Asad Abbasi, Hussein Adams, Sudha Vijaykumar, Jacob Rodriguez

Real vs Fake News Classification

I. Introduction

Distinguishing between real and fake news has never been more important as it is today. Harmful actors in society have taken advantage of the widespread use of media outlets and social media platforms, such as Twitter and Facebook, to spread misinformation and fake news. The significance of this problem has highlighted the potential applications of using data science and machine learning algorithms to classify real and fake news in real-time. Such applications would be able to monitor social media platforms and filter out any misinformation before being conveyed to the general public.

This project solves this problem by developing a robust text classification system using data mining techniques that can be deployed to classify real and fake news in real-time. Additionally, we provide an analysis on the comparison between different data mining models as well as different natural language processing (NLP) techniques to highlight the advantages and disadvantages of each technique as it relates to this text classification problem.

In order to build a robust and generalized text classification system that can be used to distinguish between real and fake news we take a ground-up approach. We start with the training and evaluation of basic classification models, such as logistic regression and random forest. We quickly realize that there exists a clear decision boundary between the two classes in the dataset, which leads us to utilize the support vector machine (SVM) framework to take advantage of this feature. Finally, we train a deep learning NLP model on the dataset to evaluate the usefulness and effectiveness of deep learning methods in this field. Finally, we also evaluate the effect of semantic embedding models on the performance of our classification models. A thorough analysis is conducted to find the best performing semantic embedding model that fits this problem. These embeddings are used to re-train all of the previous models to determine the impact of using these embedding techniques in this text classification problem.

There has been a substantial amount of work in building text classification systems that can accurately distinguish between fake and real news, specifically in the field of deep learning and artificial intelligence. Ruchansky et al. [1] use a Recurrent Neural Network (RNN) in a hybrid model to capture user activity on a given article in order to learn the characteristics of the source of the article. Finally, they use this information to classify an article as fake or not. This model has shown to be effective in extracting meaningful information from both the text of the article and the users who have viewed the article. Lin et al. [2] go further by using a hierarchical RNN framework to distinguish between rumors and factual news on online social media platforms. They are able to achieve this by using the deep learning framework to learn the contextual information by conducting time series analysis of events on social media.

Despite this success with using deep learning models to detect fake news, in this experiment we train and implement alternative models such as logistic regression, random forests, and support vector machines along with neural networks in order to compare and evaluate the performance and accuracy tradeoffs between the two sets of models on this task.

II. System Design and Implementation

Statistical machine learning (ML) algorithms work with numerical data. In order to apply ML algorithms to the task of binary classification of news articles as fake or not fake, a numerical representation of the documents becomes necessary. There are multiple ways of representing documents as numbers, each with its own advantages and disadvantages. In this paper, we'll compare some of these algorithms. Specifically, we'll compare the embeddings and non-embeddings methods of representing text in the context of classification. Each method was evaluated using logistic regression, random forests, SVM, and neural networks. Logistic regression is a statistical model that is often used in classification problems and is grounded in probability theory. Random forests, on the other hand, is an ensemble-based learning algorithm that combines multiple decision trees to increase performance. Lastly, SVM is a well-proven classification algorithm that classifies data points by finding a hyperplane that separates them.

Non-embedding is a method of representing text as numbers where each number reflects a score of how important it's respective word is to a given document. In other words, it's a scoring scheme for words in a document. For example, the following document "I like learning and I like sports and cars." might have the numerical representation seen in **Table 1**. This method of document representation does not have contextual understanding of the words. It only knows of words and their counts. The two methods compared in this work are TF-IDF and Count Vectors. We ended up selecting the TF-IDF model because it de-emphasizes words that are not as important to a document. This is different from Count Vectors that is essentially a count of all words.

Words	I	like	learning	and	sports	cars.
Count	2	2	1	2	1	1

Table 1

The other method of representing text as numerical data is embeddings. Word embeddings is a hot area of study in the NLP world. It deals with the semantic representation of text as numerical data. This means converting text to numbers that understand language. This is represented by a vector of numbers of a certain length (i.e. 200, 300, 512..etc) where each number in the vector represents an attribute of that word. For example, the word "Apple" might have the vector representation of size five as seen in **Table 2**. Each attribute value represents how close the word "Apple" is to that particular attribute. The result is vectors that represent the

words which can be used to perform similarity measures of how close two words are. This is different from the non-embeddings methods such as TF-IDF or Bag of Words models which only measure how important words are to a specific document.

Word/Attribute	Food	Airplane	Fruit	House	Pie
Apple	0.9	0.01	0.99	0.1	0.85

Table 2

There are two approaches to take when working with word embeddings. First, employing pre-trained models. This method involves downloading a model that was previously trained on a large corpus of text (wikipedia, reddit, twitter..etc) Then, loading the model to infer news articles vectors. The second approach is to train a model from scratch on the news articles. Both approaches were taken and compared. In the first approach, we compared fourteen different pre-trained models. We'll mention eight of the best performing ones. Namely, the models are, BERT, USE, Elmo, GloVe, Word2vec, Albert, BART and T5. In the second approach, the doc2vec model was trained from scratch on the news articles alone.

The first embeddings approach is to use pre-models models. Because these models are extremely large (2-5 GB) with a number of attributes ranging from 300 to 1024 (embedding vector size) inference can take a very long time. This is especially true if a GPU is not present. We found that BERT could take two hours to infer 10k documents. Elmo took four hours to infer the same number of documents. Therefore, due to time constraints, we selected a sample of 10k documents to perform evaluation of the pre-trained embeddings models. The same documents were used in all of the experiments. Then, logistic regression was used to evaluate classification accuracy of each model's embeddings. The procedure we followed to evaluate each model is outlined below. Finally, **Table 3** shows which models were used and their accuracies.

1. Download and load the model.
2. Infer 10k documents
3. Perform test/train split with test size of 25%
4. Use Logistic Regression with default parameters to classify the documents.

	BERT	**USE**	**Elmo**	**GloVe**	**Word2vec**	**Albert**	**BART**	**T5**
Train Accuracy	98.4%	96.3%	99%	96.7%	95%	98%	99.9%	99.6%
Test Accuracy	97.8%	95.5%	98%	96.9%	94%	97%	99.4%	99.0%

Model	bert_base_uncased	use_dan	elmo_bi_lm	wiki_300	google_news_300	albert_base	Bart-large	t5-large

Table 3

The second embeddings approach is to train a model from scratch. We used a Doc2Vec model to train on n documents. We split the dataset into three different sets (train, test and validation) The train and test set was used to train the Doc2Vec model to learn the documents representation. The test set was used to infer the documents that were used to train the model. Logistic regression was used on the test set to classify the documents. This resulted in a classification accuracy of 98 percent. However, the test set was seen by the doc2vec model during training. The real evaluation is to infer and classify documents that were never seen by the doc2vec model. Hence, using Logistic Regression on the validation set yielded accuracies between 73-93 percent. This result is far worse than the pre-trained models.

Finally, through experimentation, we found that the pre-trained models documents representations performed better than training a model from scratch. Specifically, using the data in **Table 3** we found that BART word embeddings performed the best in classifying the news articles as fake/not fake. Therefore, for the rest of this study, we'll use BART word embeddings to compare the classification performance of embeddings against the classification performance of non-embeddings representation using logistic regression, random forests, support vector machines and dense-layered neural networks.

Various libraries and packages were used to preprocess our data and train our classification models. The natural language toolkit (NLTK) was used for preprocessing of our text. We used the NLTK stopwords corpus and the punctuation corpus to clean the text and remove all non-alphanumeric characters. The logistic regression, random forest, and SVM models were trained using the Scikit-Learn (Sklearn) machine learning library. Sklearn also supports hyperparameter tuning and different evaluation methods for each model. Finally, the deep neural network models were trained using the TensorFlow and Keras libraries. All experimentation was done in the Google Colab environment. Lastly, we used the GPU hardware accelerator to increase the efficiency of our neural network training process.

For the architecture of the neural network itself, we wanted to keep the amount of hidden layers to a minimum so as to not overfit but also make sure there was enough "memory" so as to accurately capture the relevant patterns. Although two separate models were constructed, one for embedding data and another for non-embedding data, their architecture mirrored one another in almost every way. The only key difference was the regularization layers that were added to address overfitting. After experimentation with different layer combinations, the final model below is what performed best and was subsequently used for the neural network testing and validation.

Link to model architecture

The potential applications of a text classification system that can accurately distinguish between fake and real news are innumerable. The deployment of such a system on online social media platforms can filter fake and malicious information in real-time. In addition, this text classification model can be further fine tuned towards different media platforms in order to further decrease the number of false positives or false negatives. Implementation of this model will ensure that only factual information is being conveyed to the public.

III. Experiments/Proof of Concept

We initially started out by training all of our models on the Fake and real news dataset provided by Kaggle [1]. This dataset consisted of 17,903 fake news articles and 20,826 real news articles. Each data point in this dataset had four features, which included the title of the article, the contents of the article, the article's subject, and the date of publication. While this dataset had a substantial number of data points we found that all of our models were overfitting to this dataset very quickly for multiple reasons. During initial exploration of this dataset we found that the topics in the subject column were exclusive. This meant that the model could very easily classify a piece of text as fake or real news just by looking at the subject column. To fix this issue, we completely removed the subject feature from this dataset.

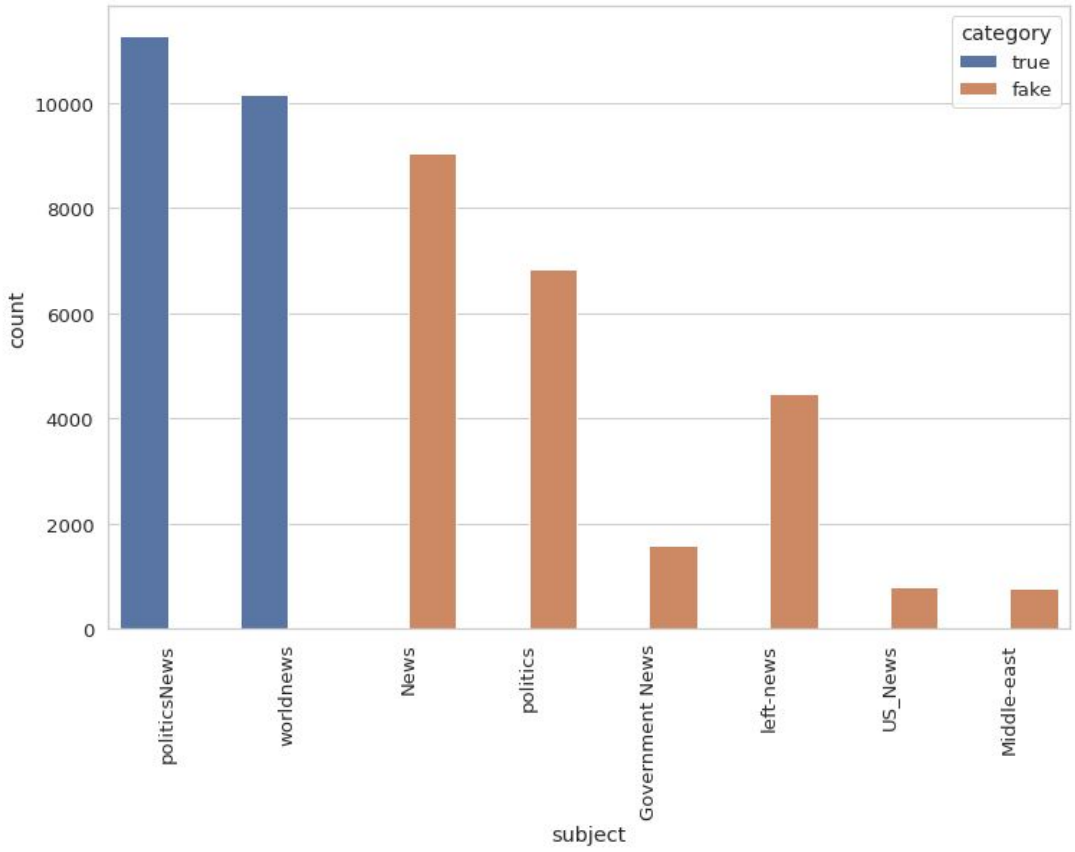

Figure 1

Additionally, through word cloud analysis we noticed that the two categories of texts significantly differed in keywords and subject matter which also made it easier for models to classify between the two classes. However, this also led to significant overfitting as the classification models were not focusing on the semantic relationships between the words as much as they were on the specific keywords. To resolve this issue and make our text classification system more robust and generalizable, we supplemented our current dataset with several thousand data points from the Fake News corpus. The Fake News corpus is an open source dataset that contains articles from millions of sources online. We mined an additionally 22,500 documents from each category and added those to our current dataset. This new set of documents was much more varied which made the models less prone to overfitting.

Before preprocessing our dataset, we conducted exploratory data analysis (EDA) to gain insights into the dataset and use these insights to implement our preprocessing and model training methods. Firstly, we found that the distribution of the number of true documents and fake documents was roughly equal. This was beneficial because it meant that our models would not be skewed by an imbalanced dataset. Additionally, the distribution of the length of the texts in each category were roughly similar. This meant that we would not have to do additional preprocessing to ensure that all texts were of the same length. If documents of one class were longer than documents of the other, the model could overfit by learning the average length of the texts rather than the contents of the texts. Lastly, through word frequency analysis we found that particular words were unique to a specific class. For example, the words 'Reuters' and 'Washington' were found in 21,396 documents out of the 21,417 real news documents in the original dataset, while those same words were only found in 3560 documents out of the 21,417 fake documents in the same dataset. This was significant because the presence of these words in one category and the lack of their presence in the other would cause the classification models to overfit to this noise and associate any article that has these words with the real news category. To avoid this issue, we decided to remove these words from our dataset. N-gram analysis was also conducted to further explore the make-up of each set of documents.

Extensive data preprocessing was conducted to ensure that our models could learn the semantic meanings and relationships of the text without much variation and noise. The format of the data was raw text, so additional cleaning of the text was also required. Our preprocessing approach is outlined below.

1. Making all texts lowercase.
2. Removing all non-alphanumeric characters.
3. Numbers are often represented as # in vectors, so we converted all numbers into #.
4. Removing stop words and other words that can skew the model towards a particular category.
5. Turning contradictions into full form, i.e. can't => can not.
6. Changing target values from "Real" and "Fake" to 1 and 0.

This preprocessed text was combined in a dataframe and was ready to be passed on for training of the models. Eight different models were trained in this experiment. We processed this data differently for models trained without embeddings, models trained with embeddings, and deep neural network models. For models that were trained without word embeddings, our data was vectorized using the term frequency-inverse document frequency (TF-IDF) method. This vectorized data was then split into a training set and a test set. Due to the large number of documents in our dataset, we proportioned 80% of our dataset to our training set and the rest of the 20% of the dataset to our test set. For the models trained with embeddings, we used pre-trained BART embeddings as inputs to our model. Lastly, for the neural network models, we made sure to pad our data so that all the documents were of the same length.

In order to evaluate the effectiveness of word embeddings in the performance of our models we ran the logistic regression, random forest, and SVM models with both embeddings and non-embeddings. In order to further optimize the performance of our models, we performed hyperparameter tuning through RandomizedSearchCV from sklearn to find an optimal combination of hyperparameters and further increase accuracy.

The neural network creation process was split into two separate attempts: one for BART embeddings and one for non-embeddings. The embeddings model consisted of six dense hidden layers, each with 64 output neural connections. The larger dataset made it harder to overfit the data during training but that still was an issue that had to be addressed since there was a discernible difference between training and test accuracy. After experimentation with a combination of both batch normalization and dropout layers, the final model only added batch normalization to the existing dense layer structure. This allowed for near perfect accuracy during training and a tight testing accuracy to follow. The non-embeddings model presented a different issue due to the large feature count that was in the data itself. The system environment kept on crashing during training rounds since large amounts of RAM had to be allocated each time a document was passed. This limiting factor influenced the final model design, since emphasis had to be put on keeping the parameter space small. Still, model performance was able to get into the 90th percentile. Again, this model had 6 dense hidden layers, but after experimentation, it was found that dropout layers rather than batch normalization helped in getting testing accuracy close to training accuracy. Overall accuracy was lower than the embedding model but if not for the constraint of having such a large feature space, more layers could have proven to be a useful strategy for getting a higher accuracy. Overall, the model design did not need to be too complex to capture good performance for both embedding and non-embedding data. It can be seen that neural networks perform very well when it comes to text classification and should be considered a viable option when attempting to solve these types of problems.

We found that for this problem, embeddings did not make a significant difference to the performance of the model with the exception of the neural network model, but this may be due to the aforementioned model constraints. This might be because the types of words in the two classes are very different. It could also mean that the presence or absence of certain words has

more impact than their meaning or positioning. However, the fact that Neural Networks performed better on embeddings than the statistical methods could mean further parameter tuning is needed or that the statistical methods do not perform good enough with extremely high dimensions and a large number of data points.

A summarization of the results of our experiment is included in the table below. Overall, the neural network model trained with embeddings had the highest accuracy and performance out of all the models trained. Furthermore, in the comparison between embeddings and on-embeddings, we found that logistic regression, random forests, and the SVM model performed significantly better when trained without embeddings as compared to when they were trained with embeddings. Due to the low complexity of the problem and exclusivity of content between the documents of the two classes, it is likely that the models trained with embeddings overfit to the training data, which resulted in the non-embedding models having a higher accuracy. Finally, in the comparison between the performance of logistic regression, random forests, and SVM, we found that the logistic regression and SVM models performed significantly better than the random forest models. This is likely because the nature of the dataset used in this experiment presents relatively clean and linearly separable classes. The separability of the classes on this dataset yield better results with logistic regression and SVM, as their aim is to find a separation between the classes. Random forests, on the other hand, are simpler models due to their tree-based foundation. In addition, while they are known to perform extremely well on categorical data, they do not perform as well on continuous numerical data. Additional feature engineering could help in solving some of these issues. Overall, both logistic regression and SVM had the highest average accuracies across all eight models trained with both embeddings and non-embeddings.

	Logistic Regression	Random Forests	SVM	Neural Networks
Non-Embeddings	98.5%	96.8%	98.1%	93.2%
Embeddings	94.8%	93.3%	95.4%	98.8%

Table 4

Link to Visualization Dashboard

IV. Discussion and Conclusion

While all the classification models in this experiment had a validation accuracy greater than 90%, some performed significantly better than the others. However, selection of which model to deploy might depend on the user's requirements. For example, the non-embeddings model (TF-IDF vectors) would not be a good choice to deploy. This is because, to classify new unseen documents, we need to retrain on the entire corpus in order to account for new words. Additionally, we looked at the tradeoff between speed and accuracy in the data science process. Fake news classification is a dynamic problem. A classification system would have to be

re-trained on a regular basis in order to be up-to-date. While both logistic regression and SVM had similar performance and accuracy, we found that the logistic regression model trained much quicker than the SVM model, which made it a better candidate. To speed up the training process of our models, we looked at conducting dimensionality reduction through PCA. However, we found that there was a significant tradeoff between speed and accuracy in this instance as well. Some of our decisions throughout the experimental process were instrumental in developing a well-performing classification system. Our decision to harvest and get an additional dataset from the Fake News corpus was key in generalizing the model and making it less prone to overfitting. In addition, our decision to compare the effectiveness of word embeddings in this classification problem showed us that embeddings did not make a significant difference to the performance of our models. This experiment helped us realize that increasing the complexity of a model does not necessarily increase model performance. In this project, the simpler models performed as well, if not better, than the more complex models.

We also faced some difficulties during this experimentation process. The initial difficulty we faced across every model was overfitting of the training data. This became less of an issue when incorporating the new dataset to the existing one but it still had to be addressed through regularization techniques such as L1 and L2 considerations. Another difficulty we faced was the fact that the dataset was far larger than what our local machines could handle. Google Collab was used to great effect for working with the data due to the larger resources it has available, such as access to the GPU and additional RAM. Even so, we still found that certain computations would crash the notebook environment and thus we would lose out on that experimentation session. These discovered constraints influenced our model creation and experimentation to some degree, such as the previously noted non-embedding neural network model. As with any problem of this nature, the tradeoff between creating the optimal model and actually being able to execute it on the given hardware helped narrow down our experimentation attempts to a more manageable timeframe for the scope of this project.

Overall, our EDA helped us develop good intuition, which guided us through our preprocessing and development process. Wide experimentation and comparison between different techniques helped us evaluate tradeoffs between key metrics, such as accuracy and computation cost. Finally, visualizing model performances gave us the ability to evaluate each model quickly and efficiently.

Despite multiple iterations of training and refinement of our models, we found that models would tend to overfit on the training data. For future work and research, we recommend training this classification system on a much larger and varied corpus of data. This would help further generalize the classification models and it will contribute towards the development of a robust and well-performing system that can be deployed in the real world.

V. Project Plan & Task Distribution

We split up our project into eight key objectives and tasks. These included data exploration, data preprocessing, EDA, model development, embedding analysis, visualization of results, the project report, and the presentation. As a team we jointly collaborated and worked on the data exploration, data preprocessing, EDA, project report, and presentation. We also split up and divided up the rest of the tasks to facilitate a faster and more efficient research and development process. Asad worked on the model development and training of the logistic regression, random forests, and SVM models on both embeddings and non-embeddings data. He conducted hyperparameter tuning of each of those models using RandomizedSearchCV. Hussein worked on selecting the best performing word-embeddings representation from 14 different pre-trained models by using logistic regression to classify a sample of the dataset and evaluate each embedding type on that dataset. He also contributed by hosting data on the cloud for on-demand access. Sudha worked on the process of generating model performance reports. She created a visualization dashboard that presented different metrics of model performances using Dash/Plotly Chart Studio. She also deployed this dashboard on Google Cloud in order to enable constant and real-time access to these visualizations. Jacob worked on the model development of the neural network models. He created deep neural networks for both embeddings and non-embeddings. Additionally, he spent multiple iterations fine-tuning and improving upon both models.

VI. References

[1] Natali Ruchansky, Sungyong Seo, Yan Liu (2017). CSI: A Hybrid Deep Model for Fake News Detection

[2] Xiang Lin, Xiangwen Liao, Tong Xu, Wenjing Pian, Kam-Fai Wong (2019). Rumor Detection with Recurrent Hierarchical Convolutional Neural Networks

[3] Jacob Devlin, Ming-Wei Chang, Kenton Lee, & Kristina Toutanova. (2019). BERT: Pre-training of Deep Bidirectional Transformers for Language Understanding.

[4] Mike Lewis, Yinhan Liu, Naman Goyal, Marjan Ghazvininejad, Abdelrahman Mohamed, Omer Levy, Ves Stoyanov, & Luke Zettlemoyer. (2019). BART: Denoising Sequence-to-Sequence Pre-training for Natural Language Generation, Translation, and Comprehension.

[5] Matthew E. Peters, Mark Neumann, Mohit Iyyer, Matt Gardner, Christopher Clark, Kenton Lee, & Luke Zettlemoyer. (2018). Deep contextualized word representations.

[6] Daniel Cer, Yinfei Yang, Sheng-yi Kong, Nan Hua, Nicole Limtiaco, Rhomni St. John, Noah Constant, Mario Guajardo-Cespedes, Steve Yuan, Chris Tar, Yun-Hsuan Sung, Brian Strope, & Ray Kurzweil. (2018). Universal Sentence Encoder.

[7] Tomas Mikolov, Kai Chen, Greg Corrado, & Jeffrey Dean. (2013). Efficient Estimation of Word Representations in Vector Space.

[8] Pennington, C. (2014). GloVe: Global Vectors for Word Representation. In *Proceedings of the 2014 Conference on Empirical Methods in Natural Language Processing (EMNLP)* (pp. 1532–1543). Association for Computational Linguistics.

[9] Zhenzhong Lan, Mingda Chen, Sebastian Goodman, Kevin Gimpel, Piyush Sharma, & Radu Soricut. (2020). ALBERT: A Lite BERT for Self-supervised Learning of Language Representations.

[10] Colin Raffel, Noam Shazeer, Adam Roberts, Katherine Lee, Sharan Narang, Michael Matena, Yanqi Zhou, Wei Li, & Peter J. Liu. (2020). Exploring the Limits of Transfer Learning with a Unified Text-to-Text Transformer.

[11] Thomas Wolf, Lysandre Debut, Victor Sanh, Julien Chaumond, Clement Delangue, Anthony Moi, Pierric Cistac, Tim Rault, Rémi Louf, Morgan Funtowicz, Joe Davison, Sam Shleifer, Patrick von Platen, Clara Ma, Yacine Jernite, Julien Plu, Canwen Xu, Teven Le Scao,

Sylvain Gugger, Mariama Drame, Quentin Lhoest, & Alexander M. Rush. (2020). HuggingFace's Transformers: State-of-the-art Natural Language Processing.

[12] Ahmed H, Traore I, Saad S. (2017) “Detection of Online Fake News Using N-Gram Analysis and Machine Learning Techniques. In: Traore I., Woungang I., Awad A. (eds) Intelligent, Secure, and Dependable Systems in Distributed and Cloud Environments. ISDDC 2017. Lecture Notes in Computer Science, vol 10618. Springer, Cham (pp. 127-138).

[13] Ahmed H, Traore I, Saad S. “Detecting opinion spams and fake news using text classification”, Journal of Security and Privacy, Volume 1, Issue 1, Wiley, January/February 2018.

[14] Quoc V. Le, & Tomas Mikolov. (2014). Distributed Representations of Sentences and Documents.