Fake News Detection using Natural Language Processing

Mohamed Ayman†
Computer Engineering
The American University in Cairo
mohamedayman15069@aucegypt.edu

Abdelrahman Shaaban
Computer Engineering
The American University in Cairo
abdelrahmanfawzy@aucegypt.edu

1. Model Choice

Based on the pilot study and after experimenting two different embedding approaches with multiple classifiers and observing the results in the third phase, we found that a deep neural network model with an earlier wor2vec layer outperforms all the other models. Our initial design for this model is shown below.

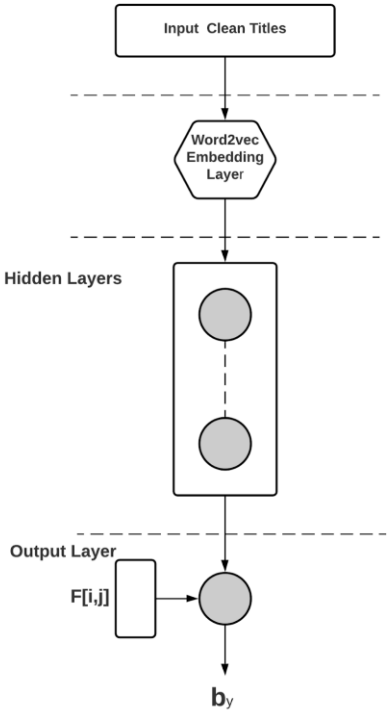

Fig.1: a brief diagram for the deep learning model.

WOODSTOCK'18, June, 2018, El Paso, Texas USA

https://doi.org/10.1145/1234567890

1.1 Word2Vec Embedding Layer

Word2vec embedding is the first layer of the model to be implemented and used to learn the training data's word embeddings. We will implement the CBOW model, that is shown in figure 2, for training the embedding layer as it has a better generalization and faster performance compared to the skip-gram model. In the CBOW model, the objective is to predict the current word from a window of context words. For training the CBOW model, we will use an embedding vector size of 500 with the default size of words' window =4. By training the model, we need to loop over the whole training and testing samples and apply the embedding on each word to get a training set of (400660, 500) and testing set of (133554, 500).

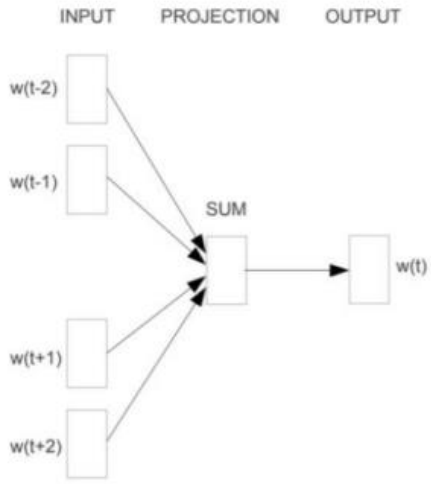

Fig.2: Word2Vector neural network.

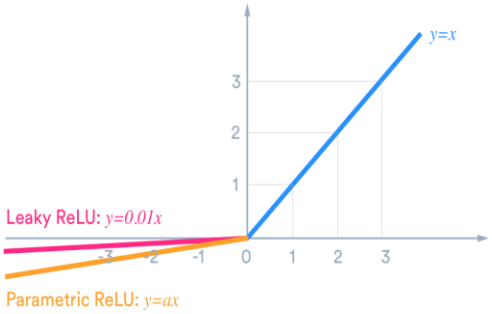

Fig.4: Leaky Relu graph with its graph

1.2 Deep Learning Model

```
In [89]: from sklearn.neural_network import MLPClassifier
         NN = MLPClassifier(random_state=1, max_iter=5000, activation='logit')
         NN.fit(w2f_df.values,y_train.values)
         test_NN_word2vec = NN.predict(w2f_test_df.values)
         print(classification_report(y_test.values,test_NN_word2vec))
```

	precision	recall	f1-score	support
0	0.74	0.75	0.74	70421
1	0.72	0.70	0.71	63133
accuracy			0.73	133554
macro avg	0.73	0.73	0.73	133554
weighted avg	0.73	0.73	0.73	133554

Fig.3: MLP classifier

As shown in figure 3, the pilot study proves that the MLP Classifier outperforms all the other models. This basic model that consists of the default number of hidden layers equals to (100,), and uses a logistic activation function, a learning rate of 0.0001 and "Adam" optimizer. We will start by implementing this model and working on its finetuning, experimenting different numbers of hidden layers, different activation functions as explained below, different optimizers, and different learning rates. When it comes to the reason why we choose this model, a neural network is the fittest for such a kind of classification problems with its high complexity that is essential for getting the many text features' patterns. To clarify, it is basically a deep learning model that contains an input layer to get the text embeddings, multiple hidden layers to detect its pattern, whose nodes will use a leaky relu activation function, and a tanh output layer for the binary classification. The reason for choosing leaky relu to experiment with in the implementation phase is that the embedding vectors have negative numbers, and we understand that normal relu discards any negative number to zero which will affect negatively on our data. In this sense, it is important to use an activation function not discarding any negative numbers, such as leaky relu. About the output layer activation function, we completely understand that our embedding data has negative numbers. Most probably (we need to experiment it), the hidden layers would be provided with leaky relu, shown in figure 4, as an activation function. In this sense, tanh, as shown in figure 5, would fit as an output layer with taking into consideration that from 1 to 0 is the label with true and 0 to -1 as false. The training and the validation processes will

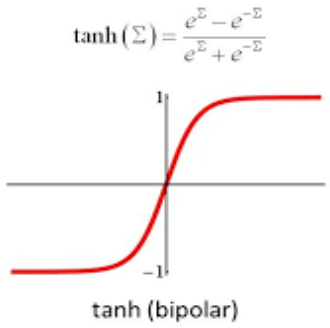

Fig.5: Tanh equation with its graph

include optimizing the layers' weights throughout forward and backward propagation, searching over the best hyperparameters such as (Number of hidden layers, optimizer, learning rate). In more detail, we understand that when learning rate is large, it is not guaranteed to converge but very fast, while the small learning rate leads to a guaranteed convergence but takes much time. We have found adaptive learning rate, like ADAM optimizer trying to adapt the learning rate somehow. Regarding hidden layers, we will try to implement first with the default number of the hidden layers, but we need to experiment with other ones in our implementation and choose the best one. To be able to train, we need to calculate the error and then distribute the error using loss function. and finally evaluating the model performance.

$$H_p(q) = -\frac{1}{N}\sum_{i=1}^{N} y_i \cdot log(p(y_i)) + (1 - y_i) \cdot log(1 - p(y_i))$$

Binary Cross-Entropy / Log Loss

Fig.6: Binary Cross-Entropy equation

Our problem is a binary classification problem that is often framed as predicting a value of 0 or 1 for whether the title is true or fake and is often implemented as predicting the probability of the example belonging to true news. We found that the Binary Cross-Entropy/ Log loss function is the most suitable to be used. Mathematically, it is the preferred loss function under the inference framework of maximum likelihood. It mainly creates a criterion that measures the Binary Cross Entropy between the target and the output using the formula shown in figure 6.

The pros of using such a model are that it has the complexity needed for such kinds of NLP problems with thousands of problems. From the pilot study, we found that it is the most efficient at delivering high-quality results (accuracy, precision, recall) compared to the other machine learning models. On the other hand, one of the most common problems of the neural networks is overfitting. It mainly happens when an algorithm learns the detail in the training data to the extent that negatively impacts the

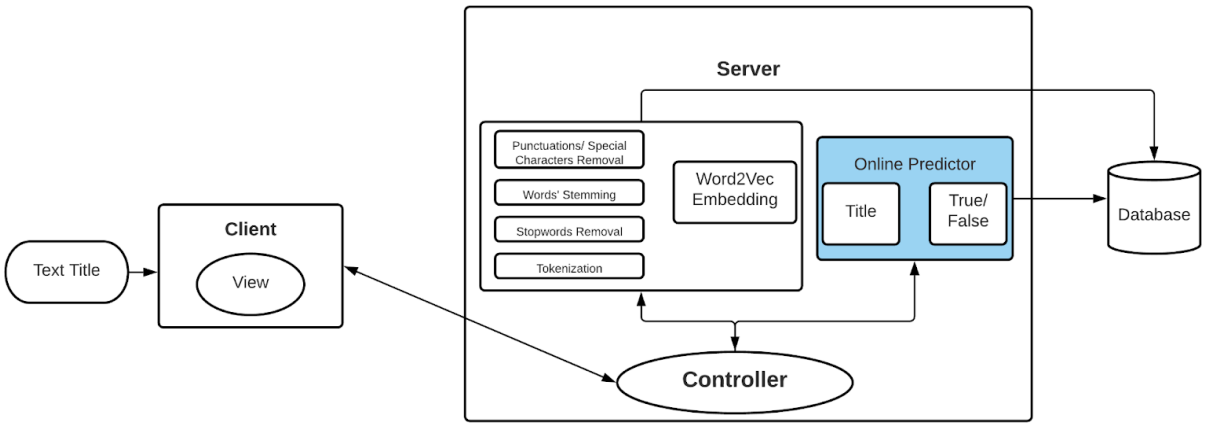

Fig.7: MVC (Model View Controller) scheme to our project

performance of the model in real-life scenarios. Therefore, we need to handle such a problem that can cause a high drop in the model's evaluation using different techniques of regularization, dropout, and early stopping.

2. The Application Utility

In our project, we simply need to create a simple application that is applicable to take normal input consisting of a string of words from any user with good visual and accessibility. Then, it responds to the user with a message whether the news is fake or not using some APIs. In this section, we mention a complete design of the application in terms of frontend and backend, how the client is communicating with the server, some diagrams showing exactly how the data flows, and a sample of test cases showing the input and output clearly for the user.

2.1 Frontend:

We will be using Django as our web interface for a couple of reasons. Django is a high-level Python web framework that leads to rapid development of secure websites. Django has some implemented packages for necessary functionalities in frontend to avoid any unnecessary problems we may encounter. Also, our intention is to write a highly accessible frontend without any problems in terms of visualization and connection with the backend. The hierarchy of the project is simply a server and several applications as user interfaces (The detail is in architecture of Django). Fortunately, Django can create the server itself. This server contains the database, as Django is suitable for backend and frontend, some sub-URLs called end points to be able to go from a specific URL to another one and so on.

2.2 Backend:

We will use Django in the backend as well since it is suitable for the frontend and backend with suitable APIs. This simply will take the input from the client using **POST** request. The preprocessing and predicting data will occur in the backend. Afterwards, the results will be returned to the frontend using **GET** request.

2.3 The Architecture of Django and our project:

In more detail, Django is a Model-View Controller, which is widely used in web project development. The main advantage of such an architecture is that it can separate the input layer (the client or user), model processing layer, and control flow from each other completely. This is quite good in terms of separation of concern. You can see the architecture of such a generic architecture in figure 8.

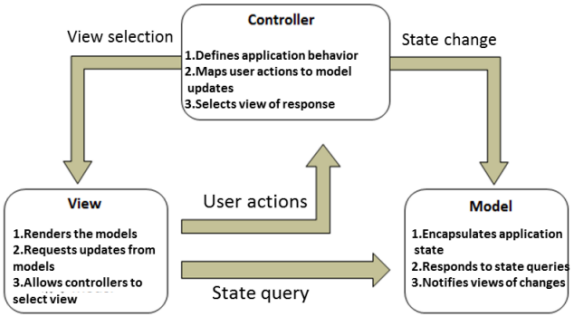

Fig.8: Model View Controller Scheme in Django

Related to our project in the backend part, we have three parts in our project: the client giving to us the input (a series of words), the two server models for preprocessing data, embedding the text using Word2Vec, and predicting the model, and the database needed for storing the pretrained model values, and saving the pretrained weights for prediction further. From figure 7, the client side is the view interface for the user which can write the string of words and request updates to the server. On the other hand, the server side- two models and controller- receives the input of the user in terms of a string of words to do some essential processes. We have a controller which is responsible for exchanging data between client and server

models, and between two server models. Then, the two server models are responsible for preprocessing the text to appropriate numerical presentation to be valid for the model and predict the preprocessed string of words whether the news is fake or not. In other words, the first server model is accountable to preprocess- deleting the stopping words, non-English words, tokenizing and stemming the words, and embedding the words- the title in the First server model. This model will come out with numerical embedding vectors. This second server model is mainly responsible for predicting whether the tweet or title of the news is fake or not. It takes the numerical representation resulting from the first server model with the help of the controller and predicts the authenticity of the input in the second server model. With respect to the database, it can connect with the servers to get all necessary information. We need a dictionary for the pretrained value to be able to predict without training the model and the dictionary of our implementation and customized with our data for embedding. Then, the prediction will return to the client with the prediction of the string of words.

2.4 The Data Flow:

2.4.1 The Training Data Flow

The Fakeddit dataset needs to be preprocessed somehow to be able to train the weights of the Neural network. In the second phase, we preprocessed the dataset as follows:

1. We dropped the unnecessary columns with discernible justification as mentioned in other phases.
2. We removed all special characters and punctuations.
3. We needed to stem all words to their originality for the sake of decreasing the number of words.
4. We removed all stopping words
5. Finally, we made a threshold to remove all rare and frequent words.

From figure 9, we can see that we put the preprocessed dataset in terms of arrays in Word2Vec Embedding, coming out with a complete numerical representation for each unique word. Now, the words in terms of numerical representation are ready to be considered as input to the model to be trained. After training and validating, the pretrained weights are saved in the database for testing later.

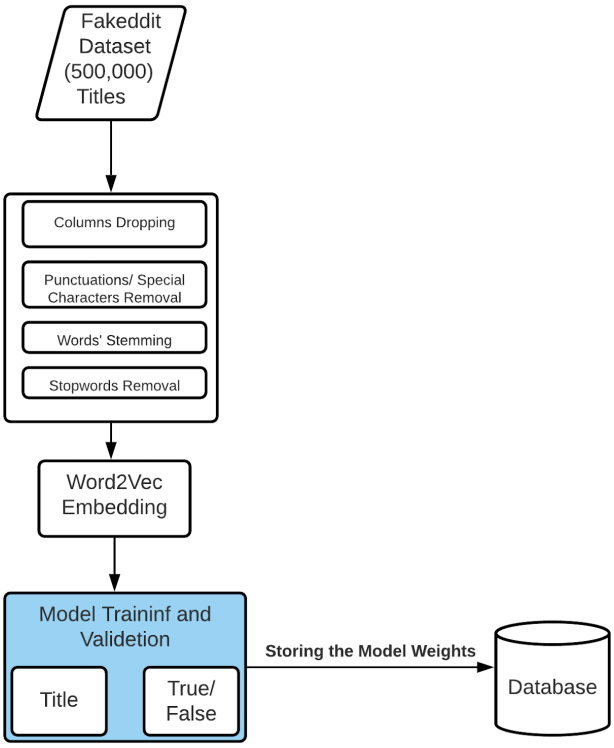

Fig.9: The training data flow.

2.4.2 The Testing Data Flow

The input of the user is a string of words. It needs to be preprocessed again to be valid to be predicted. The same steps will occur as in the training data flow. Then, the numerical representation of the tested title is ready to be predicted. The online predictor will take from the database the pretrained model to predict the title without much time.

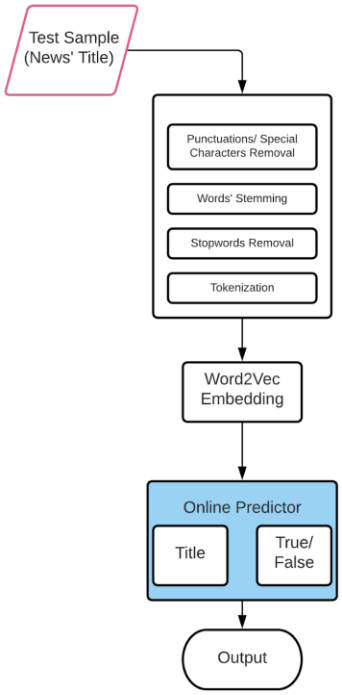

Fig.10: The testing data flow.