Parallelization of a NLP Fake News Detection Algorithm, using Python

1st Milan Tasevski
Ss. Cyril and Methodius University in Skopje
Faculty of Computer Science and Engineering
Skopje, North Macedonia
milan.tasevski@students.finki.ukim.mk

2nd Vladimir Zdraveski Ph.D.
Ss. Cyril and Methodius University in Skopje
Faculty of Computer Science and Engineering
Skopje, North Macedonia
vladimir.zdraveski@finki.ukim.mk

***Abstract*—As we go deep into the digital era, more and more people tend to use the Internet as their primary source of information. However, as this trend continues to rise, so does the worrying fact that a huge portion of the information we take for granted, is in fact, fake news. In this paper we are going to analyze a Machine Learning algorithm in Python for detecting fake news, using BERT, TensorFlow and PyCaret. We are going to parallelize the training process of the best of the different models we are going to use for solving this problem and analyze the results. In the end, we are going to comment, only briefly, whether this kind of parallelization can in fact, affect the accuracy of the algorithm in a negative way compared to the sequential implementation of the same algorithm. The dataset we are going to use [1] is open-source, and made from real news, which fits perfectly into the problem we are trying to solve.**

***Index Terms*—parallel processing, text classification, dataset**

I. Introduction

We live in a world where smartphones, tablets and all of the other multifunctional gadgets are an inevitable part of our life. We spend huge portions of our days using these devices for various reasons and they have slowly become a primary source of information for the majority of the people using them. However, as these trends grew exponentially so did the amount of fake news all over the Internet. People started noticing that and the Internet is now becoming a less and less reliable source of information for a lot of it's users. One of the primary goals of Informatics, easy access to information, is starting to shatter because of this huge problem. According to a study by Amy Watson [2], more than 80% of consumers in the United States reported having seen fake news on the coronavirus outbreak, highlighting the extent of the issue and the reach fake news can achieve. Watson also mentions that the public's confidence in their ability to recognize fake news is on the increase. But this is hardly enough to solve the problem.

Artificial Intelligence plays a huge role in detecting fake news over the Internet. Using all sorts of Machine Learning models, researchers have suggested a lot of solutions, some of them more advanced than the others. From different Neural Networks, to Random Trees and Random Forests, these powerful algorithms offer us ways to solve this problem that are impossible to develop without them.

However, the main drawback of almost all of the ML algorithms is the time and computational power needed to train the algorithm. That is, to make the computer learn from the data and apply that knowledge on unseen instances of new data. Computer Scientists have been trying to speed up this process and parallel processing is one way to do that. The idea of optimizing the training phase is to split the data and then run the algorithm on several instances of a processor, or work stations, each using a portion of the split data and then combining the results. That way, the work is split and done in parallel instead of doing it sequentially, thus hopefully reducing the time needed for training as much as possible. However, this may affect the accuracy of the algorithm since combining results could lead to additional error in some cases.

Our goal is to see if the training of such algorithm (specifically a Random Forest Classifier, as it will be clear why) can be paralellized so that we reduce the time needed for training and not lose much of the accuracy. We will explore related scientific papers dealing with the classification problem of detecting fake news as well as paralellizing the models used, then see how they relate to the work done in this paper. Then, we will overview the solution starting from explaining how the model works and ending with a 1000 foot view of the parallel implementation. After that, we will see the algorithm in action and we will play around with the parameters and number of processors used in order to find the most optimal execution, noting the results we get along the way. At the end, we are going to compare the results and do an overview of the solution in the conclusion. We will also mention how our work can be further continued in the final section.

II. Related work

The turning point of the trend of using the Internet as a primary source of information seems to be around 2009, when an article was published by Reuters [3], stating that more than 50% of U.S. adults said they would select the Internet if they had to choose only one source of news available to them. However, just a small fraction of U.S. adults considered social websites such as Facebook and MySpace as a good source of news. Since then, as the area of Machine Learning grew exponentially, according to M. I. Jordan and T. M. Mitchell [4], great opportunities for detecting Internet fake news arose. Many models were made using different approaches [5] and paradigms [6]. Some of them use pretrained algorithms such as TFIDF and CV and W2V as features for processing textual data [7]. Others, like G.E. Agudelo, O. J. S. Parra and J. B. Velandia propose “CountVectorizer”, “TfidfVectorizer”, a Naive Bayes Model and natural language processing for the identification of false news in public data sets [8]. Inspired by the problems with fake news in 2016 U.S. presidential election cycle, A. Prabha, Aisuwariya, V. K. Kiran, V. Shriram propose an even more innovative and implementable approach for online fake news detection through machine learning [9]. In this approach, they solve the problem by using the standard set of machine learning algorithms like K -Nearest Neighbors, Support Vector Machine, Naive Bayes and Passive Aggressive Classifier, concluding that PAC classifier produces the best accuracy of even 94.63%!

Regarding parallelizations of machine learning algorithms, a huge impact was made by R. Bekkerman, M. Bilenko and J. Langford [10] in their book ”Scaling up machine learning: Parallel and distributed approaches”, laying the foundations of this huge area and inspiring future work of a lot of scientists. One of the first papers published of this exact problematics was done by C. Chu, SK. Kim and Y. Lin [11], successfully achieving a 1.9 speedup on a dual processor and up to 54 times speedup on 64 cores, paralellizing the map-reduce for machine learning on multicore.

III. Solution overview

First, we develop a model using the Universal Sentence Encoder and PyCaret. The Universal Sentence Encoder is an Encoder that has already been trained on several classification tasks, permitting to change each instance of the dataset into a 512 dimensional vector. After splitting the train-test sets, we want to determine which of the most famous and efficient classification algorithms is most suitable for our specific task. PyCaret is the tool that helps us find this algorithm. Since the full dataset contains more than 40000 entries and we don’t want to use 512x40000 numbers, it is wise to perform a Principal Component Analysis (PCA) dimensional reduction. At this point, we can visualize the data (after splitting it into 3 components), as it is shown in Fig. 1.

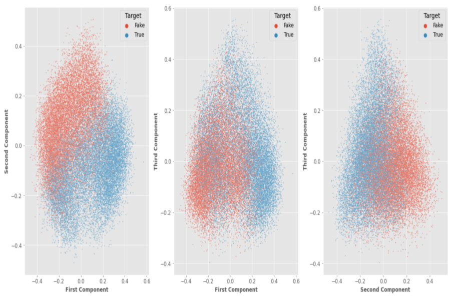

Fig. 1. The data are almost linearly separable.

Using PyCaret we do the setup. First, with only one CPU job and no usage of GPU, and then we modify to see the changes (using more than one CPU). Here, we only care to find the optimal model, we are going to analyze the results of the parallelization in section IV. According to PyCaret’s computations, the most optimal model seems to be the one using the Random Forest Classifier, reaching an accuracy of 0.8834 which is no surprise given that the algorithm is one of the most powerful in the Machine Learning paradigms. In order to paralellize the most expensive tasks a RFC does, we first need to do an overview on how RFC works and dive into the conceptual design of the model. A Random forest consists of decision trees, so this is our starting point.

A. Decision Tree

A Decision Tree, or CART (Classification and Regression Tree) is a predicting model capable of performing both classification and regression tasks [12]. The idea of a decision tree comes from the way people make decisions, which makes them easy to understand even though their algorithmic structure has a tendency to seem quite complex. Each inner node is a question and each edge is an option of answering that question. The leaves are the outcomes. You start answering from the root and repeat until you end up in a leaf, where you have the outcome.

The process of learning is actually the process of building the tree. To do the split in each node, the algorithm finds the feature (or feature threshold) whose split best separates the classes. This ”best” split is actually the split from which we get most information gain, based on the **Entropy** (for classification tasks) or **Root Mean Squared Error (RMSE)** (for regression tasks). The mathematical formulas representing each of these are shown in Table 1. Additionally, Decision Trees are greedy algorithms, since they attempt to make the optimal split in each step.

TABLE I
Mathematical formulas used in decision trees

Impurity	***Task***	***Formula***
Entropy	Classification	$\sum_{i=1}^{C} -f_i log(f_i)$
Mean Square Error	Regression	$1\frac{}{N}\sum_{i=1}^{N}(y_i-\mu)^2$

Without regularization (pruning in tree algorithms), the tree splits until each node is left with a single outcome. This means that a simple decision tree algorithm will fail to generalize to unseen instances - overfitting will occur. This is the biggest problem regarding decision trees, since each pruning method results with a reduction of the accuracy during training. Thus, a new idea was born - Random Forest.

B. Random Forest

The Random Forest method was first developed by Leo Breiman in 2001 [13]. It is an ensemble method that groups multiple Decision trees, using bootstrap aggregation (bagging) [14], achieving a powerful predictor that has better generalization error than an individual Decision tree.

Bagging is a method that generates multiple versions of a predictor, thus achieving an aggregated predictor. This predictor is called the Random forest, whose output is the average output of each Decision tree (for regression tasks), or the relative majority of the predictions of each model (for classification tasks). The randomness is achieved by randomly sampling instances from the training set with replacement and using the new bootstrap data as the new training set (Fig. 2). This is also where paralellization can be implemented.

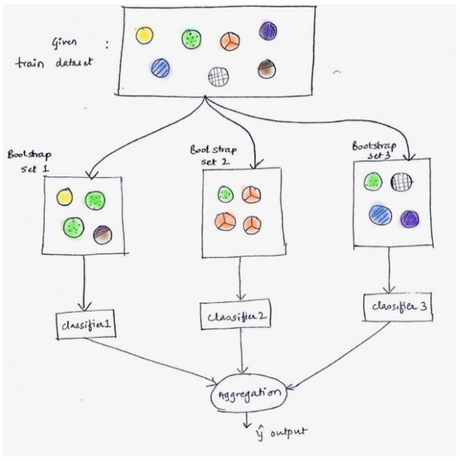

Fig. 2. An example of bagging.

Although each tree is overfitting regarding it's own data, the fact that each tree is trained on different data means that they make different errors, resulting in the models complementing each other in correctness. This is the strength of the Random forest.

The biggest problem of the Random forest is the complexity of the algorithm. Sequential implementation means that the training process is going to take a lot of time. Add in paralellization and we can hope for some good results.

Since our main problem is a classification problem (fake news or not), from now on we are going to discuss a parallel solution for a Random Forest Classifier.

C. *Paralellization*

There are multiple ways to implement a parallel solution to the Random Forest Classifier. Since the structure itself consists of multiple components (each component is a Different decision tree) this is the place where we are going to look for paralellization. Furthermore, we are going to focus on the training process since this is the most expensive one.

We mentioned that the paralellization is going to happen in the bagging process. We can chose the number of Decision trees in the Random forest to be divisible by the number of processes (threads, microprocessors or different machines) and then divide them evenly such that each process works only on their own set of decision trees (Figure 3). We can call these sets mini-Random forests, since each of them is actually (by definition) a random forest. Then, we can combine the results from each process and take the majority of the predictions as the final result. An interesting point of view is that this is actually how a Random forest works, only this time, instead of decision trees we have processes and instead of a single output (ex. fake news or not) we have multiple outputs that are combined together. Not that for a regression task the same would apply, differentiating only in the combination of results of each processor. Since the outputs are going to be real numbers, we could combine them by taking the mean (majority would not work for obvious reasons).

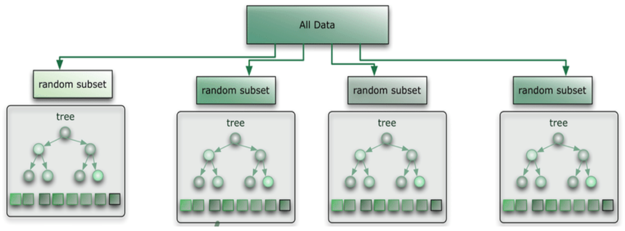

Fig. 3. Dividing the data into 4 random subsets. Each process works only on their own set of decision trees. In this case the sets each only have 1 element for simplicity.

The advantages of this kind of paralellization are vast. We don't spend a lot of time in communication since the processes only communicate at the end, when we combine the results. We also gain a big speed-up during the training phase. The time needed to train a parallel Random forest is going to be the maximum of the times of each processes, plus the time needed to combine the results. The disadvantage is that each process has to work on its mini-Random forest sequentially. Let's see the expected speed up in action.

IV. Results

In this section, we are going to create a testing environment to decide which model (using the methods from the beginning of section 3) to use. Then, we are going to see the difference of the training times using the sequential implementation of the algorithm opposed to the parallel one, where we are going to play around with the number of processor cores we are using, just to see the difference in both time and accuracy.

I should note that these are going to be only the initial results of another dataset that is very similar to ours but generated in runtime using the make_classification function that the sklearn library offers. We set the number of dimensions to 20 and this same number can be achieved using PCA in the original dataset. This is made because of practical reasons, because the original algorithm uses vectorizing that takes a lot of time and makes it hard to experiment, and this part is not in the focus of the paper. This can be easily extended to use the original dataset, with the reduced number of dimensions.

A. *Sequential vs Parallel*

In section 3 we visualized the data and saw that the accuracy the Random Forest Classifier gives us is 0.8834, which was quite impressive. With this new improvised dataset, the situation is better: the sequential implementation gives us an accuracy of 0.947. Now, this is an excellent result, but let's see if we could keep it up with the parallel implementation.

The solution is implemented in a way that it can be executed and tested on different types of multi core processors. We just have to make sure the data is split evenly among the cores we are working with and we are good to go. For the experimental purposes, we choose the training data to have 9600 entries, since we are going to use all the possible variations with the cores available (the number is divisible by 2, 3, 4, 5 and 6). The experimental methods will be done on a machine with a Intel® Core™ i7-8750H Processor with 6 cores.

A visual representation of the speed-up we gain is shown in Fig. 4. The training time using the sequential implementation is 9.202 seconds, and we can see the gradual improvement as we introduce more cores. Using 2 cores it takes 5.154 seconds, 3 cores - 3.472 seconds, 4 cores - 2.731 seconds, 5 cores - 2.249 seconds and 6 cores - only 2.183 seconds. We have an improvement by 4.22 times! Obviously, it's an extremely positive result.

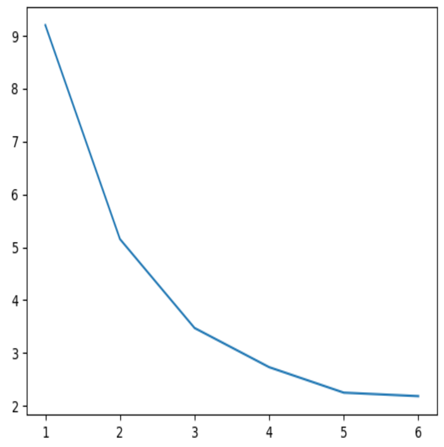

Fig. 4. The vertical axis is for the training time in seconds, the horizontal one is for the number of processors used.

Even so, we need to see the change of accuracy in order to draw a final conclusion, since the accuracy is a vital element of any Machine Learning model.

B. Change of Accuracy

The one last modification of the code is going to be the evaluation of the model, specifically the accuracy of the predictions. We already mentioned that the sequential implementation gives us an accuracy of 0.947. The monitoring of the change is shown in Fig. 5.

As we can see, first there is a slight improvement and then we have a decrease of the accuracy as we introduce more processors. But, this is not a significant decrease: the difference between the best and the worst implementation is only 0.004, or 0.4%. It's not too good to be true. Actually, it makes sense, as we mentioned in section 3C: a parallel Random forest is just another Random forest. We don't have a change in the structure and the lack of accuracy reduction is to be expected.

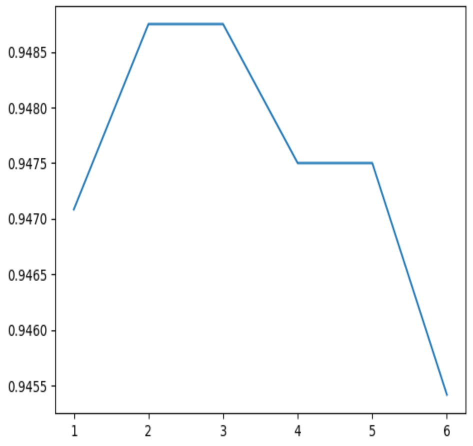

Fig. 5. The vertical axis is for the accuracy, the horizontal one is for the number of processors used.

V. Conclusion and future work

In this paper we explored a way to paralellize the training phase of a Random Forest Classifier for predicting fake news on the Internet, as this was the best model according to PyCaret's results based on the accuracy of the models used. We discovered that, not only is this model easily paralellizable, but also gives us great results in both reducing the cost of training and keeping a high accuracy. This was to be expected as the model's structure is such that can be paralellized using not much effort.

As a future activity, we could execute the algorithm using a different machine, or even a cluster of multiple computers to see if the results are going to depend of the kind of hardware used. Similar results are expected, but that has to be confirmed.

The next step of working on this problem could be to extend the model to work on the original dataset with 40000 entries that we visualized before. We can reduce the number of dimensions to 20, as is mentioned in section 3 and 4 and continue from there. We could also work on paralelizing the vectorization as this is the most expensive process of the exact problem we are trying to solve, although this would mean shifting our paradigm and way of thinking drastically, as it requires different approach than this paper's.

Moreover, as we only explored the Random Forest Classifier as the algorithm that has the best accuracy, we could look at other parameters, such as precision or recall and have a different conclusion of what we consider ”best”. For example, out of the 13 models PyCaret works with, RFC is only the third best if we compare the recall, performing worse than Quadratic Discriminant Analysis and Naive Bayes. For future work, we could consider one of these algorithms and try paralellizing their training phase, hoping for some interesting results.

References

[1] Clément Bisaillon, ”Fake and real news dataset”, https://www.kaggle.com/clmentbisaillon/fake-and-real-news-dataset?select=True.csv

[2] Amy Watson, ”Fake news in the U.S. - statistics facts” https://www.statista.com/topics/3251/fake-news/dossierKeyfigures

[3] Reuters Staff, ”Internet most popular information source: poll” https://www.reuters.com/article/us-media-internet-life-idUSTRE55G4XA20090617

[4] M. I. Jordan and T. M. Mitchell, “Machine learning: Trends, perspectives, and prospects” Vol 349, Issue 6245, pp. 255-260, 17 Jul 2015.

[5] Z. Khanam, B. N. Alwasel, H. Sirafi and M. Rashid2 “Fake News Detection Using Machine Learning Approaches”, Citation Z. Khanam et al 2021 IOP Conf. Ser.: Mater. Sci. Eng. 1099 012040

[6] Dimitrios Katsaros, George Stavropoulos, Dimitrios Papakostas, ‘Which machine learning paradigm for fake news detection?” [2019 IEEE/WIC/ACM International Conference on Web Intelligence (WI), 14-17 Oct. 2019]

[7] Sairamvinay Vijayaraghavan, Ye Wang, Zhiyuan Guo, John Voong, Wenda Xu, Armand Nasseri, Jiaru Cai, Linda Li, Kevin Vuong, Eshan Wadhwa ”Fake News Detection with Different Models”, 15 Feb 2020

[8] Gerardo Ernesto, Rolong Agudelo, Octavio José Salcedo Parra, Julio Barón Velandia, ”Raising a Model for Fake News Detection Using Machine Learning in Python”, [Conference on e-Business, e-Services and e-Society, IEEE 2018: Challenges and Opportunities in the Digital Era] pp 596-604

[9] Anushaya Prabha, T.; Aisuwariya, T.; Vamsee Krishna Kiran, M.; Vasudevan, Shriram K., ”An Innovative and Implementable Approach for Online Fake News Detection Through Machine Learning”, [Journal of Computational and Theoretical Nanoscience], Volume 17, Number 1, January 2020 pp. 130-135

[10] R. Bekkerman, M. Bilenko, J. Langford, ”Scaling up machine learning: Parallel and distributed approaches”, 2011

[11] C. Chu, SK. Kim and Y. Lin, ”Map-reduce for machine learning on multicore”, 2007

[12] Breiman, L., J. Friedman, R. Olshen, and C. Stone. (1984): Classification and regression trees.

[13] Breiman, L. (2001). Random Forest

[14] Breiman, L. (1996). Bagging Predictors