UDC 004.89, 004.912

Y. Bezliudnyi, V. Shymkovych,
P. Kravets, A. Novatsky, L. Shymkovych

PRO-RUSSIAN PROPAGANDA RECOGNITION AND ANALYTICS SYSTEM BASED ON TEXT CLASSIFICATION MODEL AND STATISTICAL DATA PROCESSING METHODS

Abstract: In this paper a neural network model for classifying the political polarity of text has been developed, along with a database for training the neural network and an analytics system for pro-Russian propaganda. This allows to classify the political polarity of the message source based on its identifier, as well as to construct and display different networks that represent useful insights about popular Twitter hashtags or Telegram channels that related to Russo-Ukrainian War. Also, a user interface has been developed that allows users to interact with the system.

Developed system will help people with navigation through the information space and avoidance of pro-Russian propaganda.

Keywords: neural networks, text classification, text transformers, news, propaganda, hashtags, networks, Telegram, Twitter.

Introduction

Propaganda is the dissemination of certain views by conveying various arguments, truthful or semi-truthful facts, rumors, or outright lies with the aim of manipulating public consciousness. Many methods based on social psychology research are used to carry out propaganda, making it difficult for ordinary citizens to recognize the type of propaganda they are exposed to while reading various mass media sources. Additionally, some segments of the population are more susceptible to propaganda, such as people without higher education, children and pensioners, emotional individuals, and some residents of southern and eastern regions of Ukraine [1].

Therefore, it would be useful to develop a software system that could help verify the information posted by users on social media and messaging platforms for political polarity and pro-Russian rhetoric. It’s also useful to provide insights about which Telegram channels or Twitter hashtags are pro-Ukrainian or pro-Russian and how they are interconnected.

Such system would help people navigate the information space more easily and avoid unwanted pro-Russian propaganda.

Given the modern format of messages, replacement of a single word, emphasis in a sentence, or the presence of an emoji in the text can change the message's meaning, which should also be considered when developing such systems. Messages can also contain complex verbal constructions, such as irony or sarcasm, which complicates the task of detecting propaganda [2].

Over the past decade, artificial neural networks have made significant progress in such tasks as semantic analysis, classification, translation, text generation, and more [3-8].

Natural language processing requires a special architecture for neural networks to meet the following requirements arising from the characteristics of text data:

- Support of variable-length data sequences;
- Tracking dependencies of different distances between data;
- Support of information about the sequence of data arrival.

The simplest model that meets all of the above requirements is the classic recurrent neural network (RNN). Over time, this model has undergone improvements, giving rise to models such as long short-term memory (LSTM) and gated recurrent units (GRU).

In recent years, models based on RNN have faced competition from models based on the deep architecture of the Text Transformer. Unlike RNN, the Transformer does not require sequential processing of data. Instead, the attention mechanism provides context for any position in the input sequence. For example, if the input data is a natural language sentence, the Transformer does not need to process its beginning before processing its end. Instead, the Transformer determines the context that gives meaning to each word in the sequence. This property enables task parallelization, reducing training time.

Since the release of the Transformer model, a number of models have been developed that use it or its parts as a basis for their architecture, such as BERT, GPT, CTRL, XLM, XLNET, T5, REFORMER, LONGFORMER, ELECTRA, and more.

Currently, there are several research studies focused on text classification using neural networks. For example, researchers Rao A. and Spasoevich N. [9] used a technique that involves word embeddings and an LSTM model to classify political polarity of messages collected from the Twitter, categorizing them as either democratic or republican. Their developed model achieved high accuracy of 87.57% on a test set. They also deployed a system that uses the trained model and provides access for integration purposes.

In another study, researchers from the Monterrey Institute of Technology [10] classified the sentiment (as positive or negative) of Twitter messages related to the Russo-Ukrainian War in different countries worldwide using the VADER [11] sentiment analyzer and an RNN model. This approach achieved good accuracy of 93% on the validation set and near 90% accuracy on the test set.

The aim of this work is to automate the process of identifying the political bias of texts using neural networks, build a database for training the model, and develop software tools based on the trained model.

Classification model construction

To solve the task of text classification, various neural network models can be used, ranging from relatively simple RNN models to deep models based on text transformers. The choice of a particular model depends on the details of the specific task and the desired accuracy of the result.

Recurrent neural networks are a class of artificial neural networks specifically designed for processing sequential data. RNNs are widely used in many areas, such as natural language processing, speech recognition, image captioning, text and audio generation, due to their ability to model temporal dependencies in sequential data.

At the core of RNN is a set of recurrent connections that allow the network to store memory of previous input data. The network takes a sequence of vectors as input data. RNN processes the sequence step by step, updating its hidden state at each step, using the current input vector and the previous hidden state.

The hidden state of an RNN can be considered as a compressed representation of the entire sequence of input data observed up to that point in time. This compressed representation can be used for predicting the next element in the sequence or for classifying the entire sequence.

Depending on the task at hand, an RNN model can have one or multiple inputs and outputs.

RNN models are typically trained using the backpropagation through time (BPTT) algorithm, which is a variant of the backpropagation algorithm adapted for sequential data.

The LSTM model is a special type of RNN architecture that is capable of learning long-term dependencies because the memory, which is implemented by the classical RNN model, is short-lived – at each training step, the information in the memory is combined with new information and is completely overwritten after a few iterations.

The LSTM model does not use activation functions inside its recurrent components. Thus, the stored value is not diluted over time, and the gradient does not disappear when using BPTT.

The key components of the LSTM module are the cell state and various filters. The cell state can be thought of as the memory of the network that carries relevant information throughout the module chain. Therefore, information from earlier steps will be present in much later steps, negating the effect of short-term memory.

As training progresses, the cell state changes, and information is added or removed from the cell state by structures called filters.

Filters control the flow of information at the module's inputs and outputs based on certain conditions. They consist of a layer of sigmoidal neural network and a pointwise multiplication operation.

The sigmoidal layer returns value in the range [0;1], which indicate what portion of each block of information should be allowed to pass through the network. Multiplication by this value is used to allow or prohibit the flow of information inside and outside of the memory. For example, the input filter controls the degree to which a new value enters the memory, while the forget filter controls the degree to which a value is retained in the memory. The output filter controls the degree to which the value in the memory is used in the calculation of the output activation function.

The Transformer model is designed to process sequences of variable length inputs without relying on RNN. Instead, the Transformer model relies on the self-attention mechanism, which allows it to capture the dependencies between different positions in a sequence.

The architecture of the Transformer is shown in Fig. 1.

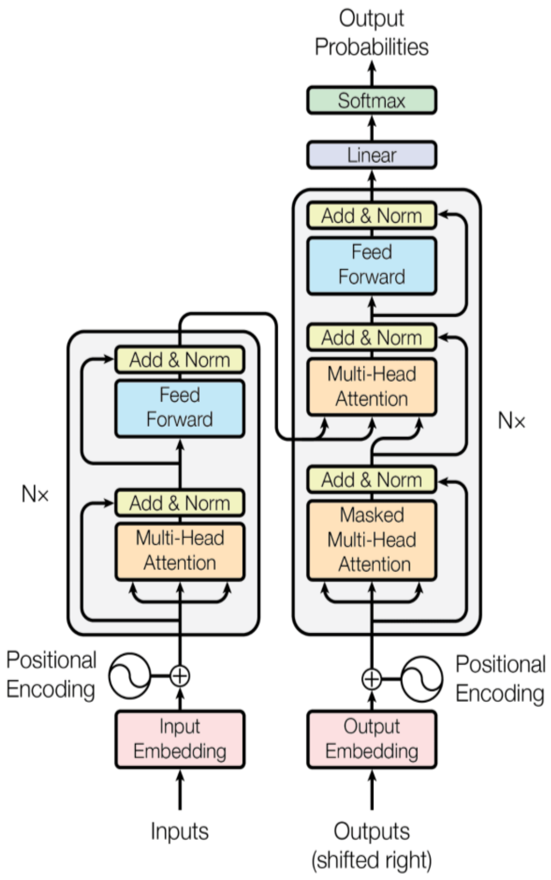

Figure 1. The architecture of the Transformer

The encoder consists of encoding layers that process the input data sequentially one layer at a time, while the decoder consists of decoding layers that do the same with the output of the encoder. The function of each encoder layer is to create a representation that contains information about which parts of the input data are relevant to each other. Each encoder layer passes its representation to the next encoder layer as input. Each decoder layer does the opposite, taking all the representations and using their contextual information to generate the output sequence. To do this, both the encoder and decoder levels use an attention mechanism. For each part of the input data, the attention mechanism computes the importance of each other part and uses it to obtain the result. Each decoder layer has an additional attention mechanism that receives information from the outputs of the previous decoders before the decoder layer receives information from the encoder representations. Both the encoder and decoder have a feedforward neural network for additional processing of the outputs and contain residual connections and normalization layers.

The attention function can be described as a mapping from a query and a set of key-value pairs to output data, where the query, keys, values, and output data are all vectors. The output is computed as a weighted sum of values, where the weight assigned to each value is computed using the compatibility function of the query with the corresponding key. In practice, the attention function is computed on a set of queries that are packed together in a matrix Q. Keys and values are also packed together in matrices K and V, respectively. The output matrix A is computed using the following formula:

$$A(Q,K,V) = softmax\left(\frac{QK^T}{\sqrt{\dim(K)}}\right)V \tag{1}$$

Instead of computing a single attention function using d_{model}-dimensional keys, values, and queries, the authors of the Transformer model found it beneficial [12] to linearly project the queries, keys, and values h times with different learned linear projections to d_k, d_k, and d_v dimensions, respectively. For each of these projected versions of queries, keys, and values, the attention function is computed in parallel, yielding a d_v-dimensional output value. These values are concatenated and then projected again, resulting in the final values of the multi-headed attention function according to the following formula:

$$MHA(Q,K,V) = Concat(head_1, \ldots, head_h)W^O \ \#(2)$$

The values of $head_i$ are determined by the following formula:

$$head_i = A\left(QW_i^Q, KW_i^K, VW_i^V\right) \ \#(3)$$

In these formulas, W_i^Q, W_i^K, and W_i^V are parameter matrices that define linear projections for the matrices Q, K, and V onto dimensions d_k, d_k, and d_v, respectively. W^O is a parameter matrix that defines the linear projection of the concatenated output values of the attention function of each layer of the encoder to the output dimension d_{model}.

A comparison of the blocks for computing the functions of classic and multi-head attention is shown in Fig. 2.

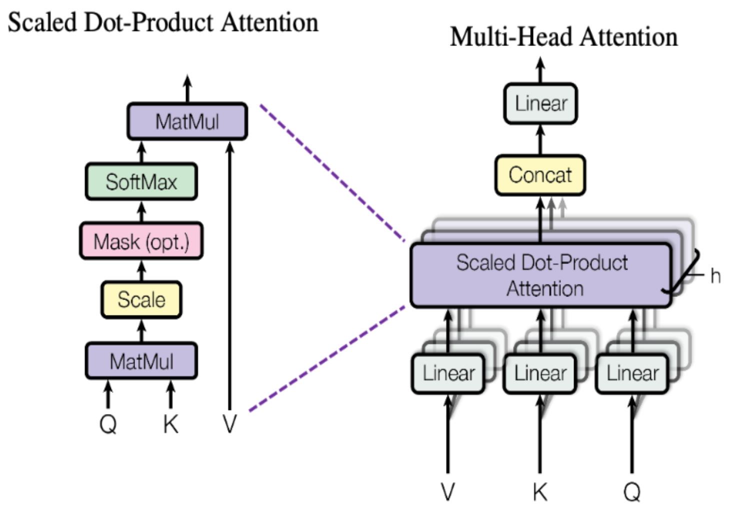

Figure 2. A comparison of the blocks for computing the functions of classic and multi-head attention

This paper presents a neural network model for classifying the political polarity of text, based on the deep Bidirectional Encoder Representations from Transformers (BERT) model developed by Google in 2018 [13]. BERT is a modified version of the Transformer model, which learns contextual relationships between words in text. In its classic form, the Transformer consists of two separate parts – an encoder and a decoder, but since BERT's goal is to create a language model, only the encoding mechanism is needed.

The key technical innovation of BERT is the use of bidirectional training. This approach differs from previous models that considered text sequences from left to right or combined left-to-right and right-to-left training. The results show that a language model trained bidirectionally has a deeper understanding of linguistic context than unidirectional language models.

To implement bidirectional learning, the BERT model uses two strategies: constructing a Masked Language Model (MLM) and predicting the next sentence (Next Sentence Prediction, NSP).

The MLM strategy involves replacing 15% of the words in each input sequence with the [MASK] token before feeding them to BERT. The model then tries to predict the original values of the masked words based on the context provided by the other unmasked words.

The algorithm for predicting the masked words is as follows:

1. A classification layer is added on top of the encoder output;

2. The output vectors are multiplied by a projection matrix to convert them to the vocabulary dimension;

3. The probabilities of each word in the vocabulary are computed using the softmax function.

The BERT loss function takes into account only the prediction of masked values and ignores the prediction of unmasked words.

The NSP strategy involves BERT receiving pairs of sentences as input during training and learning to predict whether the second sentence in the pair is the next sentence in the input text. During training, 50% of the input data consists of pairs where the second sentence is the next sentence in the input text, while the other 50% randomly selects the second sentence from a pool of sentences.

To help the model distinguish between two sentences during training, the input data is processed as follows:

1. The [CLS] marker is inserted at the beginning of the first sentence, and the [SEP] marker is inserted at the end of each sentence.

2. A sentence projection vector, indicating either sentence A or sentence B, is added to each token.

3. A position projection vector is added to each token to indicate its position in the sequence.

To predict whether the second sentence is truly related to the first, the following steps are taken:

1. The entire input sequence passes through the BERT model.

2. The output from the [CLS] marker is transformed into a vector of shape 2x1 using a simple classification layer.

3. The probability of the second sentence following the first is calculated using the softmax function.

During BERT model training, the MLM and NSP strategies are executed together to minimize the combined loss function of both strategies.

To use the BERT model in solving specific tasks, it is first pre-trained on a large text corpus. Standard text corpora for pre-training the BERT model include the WikipediaCorpus (~4.4 million articles) and the BookCorpus (~11,000 books).

After pre-training, the model is fine-tuned on the given dataset. The pre-training and fine-tuning process for the BERT model for a specific task is depicted in Fig. 3.

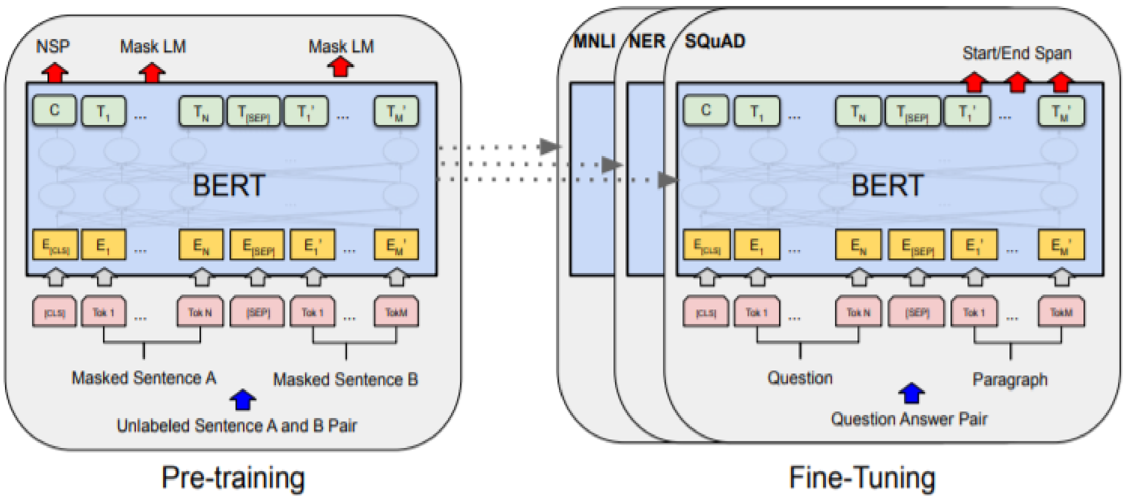

Figure 3. Pre-training and fine-tuning process for the BERT model

BERT model with 4 encoder layers, 8 attention heads, and an input vector size of 512 ($L = 4, H = 512, A = 8$) was chosen as the core for the classification model.

Building a database for training a classification model

The first step in creating a database for training a classification model was to define the classes that the model would use to determine the sentiment of text. Three classes were selected: “pro-Ukrainian”, “pro-Russian”, and “neutral”.

The next step was to determine the ratio of training, validation, and testing data, and their quantity in the database. The ratio of 16:1:1 was chosen. Accordingly, it was decided to create a database where each class contains 9600 unique messages in the training set, 1200 in the validation set, and 1200 in the testing set, and then apply data augmentation techniques to double the size of the training data set. Therefore, the database will contain a total of 67200 messages.

The messages were obtained from two sources: Telegram and Twitter. Obtaining messages from Telegram was done using the telethon library. A list of popular Telegram channels was compiled, and messages that fell into a specific class were collected. The number of selected channels per class is 16 for the training set, 3 for the validation set, and 3 for the test set.

Messages from Twitter were obtained using the snscrape library. Similar to reading messages from Telegram channels, a list of popular Twitter accounts was formed whose messages fall under a specific class. The number of selected accounts per class is 10 for the training set, 2 for the validation set, and 2 for the test set.

The algorithm for reading messages is follows:

1. Read the last 500 messages from the source (Telegram channel or Twitter account);

2. Process the message text (remove rare characters, replace emojis with their text counterparts, replace HTML symbols with their ASCII equivalents, mask links, and delete special character combinations responsible for text formatting);
3. Select messages that have a length between 50 and 5000 characters, thus removing messages that are too short or too long;
4. Translate the messages into English;
5. Save the successfully translated messages to a database, where each message corresponds to a record with the message content;

Subsequently, a programmatic reduction in the number of read messages in each category of each class was performed to achieve the desired amount.

After obtaining the desired number of messages, the final stage involved applying the following data augmentation techniques to increase the training set: reverse translation, sentence shuffling, and synonym replacement.

The process of reverse translation augmentation involves taking a source language sentence or phrase, translating it to a target language, and then translating it back to the source language. The resulting sentence may not be identical to the original source sentence, but it can still be a valid training example.

Example of new sentence generation using reverse translation: “I have no time” (English) → “je n’ai pas le temps” (French) → “I do not have time” (English).

The technique of synonym replacement has a similar goal to that of double translation, but in this technique, a certain percentage of words in the text are replaced with their synonyms, according to a synonym dictionary.

Example of new sentence generation using synonym replacement: “This article will focus on summarizing data augmentation techniques in NLP” → “This write-up will focus on summarizing data augmentation methods in NLP”.

The sentence shuffling technique involves randomly shuffling sentences (or a certain percentage of them) in the text. This technique helps prevent the model from paying excessive attention to any particular order of sentence.

Example of new text generation using sentence shuffling: “My name is Helga. I like to eat ice-cream” → “I like to eat ice-cream. My name is Helga”.

System implementation

The developed system represents an application consisting of a server-side and a client-side component.

The goal of the client-side component is to provide a fully functional graphical interface through which the user interacts with the system. In the developed system, the graphical interface is a web page that contains a number of panels, each of which serves either purely to display certain data or to interact with the user by obtaining certain data from the user through input form, processing the received data (either on the client-side or by directing the request to the server-side), and displaying the result of processing the request.

The main page of the client-side component of the developed system contains the following panels:

- A panel for entering a username of a Telegram channel or Twitter account, and displaying the averaged classification result of messages in this source;
- A panel for displaying and configuring the network of the most popular hashtags in Twitter related to the topic of the war in Ukraine, together with information about the political polarity of each of them and their pairwise occurrence in messages;
- A panel for displaying and configuring the network of interactions of popular Telegram channels, along with information about the political polarity of each of them;
- A panel for displaying statistics on changes in the popularity and political polarity of hashtags and accounts in Twitter over time;
- A panel for displaying statistics on changes in the number of subscribers and political polarity of Telegram channels over time;

The client-side component is implemented using the Angular framework, with the TypeScript language.

The server-side component is responsible for receiving requests from the client, processing them, and returning the result to the client. Structurally, the server-side component of the developed system can be divided into a request processing level, a business logic level, a level of working with the classification model, a level of reading and processing messages, and a level of working with a database.

The request processing level consists of a set of controllers that receive requests, check the source of the request and the correctness of the data that is coming in, and pass the request to the business logic level.

The business logic level processes the received requests, if necessary, by accessing other levels. This level implements the following functionality:

- Classifation of Telegram channel or Twitter account by username;
- Building a subnetwork of the most popular hashtags on Twitter based on the provided configuration and the saved version of the network;

- Building a subnetwork of the popular Telegram channels based on the provided configuration and the saved version of the network;
- Weekly creation of an updated version of the database and retraining the neural network on fresh data;
- Weekly construction of network for Twitter and Telegram, based on which subnetworks are built;
- Daily collection of statistics on changes in the popularity and political polarity of hashtags and accounts on Twitter;
- Daily collection of statistics on changes in the number of subscribers and political polarity of Telegram channels.

The algorithm for determining the political polarity of the message source using the given link can be described as follows:

1. Determine the type of the news source, whether it is a Twitter account or a Telegram messenger channel;
2. Read the latest m $(m \geq 20)$ messages from the source;
3. Perform preliminary processing of the read messages using selected methods;
4. Choose the processed messages that have a length between 50 and 5000 characters, thereby removing too short or too long messages;
5. Translate the messages into English;
6. For each translated message, extract its features using the selected approach;
7. For each translated message, determine its political polarity using a pre-trained classifier and the extracted features. The classifier is considered as a function that maps elements from the set of messages to the set of classification results. In this algorithm, the classification result is a triple $x = (u, r, n)$, where u, r, and $n(u, r, n \in R; u, r, n \in [0; 1])$ reflect the model's confidence level in assigning the message to the pro-Ukrainian, pro-Russian, and neutral classes, respectively;
8. Average the political polarity of the messages and return the averaged value x^{avg} as the result of determining the political polarity of the news source and end the algorithm.

The algorithm for constructing the graph G representing the network of the most popular hashtags is as follows:

1. Read the last n $(n \geq 100)$ messages from a given set of Twitter accounts;
2. Perform preprocessing on the read messages using selected methods;
3. Select the processed messages that have a length of 50 to 5000 characters and contain a non-empty set of hashtags;

4. Group the processed messages by hashtags, thus forming a set $B = \{b_1, b_2, \dots, b_m\}$, where $b_i = (h_i, L_i)$, h_i is the hashtag, L_i is the set of messages containing the hashtag h_i, and each hashtag appears in a number of messages greater than the parameter l_{min};

5. Build the set of vertices C of the graph G as follows:

5.1. For each hashtag h_i in set B, take p percent of the messages from the set of messages L_i in which this hashtag appears, but not less than the value of parameter m_{min} ($m_{min} \geq 20$) and not more than the value of parameter m_{max}

($m_{max} > m_{min}$). Denote this subset of messages as $L_i^{'}$;

5.2. Form the set $B^{'} = \{b_1^{'}, b_2^{'}, \dots, b_k^{'}\}$, where $b_j^{'} = (h_j, L_j^{'})$;

5.3. Translate all messages $L_i^{'}$ that are in the set $B^{'}$;

5.4. For each translated message, determine its political polarity $x_{i,j}$ using a pre-trained classifier and extracted features from the message;

5.5. Determine the average political polarity of messages x_i^{avg};

5.6. Form the set of vertices $C = \{c_1, c_2, \dots, c_k\}$, where $c_i = (h_i, t_i, x_i^{avg})$, $t_i = |L_i|$.

6. Based on the sets B and C, construct the set of edges D of the graph G as follows:

6.1. Form from the set C the set of all possible pairs of graph vertices $E = C \times C$;

6.2. For each vertex pair $(c_i, c_j) \in E$, determine the set of messages $L_{i,j}$ in which thehashtags h_i and h_j, representing the given vertices, appear pairwise;

6.3. Form the subset $E^{'}$ from the set E where the value $\frac{|L_{i,j}|}{\min(|L_i|, |L_j|)} \geq r_{min}$, where$r_{min}$is a given parameter ($r_{min} \in R, r_{min} \in [0; 1]$);

6.4. For each pair (c_i, c_j) in the subset $E^{'}$, add an edge to the graph G connecting vertices c_i and c_j with weight $w_{ij} = \frac{|L_{i,j}|}{\max(|L_i|, |L_j|)}$.

7. Return the graph $G = \langle C, D \rangle$ and complete the algorithm.

The algorithm for constructing a graph G representing a network of popular Telegram channels is as follows:

1. Read the last n ($n \geq 100$) messages from the specified set $A_0 = \{a_{0,1}, a_{0,2}, \dots, a_{0,m}\}$telegram channels. Define the set of processed channels $A_{proc} = \{\}$;

2. From the set of read messages, for each channel $a_{0,i}$, select a subset of messages containing links to other channels;

3. Perform preliminary processing of messages from a subset according to the selected methods;

4. Select among the processed messages those with a length of 50 to 5000 characters and form the set $L_{0,i}$ from them;

5. Form a set of links to channels $B_{0,i}$, which are found in messages from the set $L_{0,i}$ that have the number of subscribers greater than the parameter s_{min} and which do not belong to the set A_0 and the set of previously processed channels;

6. After searching for links to other channels from channels in the set A_0, add channels from the set A_0 to the set A_{proc} and check whether the number of processed channels $|A_{proc}|$ equal to the desired a_{dest} ($a_{dest} \geq |A_0|$). If so, go to step 7, otherwise, form the set of channels $A_1 = \bigcup B_{0,j}$, and alternately perform steps 2 – 5 for the channels from the set A_1, expanding the set A_{proc}after processing the next channel, until:

6.1. Number of processed channels $|A_{proc}|$ will be equal to the desired a_{dest}– then go to step 7;

6.2. All channels from the set A_1 are processed, but $|A_{proc}| < a_{dest}$ and $B_1 \neq \emptyset$ – then perform step 6, replacing the set A_0 with A_1 (or the set A_{i-1} with the set in A_i in the general case);

6.3. All channels from the set A_1 are processed, but $|A_{proc}| < a_{dest}$ and $B_1 = \emptyset$ – then go to step 7.

7. For each channel $a_i \in A_{proc}$, read the last k messages ($k \geq 20$);

8. Perform preliminary processing of messages from the set of read messages using the selected methods;

9. Select among the processed messages those with a length of 50 to 5000 characters and form the set R_i from them;

10. Form the set $C = \{(a_i, R_i) | a_i \in A_{proc}\}$;

11. Based on the set C, construct the set of vertices D of the graph G as follows:

11.1. Translate all messages of the set R_i that are in the set C;

11.2. For each translated message, determine its political polarity $x_{i,j}$ using a previously trained classifier and features extracted from the message;

11.3. Determine the average political polarity of messages x_i^{avg};

11.4. Form a set of vertices $D = \{d_1, d_2, \dots, d_k\}$, where $d_i = (a_i, L_i, s_i, x_i^{avg})$, s_i – number of subscribers to channel a_i, L_i – set of messages from channel a_i with existing links to other channels , which was found in step 5.

12. Based on the set D, construct the set of edges E of the graph G as follows:

12.1. Form from the set D the set of all possible pairs of vertices of the graph $H = D \times D$;

12.2. For each pair of vertices $(d_i, d_j) \in H$, determine the set of messages $L_{i,j}$ in which there were references to the channels a_i and a_j representing the given vertices;

12.3. Form a subset H' from the set H, where $|L_{i,j}| \geq r_{min}$, where r_{min} is a given parameter ($r_{min} \geq 1$);

12.4. Translate all messages $L_{i,j}$;

12.5. For each translated message, determine its political polarity $y_{i,j,k}$ using a previously trained classifier and features extracted from the message;

12.6. Determine the average political polarity of messages $y_{i,j}^{avg}$;

12.7. Form the set of edges $E = \bigcup\{e_{i,j}\}$, where $e_{i,j} = (a_{i,j}, t_{i,j}, y_{i,j}^{avg})$, $a_{i,j} = (a_i, a_j)$,$t_{i,j} = |L_{i,j}|$.

13. Return the graph $G = \langle D, E\rangle$ and complete the algorithm.

Let the given network of the most popular hashtags be represented by the graph $G = \langle C, D\rangle$and the parameters $t_{min}, t_{max}, s_{min}, s_{max}, x_{min}, x_{max}, y_{min}, y_{max}$, where $C = \{c_1, c_2, \dots, c_k\}, c_i = (h_i, t_i, x_i^{avg}), D = \bigcup\{d_{i,j}\}, d_{i,j} = (h_{i,j}, s_{i,j}, y_{i,j}^{avg})$. Then, the algorithm for constructing the subgraph G', which represents the subnet of the most popular hashtags, looks as follows:

1. Form the set $C' = \left\{ c_j \middle| \begin{array}{c} c_j \in C,\ c_j = (h_i, t_i, x_i^{avg}), \\ t_{min} \leq t_i \leq t_{max}, \\ x_{min} \leq x_i^{avg} \leq x_{max} \end{array} \right\}$;

2. Form the set $D' = \left\{ d_{i,j} \middle| \begin{array}{c} d_{i,j} \in D,\ d_{i,j} = (h_{i,j}, s_{i,j}, y_{i,j}^{avg}), \\ s_{min} \leq s_{i,j} \leq s_{max}, \\ y_{min} \leq y_{i,j}^{avg} \leq y_{max} \end{array} \right\}$;

3. Return the graph $G' = \langle C', D'\rangle$ and complete the algorithm.

Let the given network of the most popular Telegram channels, represented by the graph $G = \langle D, E\rangle$and the parameters $l_{min}, l_{max}, t_{min}, t_{max}, s_{min}, s_{max}, x_{min}, x_{max}, y_{min}$,

y_{max},

where$D = \{d_1, d_2, \dots, d_k\}, d_i = (a_i, L_i, s_i, x_i^{avg}), E = \bigcup\{e_{i,j}\}, e_{i,j} = (a_{i,j}, t_{i,j}, y_{i,j}^{avg})$are set. Then, the algorithm for constructing the subgraph G', which represents the subnetwork of the most popular Telegram channels, looks as follows:

1. Form the set $D' = \left\{ d_j \middle| \begin{array}{c} d_j \in D,\ d_i = (a_i, L_i, s_i, x_i^{avg}), \\ l_{min} \leq |L_i| \leq l_{max}, \\ s_{min} \leq s_i \leq s_{max}, \\ x_{min} \leq x_i^{avg} \leq x_{max} \end{array} \right\}$;

2. Form the set $E^{'} = \left\{ e_{i,j} \middle| \begin{matrix} e_{i,j} \in E,\ e_{i,j} = \left(a_{i,j}, t_{i,j}, y_{i,j}^{avg}\right), \\ t_{min} \le t_{i,j} \le t_{max}, \\ y_{min} \le y_{i,j}^{avg} \le y_{max} \end{matrix} \right\}$;
3. Return the graph $G' = \langle D', E' \rangle$ and complete the algorithm.

The server part was implemented in the Python programming language. Tensorflow and keras libraries were used to work with neural networks. To read messages, telethon and snscrape libraries were used. To increase the data, nlpaug library was used. To translate messages, googletrans library was used. Networkx and matplotlib libraries were used to construct and display graphs.

The structure of implemented classification model is shown in Fig. 4.

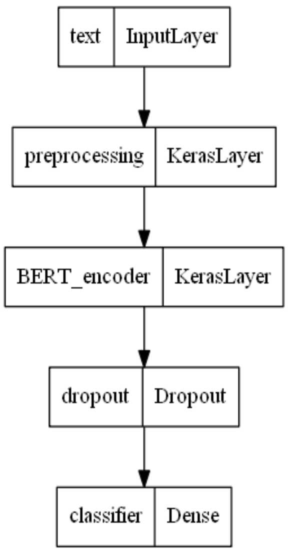

Figure 4. The structure of implemented classification model

Classification model training parameters are described in Table 1.

Table 1.

Classification model training parameters

Parameter	**Value**
Number of epochs	3
Initial learning rate	10^{-4}
Loss function	Categorical Crossentropy
Performance metrics	Categorical Accuracy
Optimization method	AdamW

In addition to the parts described above, the structure of the software solution also includes the PostgreSQL database, which is accessed by the server part.

System testing

PC characteristics on which the system was tested:

1. The clock frequency of the processor is 2.6 GHz;
2. 6 physical cores, 12 logical processors;
3. 16 GB of RAM;
4. Available NVIDIA GeForce GTX 1660 Ti video card.

The following technical requirements were set before developing the system:

1. The accuracy of the text classification should be not less than90% on the training sample, and not less than 80% on the test sample;
2. The minimum and maximum number of vertices of the graph of the most popular hashtags in the Twitter network and the graph of the interaction of popular Telegram channels are 10 and 100, respectively;
3. The maximum time for determining the political polarity of the message source is 10 seconds.

The results of learning the developed model are as follows:

1. Accuracy of text classification on the training set: 95%;
2. Accuracy of text classification on the test set: 83%;
3. The average study time during the epoch is 9 minutes.

The results of testing the system's average request processing time:

1. Determining the political polarity of the message source - 7 s;
2. Construction and display of subgraphs based on the transferred configuration and saved version of the graph - 500 ms.

The results satisfy the technical requirements that was set before developing the system.

Conclusions

A model for the classification of the political polarity of the text has been developed, and a database for its training has been created. A software application has been implemented that uses the functionality of a trained model to generate various statistics related to pro-Russian propaganda. The implemented application meets the requirements for speed and classification quality. Further improvement of the quality of model classification can be done with the help of using more complex architecture or a combination of several architectures that focus on different subtasks, improving the quality of training data using more complex message processing methods, etc.

REFERENCES

1. Суспільно-політичні настрої населення України: результати опитування, проведеного 9-17 грудня 2021 року методом особистих (“face-to-face”) інтерв’ю. URL: https://www.kiis.com.ua/?lang=ukr&cat=reports&id=1080&page=1
2. Драбюк С.С. Пропаганда та її види. Шляхи протидії пропаганді. // Аналітично-порівняльне правознавство. – 2022. – №1. – с. 153-157. https://doi.org/10.24144/2788-6018.2022.01.28
3. Shymkovych V., Telenyk S., Kravets P. Hardware implementation of radial-basis neural networks with Gaussian activation functions on FPGA. // Neural Computing and Applications. – 2021. – 33(15): 9467-9479. https://doi.org/10.1007/s00521-021-05706-3
4. Dreyfus G. Neural Networks: Methodology and Applications. // Springer-Verlag, Berlin. – 2005. – 498 p. https://doi.org/10.1007/3-540-28847-3
5. Kravets P., Nevolko P., Shymkovych V., Shymkovych L. Synthesis of High-Speed Neuro-Fuzzy-Controllers Based on FPGA. // 2020 IEEE 2nd International Conference on Advanced Trends in Information Theory (ATIT). – 2020. – 291-295. https://doi.org/10.1109/ATIT50783.2020.9349299
6. Bezliudnyi Y., Shymkovysh V., Doroshenko A. Convolutional neural network model and software for classification of typical pests. // Prombles in programming. – 2021. – 4: 95-102. https://doi.org/10.15407/pp2021.04.095
7. Bouvier M., Valentian A., Mesquida T., Rummens F., Reyboz M., Vianello E., Beigne E. Spiking neural networks hardware implementations and challenges: a survey. // ACM Journal on Emerging Technologies in Computing Systems. – 2019. – 15:22. https://doi.org/10.1145/3041033
8. Gonçalves S., Cortez P., Moro S. A deep learning classifier for sentence classification in biomedical and computer science abstracts. // Neural Computing and Applications. – 2020. – 32: 6793–6807. https://doi.org/10.1007/s00521-019-04334-2
9. Rao A., Spasojevic N. Actionable and Political Text Classification using Word Embeddings and LSTM.// arXiv:1607.02501 – 2016. – 9 p. https://doi.org/10.48550/arXiv.1607.02501
10. Geovany I., Arturo J. A sentiment analysis of the Ukraine-Russia conflict tweets using Recurrent Neural Networks.– 2022. – 5 p.
11. VADER-Sentiment-Analysis. URL: https://github.com/cjhutto/vaderSentiment.
12. Vaswani A., Shazeer N., Parmar N., Uszkoreit J. Attention Is All You Need. // arXiv:1706.03762 – 2021. – 15 p. https://doi.org/10.48550/arXiv.1706.03762
13. Devlin J., Chang M., Lee K., Toutanova K. BERT: Pre-training of Deep Bidirectional Transformers for Language Understanding. // arXiv:1810.04805– 2018. – 16 p. https://doi.org/10.48550/arXiv.1810.04805