such as LinkedIn, Twitter allows the usage of bots on their platform. In general, the bots are harmless and publicly reveal themselves as bots.

However, there is a group of political Twitter accounts, which are different. These are accounts who identify themselves as real humans and whose tweets are politically engaging. There are cases where these accounts are re-tweeting falsified information or advocating for violence or civil unrest.

Natural language sentences have complicated structure, both sequential and hierarchical, that are essential for understanding them. In our project, we present an approach to detect bots on Twitter based on sentence structure identification. We used deep neural network models to determine whether a tweet written by bot or by human, using a comparison between the unknown tweet and the bot tweets and get the compatibility degree between the sentence's structures.

2. THEORY

2.1. Convolutional Neural Networks (CNN)

A Convolutional Neural Network (ConvNet / CNN) is a Deep Learning algorithm, which can take in an input, assign importance (learnable weights and biases) to various aspects in the input and be able to distinguish one from the other.

The architecture of a CNN is analogous to that of the connectivity pattern of Neurons in the human brain and was inspired by the organization of the Visual Cortex. CNN has been successful in various text classification tasks.

Convolutional Neural Networks have different architecture than regular Neural Networks. Regular Neural Networks transform an input by putting it through a series of hidden layers. Every layer made up of a set of neurons, where each layer is fully connected to all neurons in the layer before. Finally, there is a last fully connected layer – the output layer – that represent the predictions.

In Convolutional Neural Networks, the layers organized in three dimensions: width, height and depth. Additionally, the neurons in one layer do not connect to all the neurons in the next layer but only to a small region of it. Lastly, the final output will be reduced to a single vector of probability scores [1].

2.2. Convolution layer

The primary purpose of convolution layers is to summarize the meaning of the input, until reaching a fixed length vectorial representation in the final layer.

The input of the beginning convolution layer to the most natural language processing (NLP) tasks consists of a vector representation of sentences or documents. Each row in such a matrix corresponds to one token, typically a word, but it could be a character. That is, each row is vector that represent a word (embedding of words).

The convolution layer's parameters is composed from a set of sliding windows commonly called a kernel, filter, or feature detector. For each filter, multiply its values element with the original matrix, and then sum them up. To get the full convolution it is done for each element by sliding (more precisely, convolve) the filter over the whole matrix.

CNNs have the same characteristics and follow the same approach, no matter if it is 1D, 2D or 3D. The main difference between these types is how the feature detector slides across the data:

- 1D convolution – the feature detector will always cover the whole word. The height determines how many words are considered when sliding the feature detector over the input.
- 2D convolution – the feature detector has two dimensions and it will slide both horizontally and vertically across the input.

The result of each convolution layer is smaller accurate matrix [2].

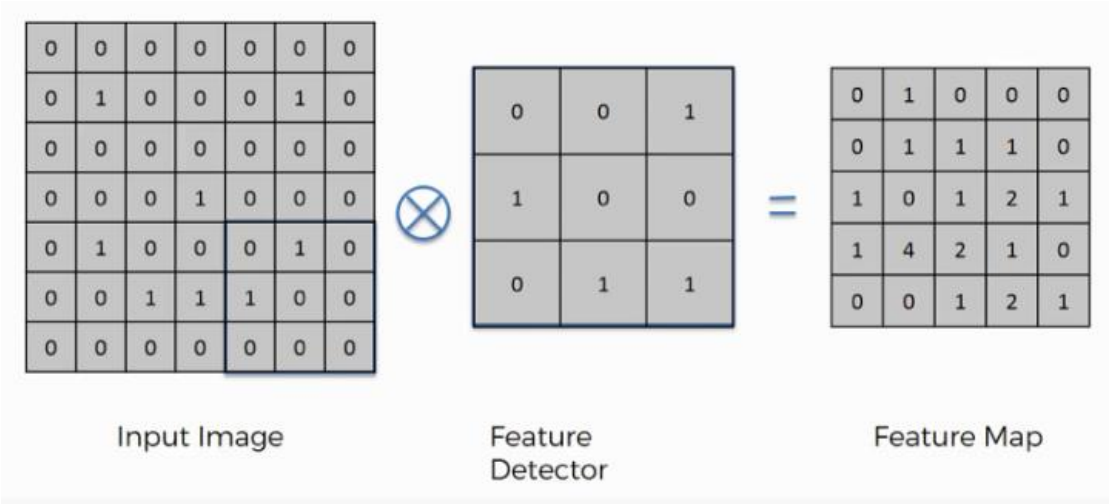

Figure 1 – Example of 2D Convolution

2.3. Max pooling

The max-pooling layer operates after each convolution layer. After getting the full convolutions using different filters, for any composition type the pooling choose the maximum value between several adjacent units window. Doing the max- pooling layer shrinks the size of the representation (the larger the window, the shrink is bigger) and if a low value to undesirable composition of words is given, the pooling also filters them out [3].

As in convolution, there are few types of max pooling:

- 1D max pooling – select the maximum value in every two unit window for every composition f

$$z_i^{(l,f)} = \max\left(z_{i-1}^{(l-1,f)}, z_{2i}^{(l-1,f)}\right) \ , l = 2,4, \dots$$

Equation 1 – 1D max pooling

- *2D max pooling* – select the maximum value in every 2 – dimensions unit window for every composition f. for example, in ARC-II, layer 2 performs a 2D max-pooling in non- overlapping 2 X 2 windows for every composition f

$$z_{i.j}^{(2,f)} = \max\left(z_{2i-1,2j-1}^{(2,f)}, z_{2i-1,2j}^{(2,f)}, z_{2i,2j-1}^{(2,f)}, z_{2i,2j}^{(2,f)}\right), l = 2,4,\ldots$$

Equation 2 - 2D max pooling

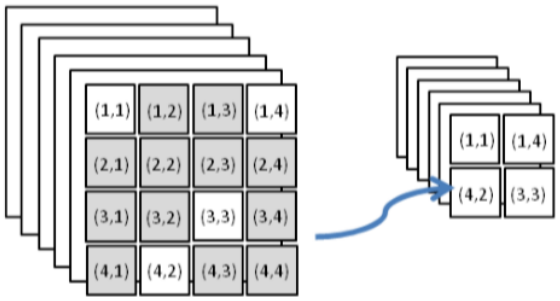

Figure 2 – Order preserving in 2D-pooling.

2.4. Length Variability

In order to use the convolution and pooling strategy in spite of the variable length of sentence, we put zero padding vectors after the last word of the sentence until the maximum length. To remove the effects caused by the great variability of sentence lengths, we add a gate that sets the output vectors to zeros if the input is all zero. This gate, working with max-pooling and positive activation function, keeps away the artifacts from padding in all layers [4].

$$g(\vec{V}) = \begin{cases} 0, & \vec{V} = \vec{0} \\ 1, & othewise \end{cases}$$

Equation 3 – Length variability gate function

2.5. Flattening

The output of the Convolution and max pooling layers is a polled feature map. The purpose of the flattening step is flatten the pooled feature map into a column (as illustrated in Figure 3) so it can pass through the neural network to have it processed further [2].

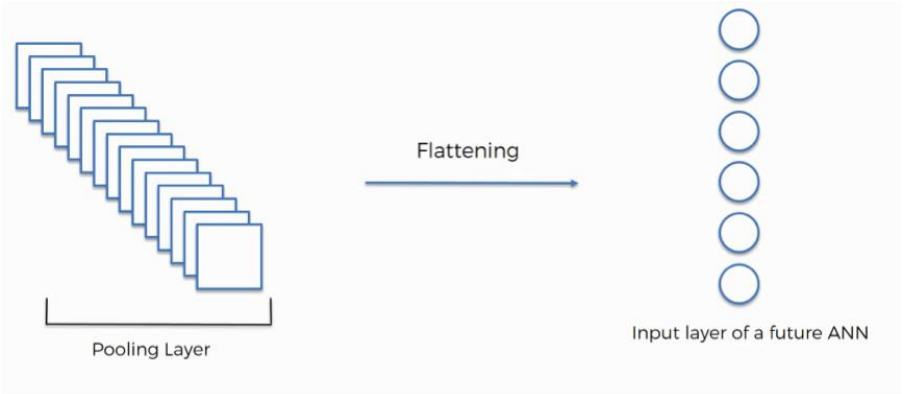

Figure 3 – Flattening

2.6. Fully connected

Fully connected is an artificial neural network, which its role is to take data and combine the features into a wider variety of attributes that make the convolutional network more capable of classifying data.

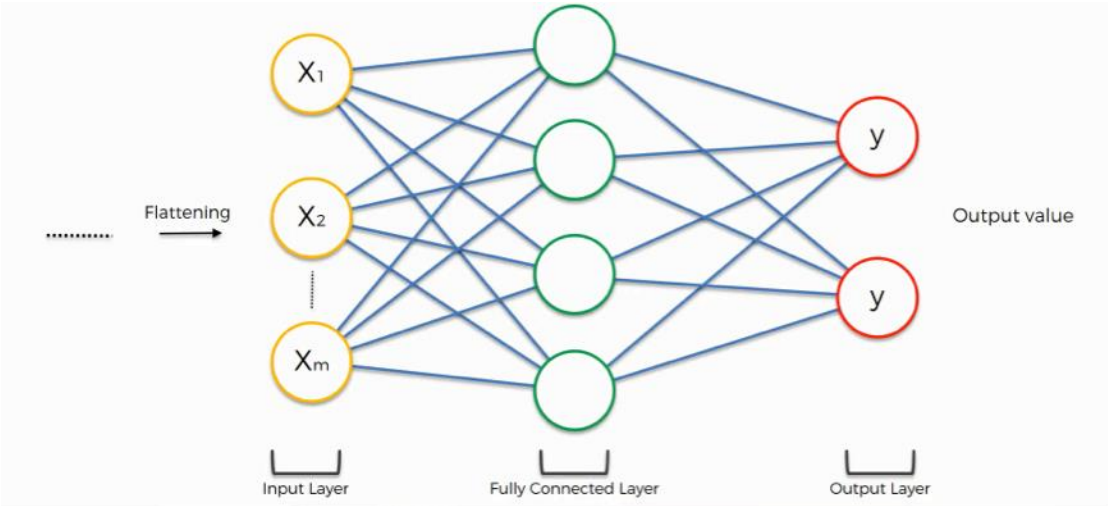

Figure 4 – Fully connected

The full connection step has three layers: input layer, fully- connected layer and output layer [5].

2.6.1.Input layer

Generally, the input layer of neural network communicates with the external environment that presents a pattern to the neural network. More specifically, in convolutional neural network, the input layer is the outcome of the flattening step. This input transferred to the fully connected layer. Every input neuron should represent some independent variable that has an influence over the output of the neural network.

2.6.2.Fully – connected layer

Fully connected layers are the layers that come after the input layer. They also called hidden layers because they are not directly expose to the input. The simplest network only have one hidden layer that directly outputs the value. In deep learning, we use many hidden layers. Any layer takes in a set of weighted inputs and produce an output through an activation function. It is a typical part of nearly any neural network.

2.6.3.Activation function

Activation function or transfer function is a simple mapping of summed weighted input to the output of the neuron. The first activation function was used to check if the summed input was above a threshold, for example 0.2, then the neuron would output a value of 1.0, otherwise it would output a 0.0.

Non-linear Activation Function is the most used activation function type. The nonlinear activation functions mostly divided by their range or curves.

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

Equation 4 – Example of non-linear activation function – Sigmoid

In our architecture, we use the most used activation function -the Rectified Linear Unit (ReLU) function. ReLU is the most commonly used activation function in deep learning models. It is simple and quicker than most activation function. A rectified linear unit has output 0 if the input is less than 0, and raw output otherwise. That is, if the input is greater than 0, the output is equal to the input [6].

$$f(z) = \max(0, x)$$

Equation 5 – ReLU function

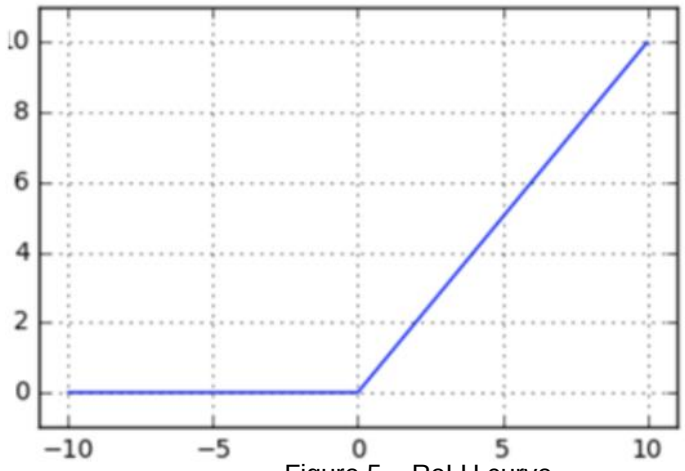

Figure 5 – ReLU curve

2.6.4.Output layer

The output layer of the neural network collects and transmits the information accordingly in way it has been designed to give. The number of neurons in output layer should be directly related to the type of work that the neural network was performing.

2.7. Dropout

Dropout method in neural networks is a technique where randomly selected neurons are ignored during training. This means that their contribute to the activation of downstream neurons is temporally removed in the forward pass and any weight updates are not applied to the neuron on the backward pass.

The effect is that the network becomes less sensitive to the specific weights of neurons. This is turn results in a network that capable of better generalization and is less likely to over fit the training data.

2.8. Word embedding

Many Machine Learning algorithms and almost all Deep Learning Architectures are incapable of processing strings or plain text in their raw form. They require numbers as inputs to perform any sort of job, like classification.

Other than that, word embedding is capable of capturing context of a word in a document, semantic and syntactic similarity, relation with other words, etc.

A word-embedding format generally tries to map a word using a dictionary to a vector. A dictionary may be a list of all unique words in the sentence. A vector representation of a word may be one-hot encoded vector where '1' stands for the position where the word exist and '0' everywhere else. This is just a very simple method to represent a word in the vector form.

One of the main goals of Word Embedding is to learn how different words are related to each other. The method takes the context of each word of the text as the input and tries to predict the word corresponding to the context.

More specifically, we use the one hot encoding of the input word and measure the output errors compared to the one hot encoding of the target word. In the process of predicting the target word, we learn the vector representation of the target word. The quality of these representations is measured in a word similarity task.

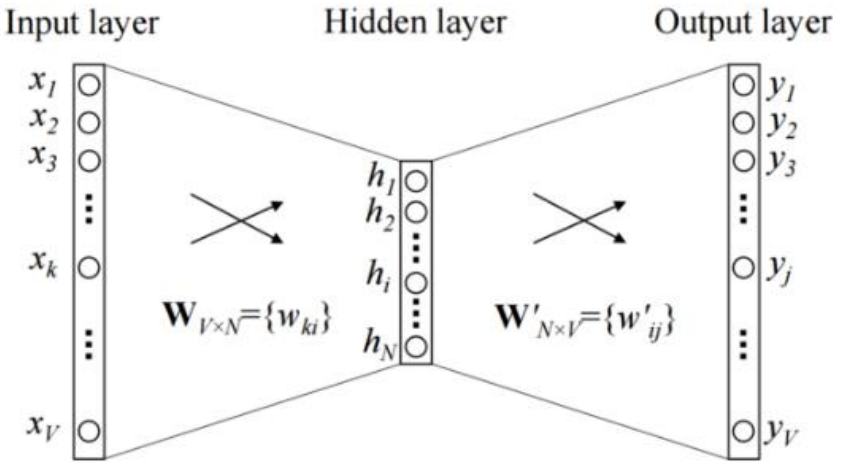

Figure 6 – Word embedding architecture

The input is one hot encoded vector of size V. The hidden layer contains N neurons and just copy the weighted sum of inputs to the next layer. There is no activation function like sigmoid, ReLU. The only non-linearity is the softmax calculation in the output layer that is a V length vector with the elements being the softmax values [7].

2.9. Bot

An Internet bot is software that performs an automated task over the Internet. More specifically, a bot is an automated application used to perform simple and repetitive tasks that would be time consuming or impossible for humans to do.

2.9.1. Twitter bot

A type of bot software that controls a Twitter account via the Twitter API. The bot software can autonomously perform actions such as tweeting, re-tweeting, liking, following, unfollowing, or direct messaging other accounts [8].

Figure 7 – Twitter bot profile

3. ARCHITECTURE

In this section, we present our algorithm. Our algorithm suggest a convolutional architecture for modeling sentences while our goal is to detect bots in Twitter using sentence structure. Our impression is that many big amount of tweets having similar structure can indicate a bot.

Generally, the architecture takes as input the embedding of words (often trained in advance with unsupervised methods) in the sentence represented as a matrix, and summarize the structure of a sentence through layers of convolution and pooling. The result is a fixed length vectorial representation in the final layer. The next step is compares the representation for the two sentences with a multi-layer perceptron (MLP) [4,9].

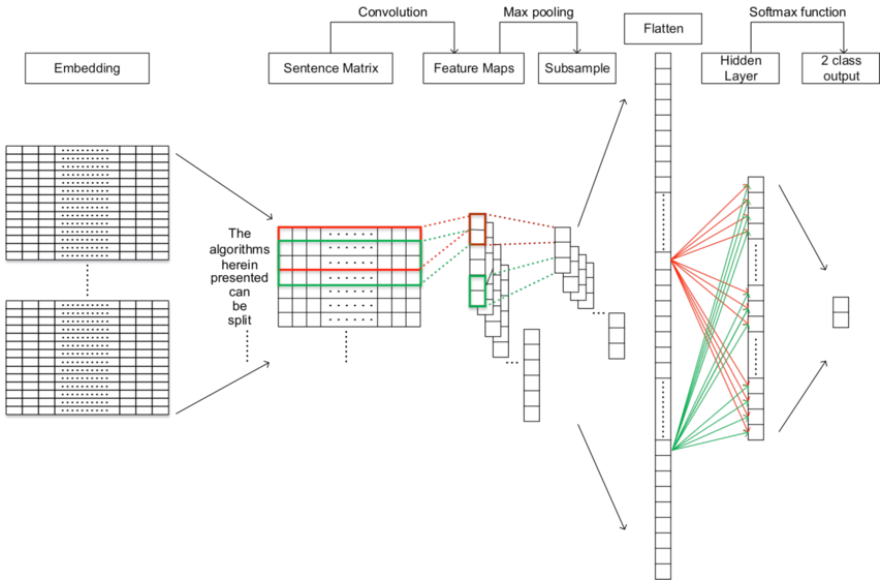

Figure 8 – Flowchart of the architecture

In more details, in our project, the user insert a tweet in order to find whether the tweet was written by a person or by a bot. As part of the training model process, we create a list of bot tweets from a database which each bot tweet in this list compared with the tweet the user has entered. The model first insert the two sentences into the same matrix by embedding.

The first layer slides filters on all the possible segments combinations of both sentences by 1D convolutions. For segment i on S_x and segment j on S_y we have

$$z_{i.j}^{(1,f)} = z_{i.j}^{(1,f)}(\mathrm{x},\mathrm{y}) = \mathrm{g}\left(\hat{z}_{i,j}^{(0)}\right) \cdot \sigma\left(w^{(l,f)}\hat{z}_{i,j}^{(0)} + b^{(l,f)}\right),$$

Equation 6 – Architecture - II general equation

Where:

- $z_{i,j}^{(l,f)}(x)$ gives the output of feature map type- f for location (i,j) in layer- l.
- $w^{(l,f)}$ is the parameters for f on layer l, with matrix $W^{(l)} = [w^{(l,1)}, \cdots, w^{(l,F_l)}]$.
- $\sigma(\cdot)$ is the ReLU activation function.
- $g\left(\hat{z}_{i,j}^{(0)}\right)$ is the length variability.
- $\hat{z}_{i,j}^{(0)}$ is concatenation of the segments of sentence S_x and S_y:

$$\hat{z}_{i.j}^{(0)} = [x_{i:i+k_1-1}^T, y_{j:j+k_1-1}^T]^T$$

Equation 7 – Concatenation of two segments

After this layer, we achieve a low-level representation of the interaction between the two sentences. The second layer is 2D max pooling in non – overlapping 2X2 windows (Equation 2).

From this layer we obtain a high-level representation, which encode the information from both sentences. The next layer perform a 2D convolution on k_3 X k_3 filters of output from the second layer:

$$z_{i.j}^{(3,f)} = \mathrm{g}\left(\hat{z}_{i,j}^{(2)}\right) \cdot \sigma\left(w^{(3,f)}\hat{z}_{i,j}^{(2)} + b^{(3,f)}\right)$$

Equation 8 – 2D convolution

This process proceed with one more layer of 2D convolution and 2D max pooling. Then we flatten the fixed length vector representation we obtained. This vector enters as an input to the multi- layer perceptron (MLP), which aims to compare the two sentences and as output, it give the degree of structural compatibility between them.

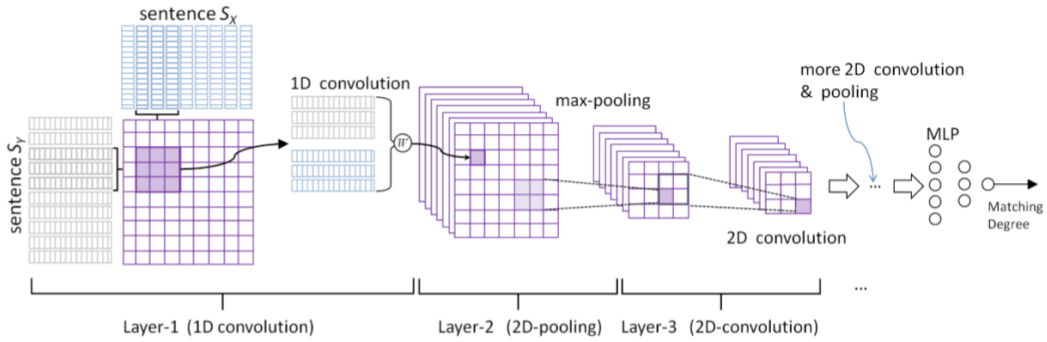

Figure 9 – Architecture-II of convolutional matching model

After we got the degree of structural compatibility between every two tweets, the bot tweet from the trained model list and the user tweet, we convert the degrees to classes (when zero means the two tweets have different sentence structure one means the opposite). The next step, we sum the classes Id and divide the amount by the size of the bot list in order to get the percentage of similarity between all the bot tweets. In case the percentage of similarity pass the threshold selected by the user, the model will identify the tweet as bot tweet.

The final model include the include the following layers:

1. Layer - 1
   a. Embedding (tweet A)
   b. Embedding (tweet B)
   c. Concatenate
   d. 1D- convolution
2. Layer – 2
   a. Reshape
   b. 2D – Max pooling
3. Layer – 3
   a. 2D – convolution
   b. 2D –Max pooling
4. Layer -4
   a. 2D – convolution
   b. 2D –Max pooling
5. Flatten Layer
6. Dropout Layer
7. Fully connected Layer with rectifier activation function (reLU)
8. Post processing to get the structural compatibility between every two tweets
9. If sum ≥ threshold
   return "bot tweet"

   else

   return "human tweet"

3.1. Tweet classification

Our assumption is that the tweets written by the same bot program may have the same sentence structure. In order to prove this assumption, we used a method called one class classification. In machine learning, one class classification tries to identify objects of a specific class among all objects.

In our model, class will be cluster of all the tweets written by the same bot. The learning process includes the distribution of a large amount of tweets to different classes according to their sentence structure. Using our architecture, we will associated the input tweet with a specific class.

4. SOFTWARE ENGINEERING DOCUMENTATION

4.1. Use Case

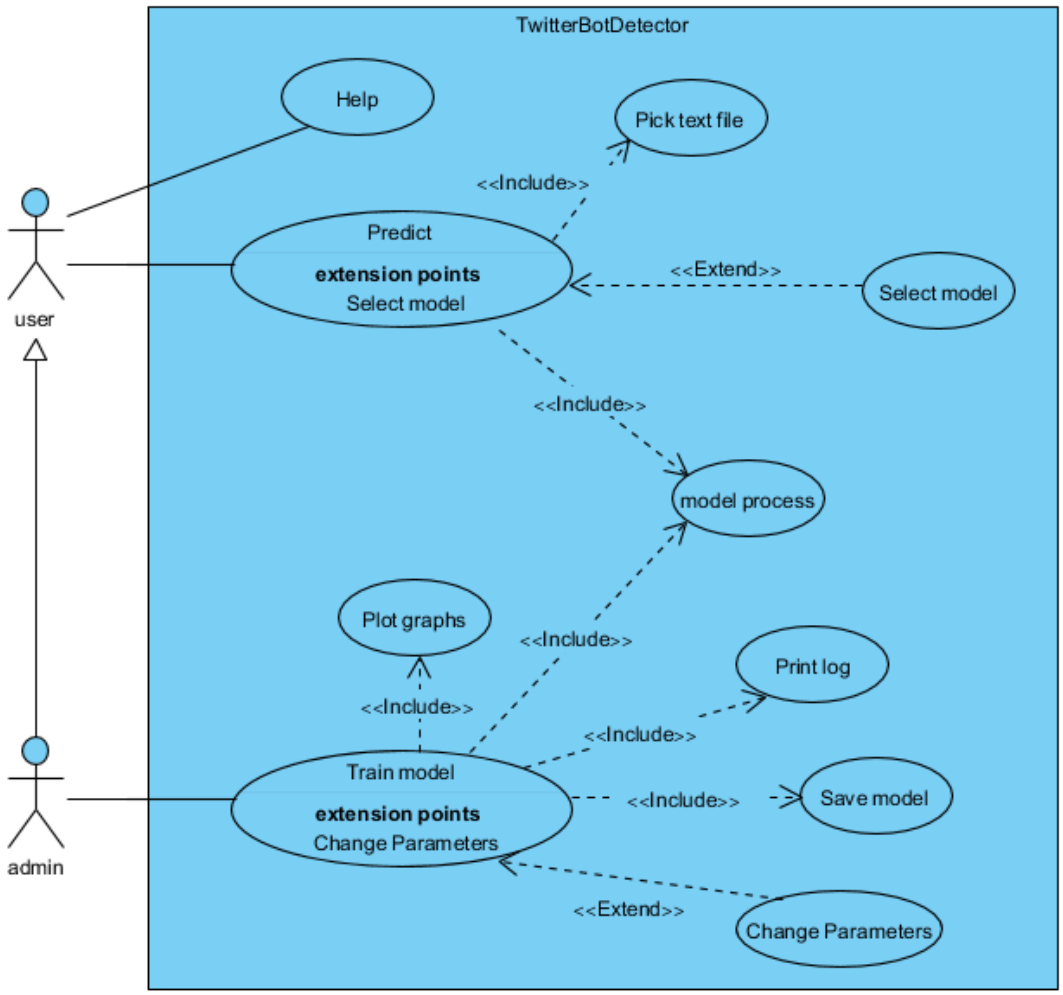

Figure 10 – Bot Detector Use Case Diagram

UC1: Predict

- Goal – run the model on a text file contains a tweet to get indication of the tweet was written by a bot.
- Preconditions – text file and learning model exist
- Possible errors – file size too big, invalid type file uploaded.
- Limitations – file size.
- Pseudo code:

Actor	System
Select the "Analyze" option	Open window for select model and text file upload
Upload text file	

Select model	
Click on "Predict" button	Insert text file to the CNN trained model
	Display an answer of the prediction model

Table 1. Predict

UC2: Train model with different settings

- Goal – setup and train the CNN.
- Preconditions – "Settings" option chosen, word2vec and database exist in db directory.
- Possible errors – word2vec or data set corrupted.
- Limitation - process time and CPU.
- Pseudo code:

Actor	**System**
Select the "Settings" option	Open setting window
Setup batch size, epoch number and percentage of data to use in training phase.	Run training process
Click "Start" button	Start training
	Display monitoring graphs and log
	Finish training, save the model and display complete message

Table 2. Train model with different settings

4.2. Design

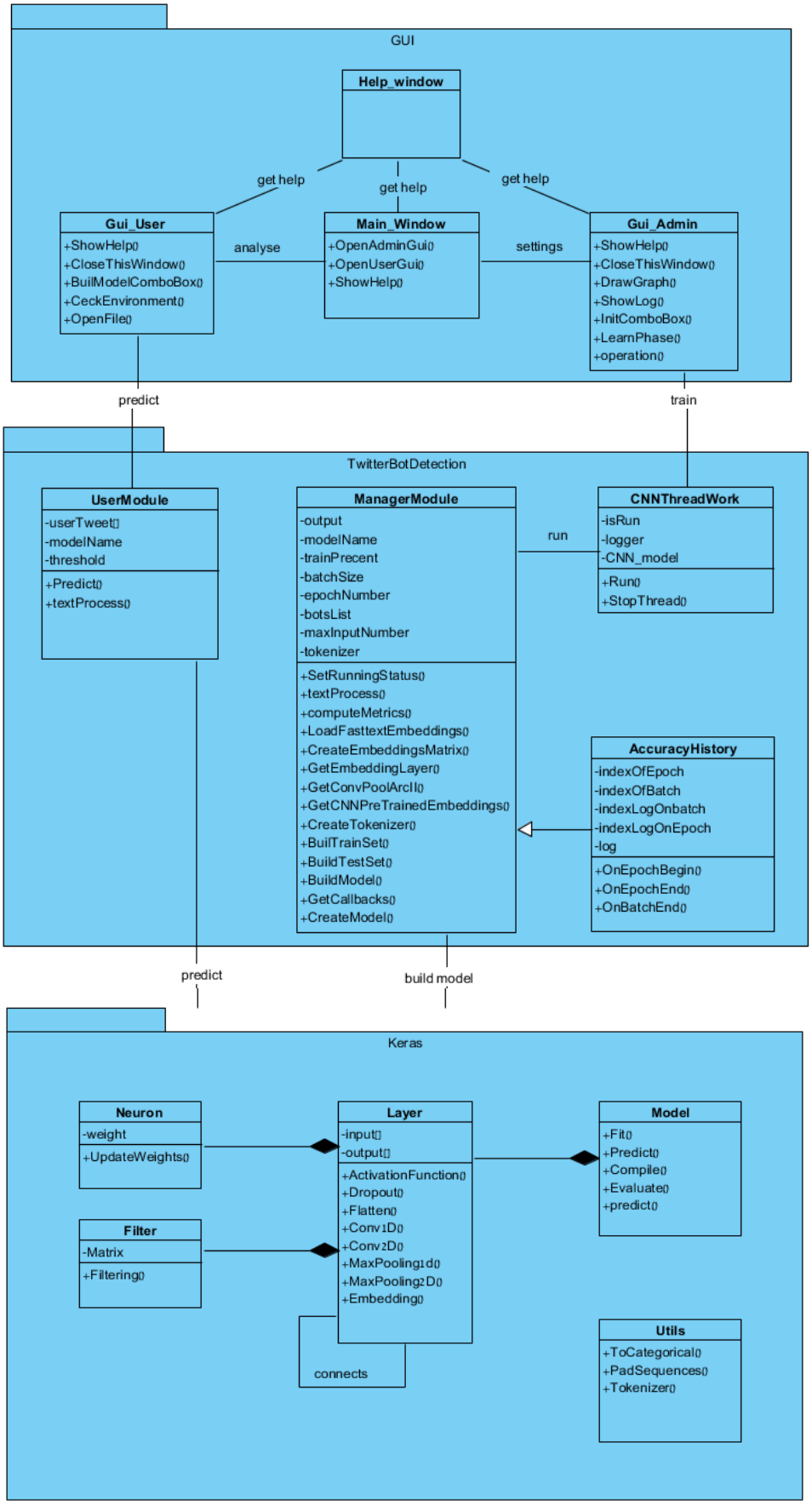

Figure 11 – Bot Detector Class Diagram

4.3. User Interface

4.3.1. Windows

Welcome window : this window gives the user two options: set and create a new model or analyze tweets on existing model.

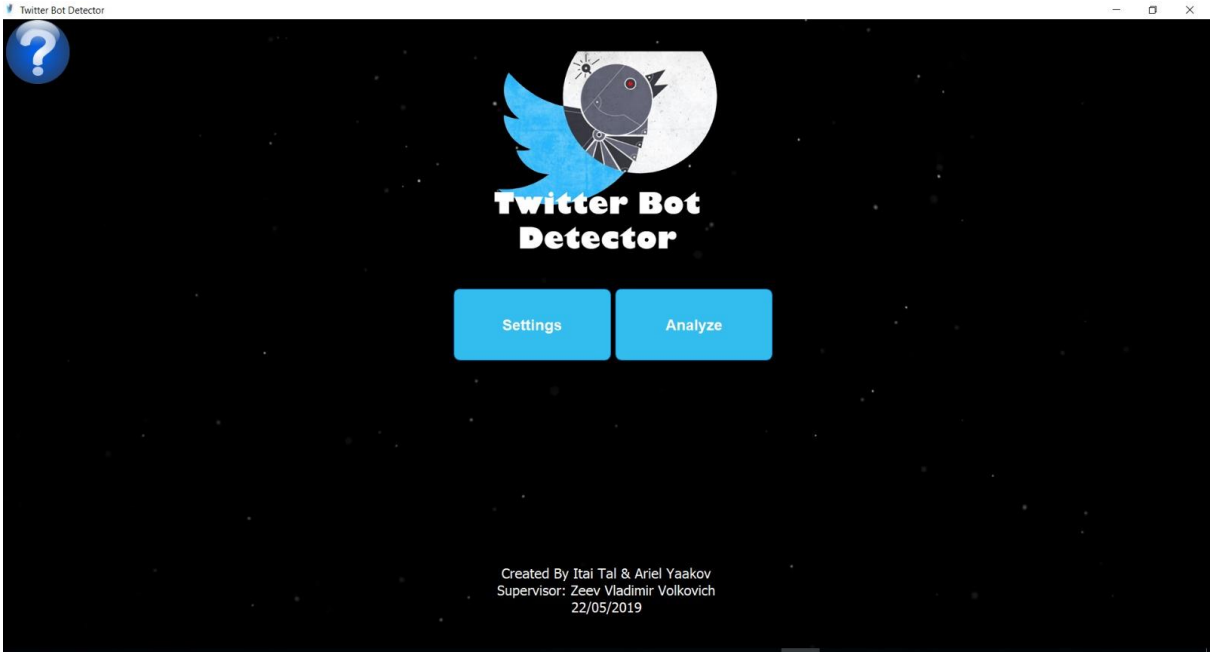

Figure 12 – Welcome window

User window: when the user press the "Analyze" button in the "Welcome window", he should select file-containing tweets that he wants to detect.

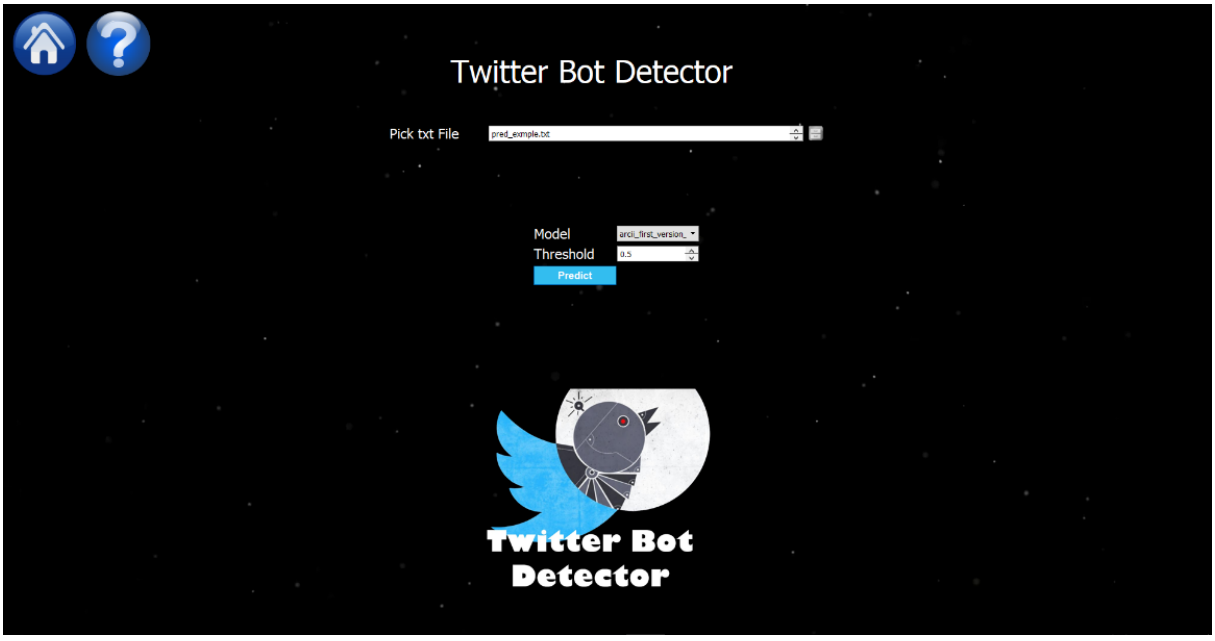

Figure 13 – User Window

Not a bot tweet window: when the user press the "Detect" button in the "Welcome window", after he choose the tweet file, the app determined that a human being wrote it.

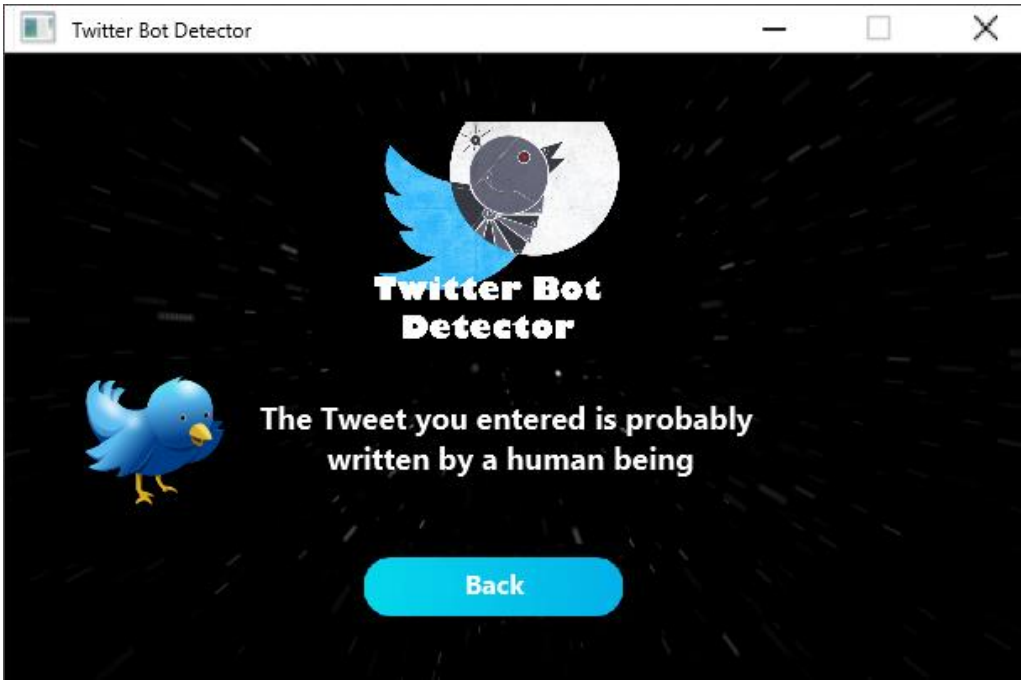

Figure 14 – Not a bot tweet window

Bot tweet window: when the user press the "Detect" button in the "Welcome window", after he choose the tweet file, the app determined that a bot wrote it.

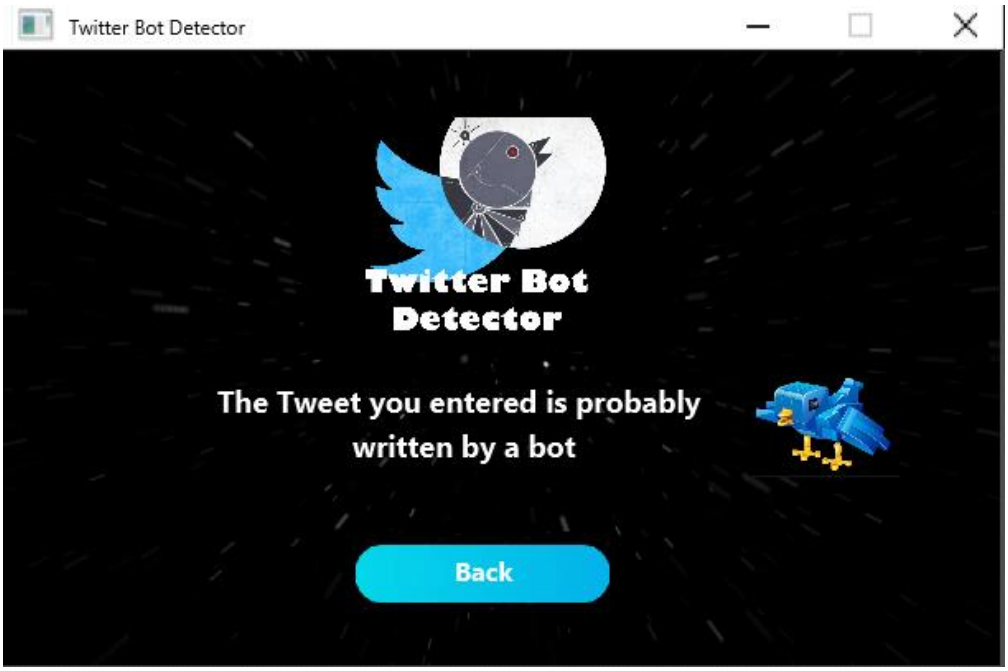

Figure 15 – Bot tweet window

Settings window: the admin have the option to create a new model by setting parameters and upload data set.

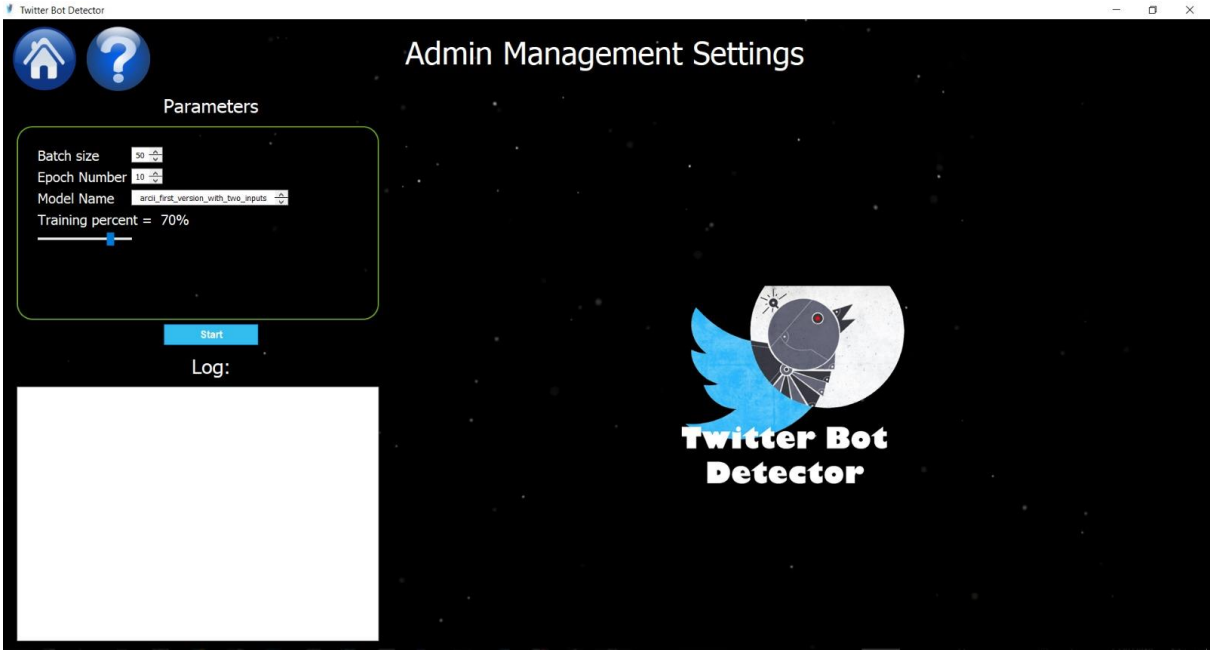

Figure 16 – Bot Detector Class Diagram

Monitor: User can see the results of learning process in real time.

Figure 17 – Settings window

Model successfully created window: in the “setting window” if the admin click on "Start", the model successfully create.

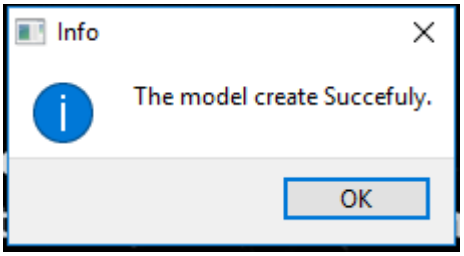

Figure 18 – Model successfully created window

4.3.2.Errors

Not Number batch size error: in the “setting window” if the admin click on "Start", if the user did not type number in batch size.

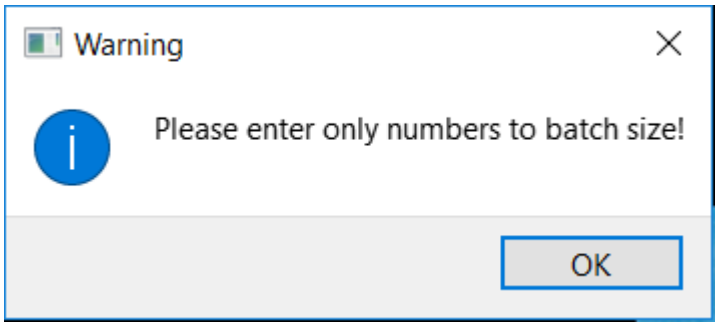

Figure 19 – Not Number batch size

Not Number epoch number error: in the “setting window” if the admin click on "Start", if the user did not type number in epoch number.

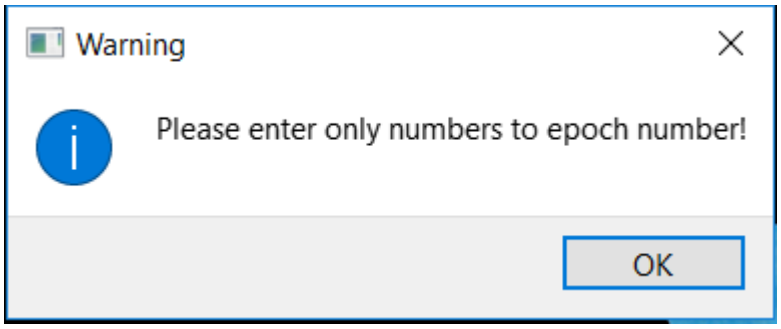

Figure 20 – Not Number epoch number

Incorrect epoch number error: in the “setting window” if the admin click on "Start", when the Batch size isn’t between 2 and 1000.

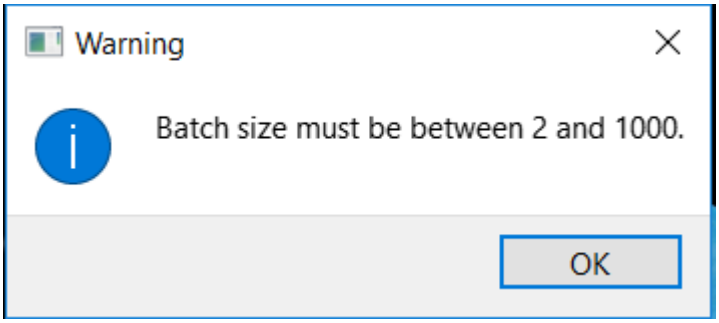

Figure 21 – Incorrect epoch number error

Incorrect batch size error: in the “setting window” if the admin click on "Start", when the epoch number isn’t positive.

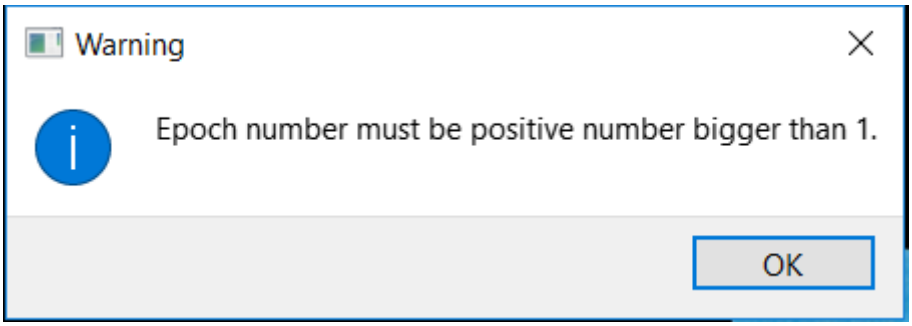

Figure 22 – Incorrect batch size error

Not predict file error: in the “User window” if the user click on "Predict", when he didn’t choose a predict file.

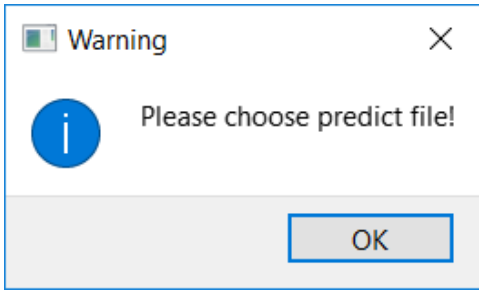

Figure 23 – Not predict file error

Not Number threshold error: in the “User window” if the user click on "Predict", when he entered not a number in the threshold.

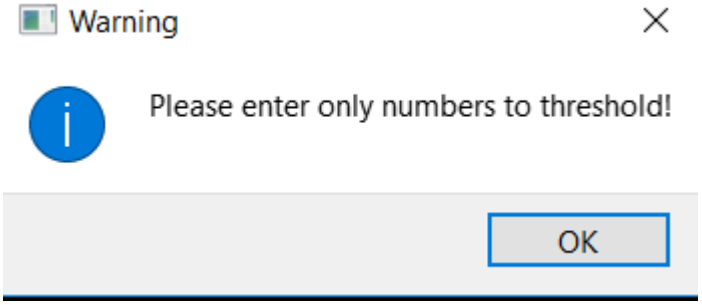

Figure 24 – Not Number threshold error

Incorrect threshold error: in the “User window” if the user click on "Predict", when he entered threshold not between 0 and 1.

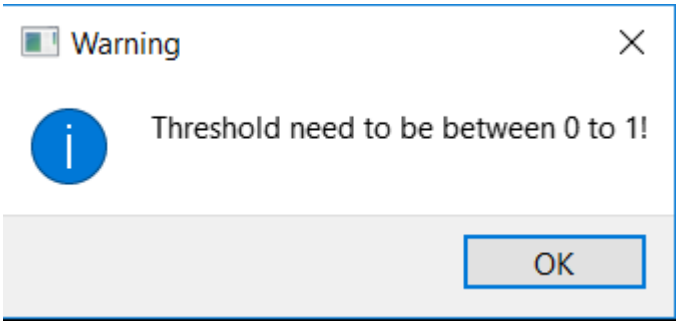

Figure 25 – Incorrect threshold error

Not model error: in the “User window” if the user click on "Predict", when he didn’t create model.

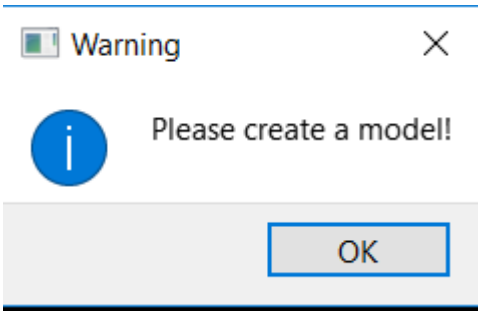

Figure 26 – Not model error

4.4. Testing plan

Test Name	Description	Expected results	Window
Valid Create model.	1) Insert 50 in batch size. 2) Insert 10 in epoch number. 3) Insert ‘model’ in model name. 4) Choose 70% in Training percent. 5) Click on “Start”.	Display a message “The model create successfully”.	Settings
Invalid Incorrect batch size.	1) Insert 1 in batch size. 2) Click on “Start”.	Display error message “Batch size must be between 2 and 1000.”	Settings
Invalid Incorrect epoch number.	1) Insert -1 in epoch number. 2) Click on “Start”	Display error message “ Epoch number must be positive number bigger than 1”	Settings
Invalid type batch size.	1) Insert ‘a’ in batch size. 2) Click on “Start”.	Display error message “Please enter only number to batch size”.	Settings
Invalid type epoch number.	1) Insert ‘a’ in epoch number. 2) Click on “Start”.	Display error message “Please enter only number to epoch .number”	Settings
Valid prediction.	1) Click on the file chooser Choose a .txt file with 1 tweet and click ok. 2) Insert 1.1 in threshold field	Display a message with That show what is the tweet.	User

	2) Click on “Predict”.		
Invalid no prediction source file.	1) Click on the file chooser Choose a .csv file with 1 tweet and click ok. 2) Click on “Predict”.	Display error message “Please choose predict file!”	Load tweet
Invalid Incorrect threshold.	1) Click on the threshold field and insert 2.0 2) Click on “Predict”.	Display error message “threshold need to be between 0 and 1!”.	User
Invalid no model.	1) don’t choose a model. 2) Click on “Predict”.	Display error message “Please create a model!”.	User

Table 3. Testing plan

5. RESULTS AND CONCLUSIONS

5.1. Results

5.1.1.Experiment 1

In this experiment, we test 268 tweets when the threshold changes to draw conclusions about appropriate threshold selection for different tasks. The test set contains 144 tweets written by bots and 124 tweets written by humans. For this experiment we used our pre-trained model with the following parameters: batch size – 50, epoch number – 10, training set – 70%, pre-trained word2vec – wiki-news-300d-1M.

Test 1: threshold = 50%

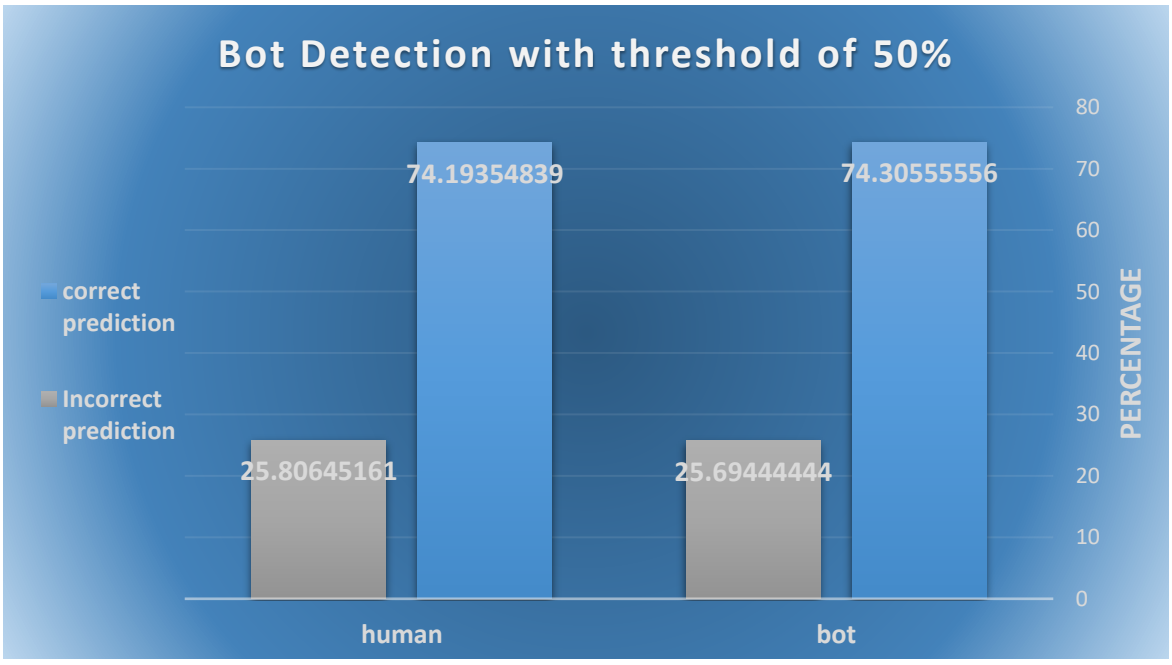

Figure 27 – threshold of 50% diagram

It can be seen that with a threshold of 50% the model predicted about 75% of the tweets written by humans as well as tweets written by bots as such.

actual\predicted	Bot	human
bot	107	37
human	32	92

Table 4 – threshold of 50% confusion matrix

Test 2: threshold = 33%

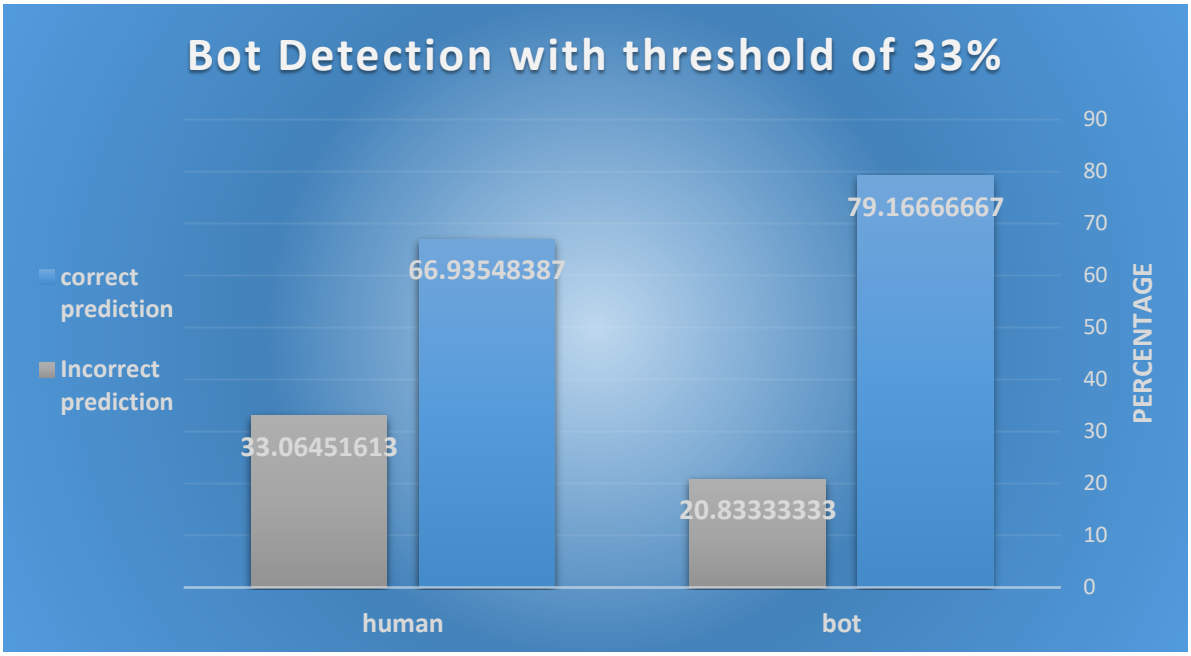

Figure 28 – threshold of 33% diagram

In this test, we can see that the degree of prediction is different between the two categories when the accuracy of the prediction of bot tweets comes at the expense of the human tweets prediction. When we tried to conclude from these results whether the user would be advantage to use this threshold, it occurred to us that with this threshold we will be able to detect offensive and malicious bots on Twitter.

actual\predicted	bot	human
bot	114	30
human	41	83

Table 5 – threshold of 33% confusion matrix

Test 3: threshold = 75%

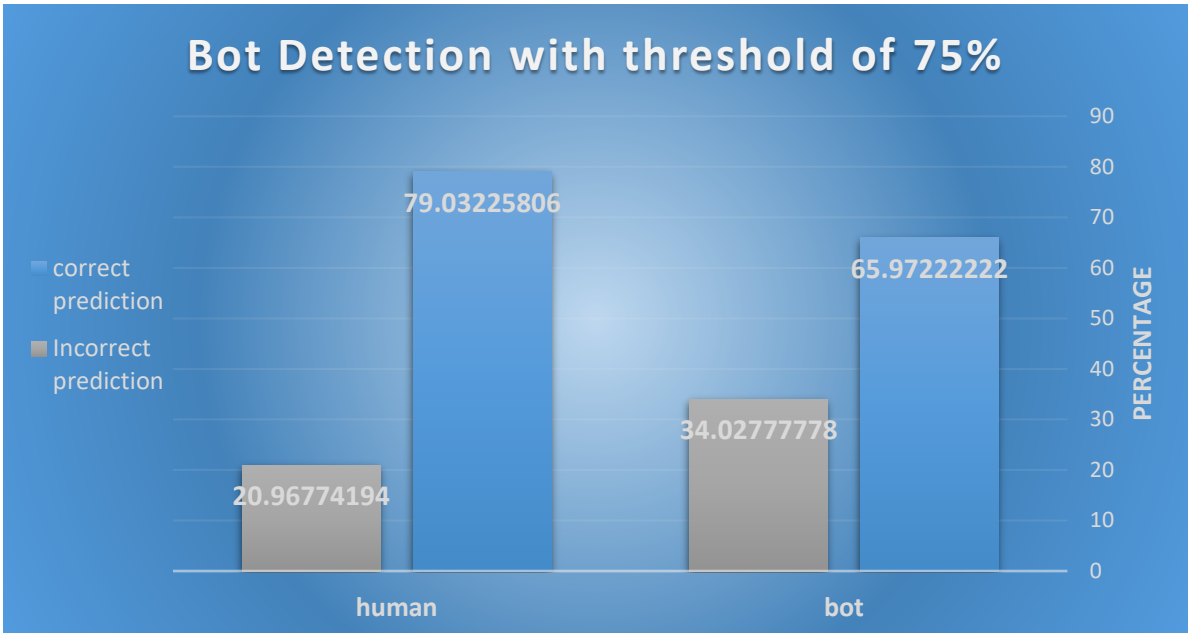

Figure 29 – threshold of 75% diagram

With threshold of 75%, we can see the same tendency as the previous test but here the accuracy of the prediction of human tweets comes at the expense of the bot tweets prediction. With this threshold most of the human will be detected as such.

actual\predicted	bot	human
bot	95	45
human	26	98

Table 6 – threshold of 33% confusion matrix

5.1.2.Experiment 2

In this experiment, we examined the possibility to change the pre-trained word vectors. For comparison, we used two vectors files for word representation. The first file is 'glove twitter' with 50 dimensions and the second is 'wiki news' with 300 dimensions. We test 268 tweets when 144 of them written by bots and 124 tweets written by humans. For this experiment, we used two pre-trained model with the following parameters: batch size – 50, epoch number – 10, training set – 70%, threshold – 50%.

Test 1: word2vec – glove-twitter-50d:

actual\predicted	bot	human
bot	85	59
human	58	66

Table 7 – glove-twitter-50d confusion matrix

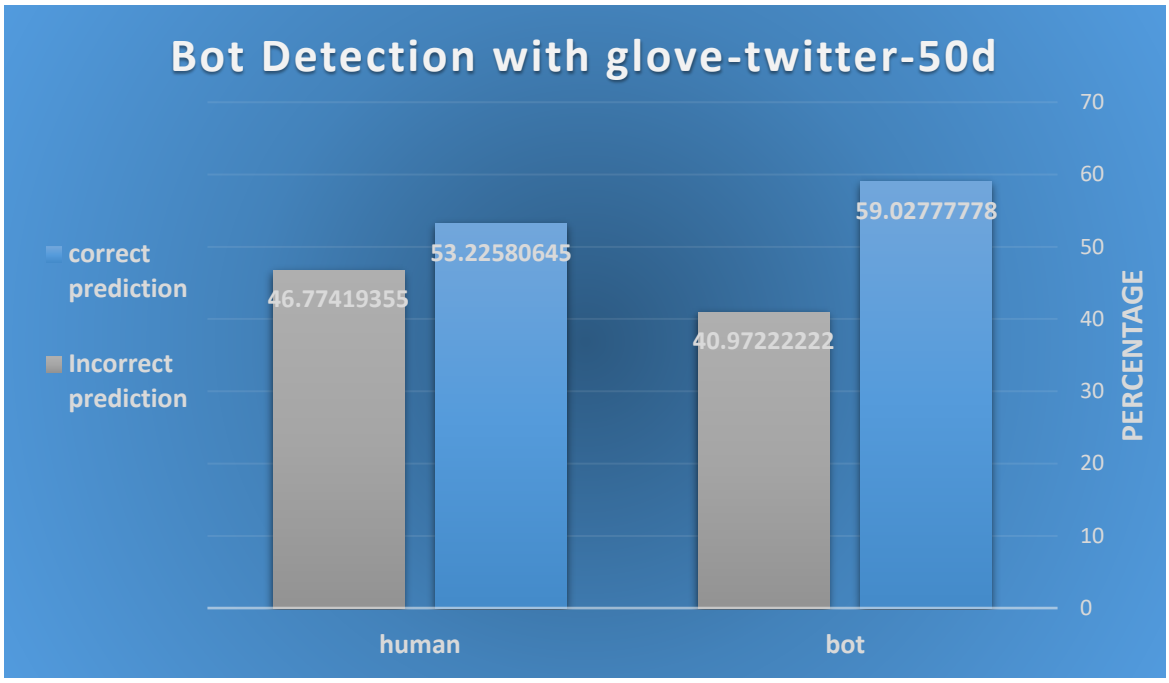

Figure 30 – glove-twitter-50d diagram

Accuracy = 56%

Precision = 59%

Recall = 59%

We can see that this model will give us vague and inaccurate information. This information can not satisfy us in order to reach the goal of detection bots on Twitter.

Test 2: word2vec – wiki-news-300d-1M:

actual\predicted	Bot	human
bot	107	37
human	32	92

Table 8 – wiki-news-300d confusion matrix

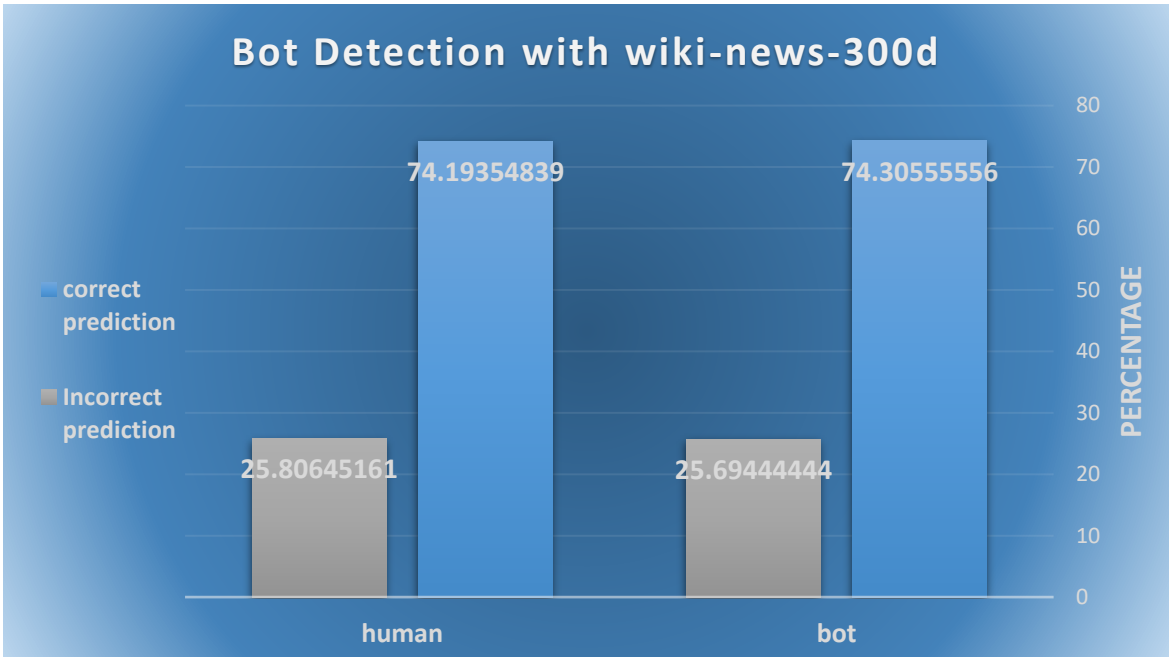

Figure 31 – wiki-news-300d diagram

Accuracy = 74%

Precision = 74%

Recall = 77%

As we can see, this model is much better than the previous model in terms of accuracy, precision and recall. The model provides us much more accurate information that helps us to conclude whether a tweet was written by bot or by a person.

5.2. Conclusions

At the experiment stage, we have created many models with different parameters, trying to understand better how the model responds to the various parameters values. We learned from the monitoring window in the learning process that small epoch number gives quite low accuracy and on the other hand, a relatively big epoch number causes over fitting.

According to the first experiment, it can be seen that different threshold value gives different degree of prediction whether it's tweets written by human or by bots as we can see in figure 32. After deep thinking, we concluded that a different threshold value could contribute to various purposes of the application and therefore we gave the user the option to change the threshold value according to the purpose for which he use the application.

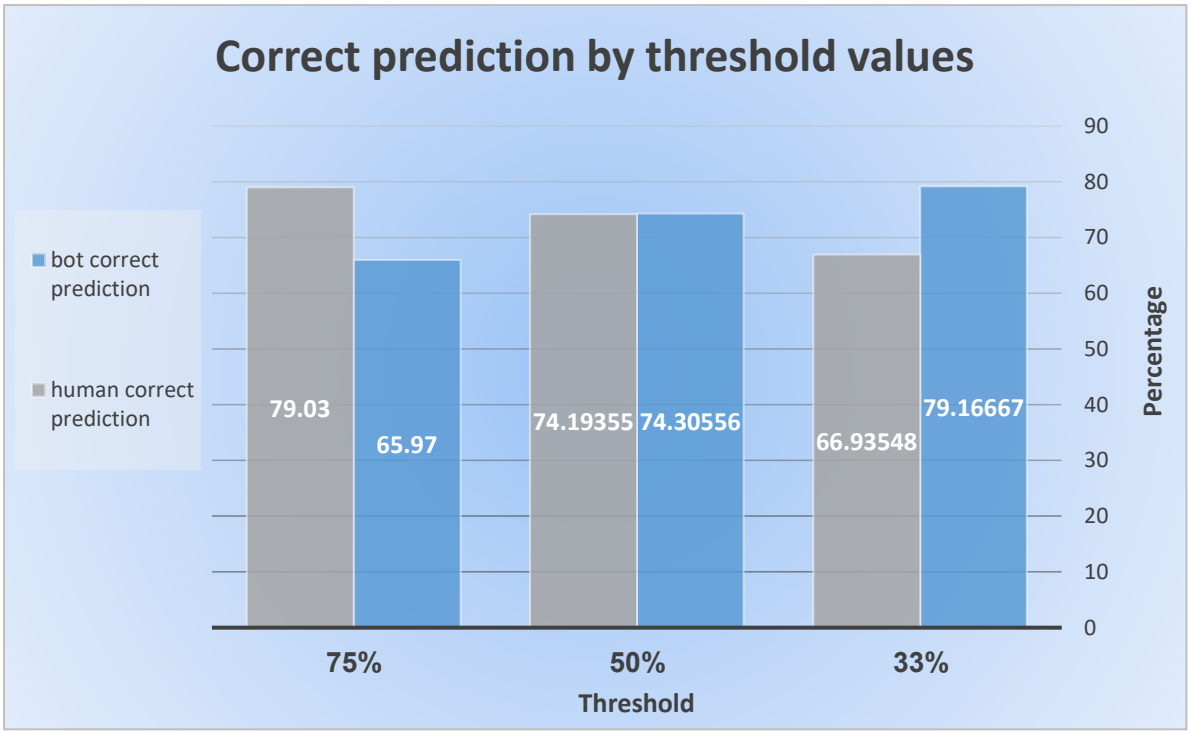

Figure 32 – Correct prediction by threshold values diagram

From experiment 2, we learned that the pre-trained word2vec model is clearly an important parameter, which effects the bot detection performance in terms of accuracy, precision and recall.

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, taking into account a 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 and inflammatory bots on Twitter.

6. REFERENCES

[1] Y. Le Cun and Y. Bengio, *the handbook of brain theory and neural networks*, MIT Press, 1998 ("Convolutional Networks for Images Speech, and Time- Series", pp 255-258).

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

[3] N. Kalchbrenner, E. Grefenstette and P. Blunsom, "A Convolutional Neural Network for Modelling Sentences", *Association for Computational Linguistics (*ACL) 1: 655–665, 2014.

[4] B. Hu , Z. Lu, H. Li and Q. Chen, "Convolutional Neural Network Architectures for Matching Natural Language Sentences", *Neural Information Processing Systems (NIPS)*, 2014.

[5] J. Long, E. Shelhamer and T. Darrell, "Fully Convolutional Networks for Semantic Segmentation", *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 2017.

[6] F. Agostinelli, M. Hoffman, P. Sadowski and P. Baldi, "Learning Activation Functions to Improve Deep Neural Networks", *ArXiv*, 2014.

[7] T. Mikolov, I. Sutskever, K. Chen, G. Corrade and J. Dean, "Distributed Representations of Words and Phrases and their Compositionality", *International Conference on Neural Information Processing*, December 2013.

[8] J.Messias, L. Schmidt, R. Oliveira and F. Benevenuto, "You followed my bot! Transforming robots into influential users in Twitter", *First Monday*, 2013.

[9] Y. Kim, "Convolutional Neural Networks for Sentence Classification", *Empirical Methods in Natural Language Processing (EMNLP)*, 2014.

[10] Z. Chu, S. Gianvecchio, H. Wang ang S. Jajodia, "Who is tweeting on Twitter: human, bot, or cyborg?", *Computer Security Applications Conference*, 2010