# CheckThat! 2019 UAICS

Lucia-Georgiana Coca, Ciprian-Gabriel Cusmuliuc, Adrian Iftene

Alexandru Ioan Cuza University, Faculty of Computer Science, Iasi, Romania
{georgiana.coca, ciprian.cusmuliuc, adiftene}@info.uaic.ro

**Abstract.** Investigative journalists or detectives lose a lot of time to prove a certain claim, searching for the source or evidence to support this assertion often by hand. In this context, in order to address this problem, the 2019 CLEF CheckThat! comes with two tasks: (1) Check-Worthiness and (2) Evidence & Factuality. Our group participated to the first task whose aim is to evaluate the check worthiness of a political claim in a debate. The method to achieve the goal of the task was to represent each claim by a feature vector and feed it to a machine learning classification algorithms in order to classify if the claim is check-worthy or not. We submitter 3 runs, one primary and two contrastive, the primary being a Naive Bayes, the first contrastive Linear Regression and the second one SVM. The best result we achieved using the official measure MAP was with the Naive Bayes, the second best was the SVM and the third was the Linear Regression. This paper presents the details of our approaches.

**Keywords:** CheckThat!, SVM, Naive Bayes, Linear Regression.

## 1 Introduction

The popularity of social networks has increased significantly in recent years, and reading news on these social networks has become a natural activity for all users. The news is instantly transmitted to these networks, which are read quickly, marked with opinions (see Facebook1), retransmitted (retweet on Twitter2, share on Facebook) without having to check many times whether they are true or false news.

This problem has also affected the political environment; growing political unrest in many countries has made politicians exchange accusations in diverse political debates, some that are more accurate than others are. The challenge is thus presented to us to first check the need to verify a political claim and then to verify if it is factually true. The first task consists of classifying claims from a presidential political debate, each candidate can make a claim and the others have a short time to issue a response, each

[1] https://www.facebook.com/
[2] https://twitter.com/

of them has the right to make accusations in order to convince the audience about their political wittiness (Atanasova et al., 2019).

In order to investigate the check-worthiness of a claim we have been provided with multiple presidential transcripts from the last elections in the United States. The goal is to provide a score for each line in the transcript, score that would signify the priority for fact checking and would be an input for task 2 (Elsayed et al., 2019).

This paper describes the participation of team UAICS, from the Faculty of Computer Science, Alexandru Ioan Cuza University of Iasi, in Task 1 at CLEF 2019.

The remaining of this paper was organized as follows: Section 2 gives a description of the task. Section 3 details the model we developed and the submitted runs and then Section 4 details the results we obtained, finally Section 5 concludes this paper.

## 2 Task Description

### 2.1 Objectives

The objective is to provide a score for each line of a presidential debate[3], this score signifying the worthiness of the line to continue for fact checking[4] (which is task 2) thus the objective is to create a filtering layer for the second task.

Given the fact that in a political debate things evolve quickly a manual checking would, be very cumbersome and slow, thus the need arises for automated checking in order to make the public more informed about the discussion and discourage fake information

### 2.2 Dataset

There were two datasets available, one to train the model and one for testing the model. They were both political debates transcripts from the last United States presidential elections.

The dataset was consisting of the following columns: *line no, speaker, text* and *label*; the test file did not have the label available. The label was a binary one, zero signifying that *the sentence should not be fact checked* and one to be fact checked. A concrete example with label 1 would be from *Trump: "So Ford is leaving"* and one with label 0 would be from *BLITZER: "Let's begin with Senator Sanders"*.

The training had 19 files and the test had seven files.

### 2.3 Evaluation metric

The task has been evaluated according to the official organizer's measures. The official measure is MAP (Beitzel et al., 2009) which calculates the usual mean of the average precision. Other measures used are the Mean Reciprocal Rank (Craswell, 2009) which allows obtaining reciprocals of rank of the first relevant document, as well as Mean

[3] https://sites.google.com/view/clef2019-checkthat/task-1-check-worthiness

[4] https://sites.google.com/view/clef2019-checkthat/task-2-evidence-factuality

Precision at k, which performs the average of k best candidates. Details on the measures used can be found in the task overview.

Evaluations are carried out on primary and contrastive runs, the resulting metrics are as described above. Each participant has the right to three models, one primary and two secondary (contrastive).

## 3 Methods and runs

Right from the start, we decide we only want to use the training data provided and no other external information for our models, thus only the presidential debates were available.

We selected multiple machine-learning algorithms in order to test which one would be best for our problem. We started with the classical ones, Decision Trees (Dobra, 2009) and Naive Bayes (Rennie, 2003), and then we moved to more advanced algorithms such as SVM (Liu, 2009), Random Forest (Ho, 1995), Logistic Regressions (Hosmer, 2000) and finally we tested a neural network, a Multilayer Perceptron Classifier. We use machine-learning algorithms in previous editions of CLEF (Iftene et al., 2009), (Iftene et al., 2012), (Iftene et al., 2013), and (Cristea et al., 2016).

In order to verify our algorithms, given we had no validation data; it was decided to split the training data 70-30, 70% would be used for training and 30 for measuring the performance.

To ease our implementation we used PySpark, combined with the PySpark MLlib that contains prebuild, ready to use machine-learning algorithms, we decided to use this tool in order to benefit from the parallel processing power of PySpark to scale the application in order to process large amounts of data.

Our metrics were based on sklearn's metrics so in the end we would have the Precision, Recall and F1 of the classifiers but also the confusion matrix. In addition to our metrics, in order to comply with the organizers requirements, we also used their provided metrics that are the following: R-Precision, Average Precision, Reciprocal Rank and Precision@k so in the end we would have a multitude of metrics that would help us better measure our classification efforts.

### 3.1 Training and test data

The data provided contained presidential elections debates from the United States in 2016. The data was of two main categories, training and test. The training had 19 files while the test had seven files. The main difference of the test and training was that the test has a missing label column that is the classification category of the phrase.

One training example with the available columns would be the following:

**Table 1**. Training example.

| Line no. | Speaker | Text | Label |
|---|---|---|---|
| 1 | Trump | So Ford is leaving. | 1 |

The test had the following format:

**Table 2.** Test example.

| Line no. | Speaker | Text |
|---|---|---|
| 1 | Sanders | They went to the DNC quietly. |

From the training data we made several decisions: the speaker is not relevant for the algorithms and it would make it more biased to certain decisions (which we do not want, so we excluded it), we would not exclude the speaker "SYSTEM" with phrases such as "(APPLAUSE)" as they are all very similar and the label 0 would be enough to make the algorithms realize that it has to predict with 0 and that besides tokenization we can run feature extraction algorithms without much pre-processing at all.

### 3.2 Preprocessing and feature extraction

Before feeding the data to the machine learning algorithms, we had to preprocess the text and extract features. This section describes in detail this process in order to fully understand the training data fed to the algorithms.

As said in the previous section, we did not take into consideration the speaker column in the classification process, on the "text" column and the "label". For preprocessing, we only applied a "Tokenizer" for each line, taking text (such as a sentence) and breaking it into individual terms (words). After the tokenization process we removed the stop words with the "StopWordsRemover" class, given the fact that the texts are in English we only removed stop words from this language.

We did not take into account irony or sentiment analysis in the preprocessing part as we believed feature extraction could represent this indirectly however we were aware this could affect certain edge cases of the classification.

After we preprocessed the text it was necessary to extract features from the text, we did this using a multitude of methods trying to find the perfect fit for every algorithm. The best results were obtained with two main methods: TF-IDF and CountVectorizer (Convert a collection of text documents to a matrix of token counts).

For the first method, TF-IDF, we used it with Logistic Regression and Multilayer Perceptron, the implementations in Pyspark are HashingTF and then next in pipeline would be IDF. The TF-IDF would create a feature model where the term frequency would yield informational value to classification algorithms. We decided to use HashingTF in order to make the implementation faster as this would create a feature map where a raw feature is mapped into an index (term) by applying a hash function, after which the IDF would take the generated term frequency vectors to fit which scales each feature and down-weighs features that appear frequently in the corpus.

As for settings, we had to fine-tune the preprocessing methods and the final form of them is that for Logistic Regression the number of features of HashingTF was 262,144 and for Multilayer Perceptron was 5,000 (as this would force us to create the same number of input layers for the neural network, we had to scale it down to this value).

The settings for IDF for both algorithms are the same, the minimum number of documents in which a term should appear for filtering is 0.

For the second model, CountVectorizer, we used it with Naive Bayes and SVM, the implementation is with the same name, after the CountVectorizer we applied IDF. The CountVectorizer is very similar to HashingTF, the main difference being that the first one is reversible (because of not doing hashing), is more computationally intensive however it does not reduce the dimension, having lower informational loss. We also tried HashingTF on these models however we obtained worse results that is the logic behind switching the feature extraction algorithm. The settings of the CountVectorizer are the following: minimum term frequency is 1 and so is the minimum definition frequency, the maximum definition frequency is 2^63-1 and the vocabulary size is 2^18.

### 3.3 Models

After we preprocessed the data and extracted the feature with the methods described in the previous sections, in this section we will talk about the algorithms used to fit the extracted features.

Our approach was to use a few diverse algorithms and select the best three - one primary and two contrastive. We started investigating these models by looking into what participants of CLEF CheckThat 2018 did, the most notable algorithms being Multilayer Perceptron and SVM but also Random Forests.

We started with the classical ones, Decision Trees and Naive Bayes, and then we moved to more advanced algorithms such as SVM, Random Forest, Logistic Regressions and finally we tested a neural network, the Multilayer Perceptron Classifier. As stated in the beginning of this section the algorithms had the implementation based in PySpark[5]. After we trained and measured the performance using the aforementioned metrics, from these initial six algorithm only four were left: Logistic Regression, Naive Bayes, SVM and Multilayer Perceptron, these had the best results and we could further improve them.

The Naive Bayes was right from the start one of the top performers of our tests, so naturally it received the most attention. The settings used for this algorithm were rather slim, the smoothing was set to 1 and the model was multinomial (given how we are classifying on word counts from the text).

The next algorithm that caught our eye was the SVM that had very interesting results, granted not as good as Bayes but very notable, we used this SVM's hyperplane to classify the multidimensional feature matrix. As for settings the SVM is using a linear kernel, the maximum iterations are set to 100 and the regression parameter is set to zero.

The third best algorithm was the Logistic Regression. Having good results close to SVM, with the settings being: the maximum iterations are set to 100, the regression parameter is set to 0 and the label distribution was set to automatically be identified (binomial or multinomial).

[5] https://www.tutorialspoint.com/pyspark/index.html

In contrast to the three algorithms that performed well we had one that yielded less than satisfactory results, even though it had long training times. The Multilayer Perceptron had very low accuracy that is why we chose not to include it in the sent result. The Multilayer Perceptron had the maximum iterations set to 1,500 and no matter how long the training iterations were set the accuracy remained low. The network had 5,000 input neurons (similar to the number of features) and it had two hidden layers with 1,000 and 2,000 neurons while the output layers were reducing to two neurons corresponding to the labels. Probably the unsatisfactory results were much related to the low number of features extracted, but long training times and the fact that the other algorithms had very fast results with high accuracy made us leave this one last.

To conclude this section, for the submission we chose as primary the Naive Bayes, as contrastive one, we chose Linear Regression and finally for contrastive 2 we chose SVM. We only made two submissions, UAICS-1 and UAICS-2, the latter being the final version of our system.

## 4 Results

In this section, the results of the three submissions will be discussed. The official results of our submissions (of team UAICS), ranking fifth out of 12 for the primary MAP metric, are:

**Table 3.** Results

| submission | MAP | RR | R-P | P@1 | P@3 | P@5 | P@10 | P@20 | P@50 |
|---|---|---|---|---|---|---|---|---|---|
| primary | .1234 | .4650 | .1460 | .4286 | .2381 | .2286 | .2429 | .1429 | .0943 |
| contr.-1 | .0649 | .2817 | .0655 | .1429 | .2381 | .1429 | .1143 | .0786 | .0343 |
| contr.-2 | .0726 | .4492 | .0547 | 4286 | .2857 | .1714 | .1143 | .0643 | .0257 |

To detail our results, we got the following places:

- MAP (Mean Average Precision) - 5th place;
- RR (Reciprocal Rank) - 1st place;
- R-P (R-Precision) - 3rd place;
- P@1 (Precision@1) - 1st place;
- P@3 (Precision@3) - 1st place;
- P@5 (Precision@5) - 2nd place;
- P@10 (Precision@10) - 1st place;
- P@20 (Precision@20) - 4th place;
- P@50 (Precision@50) - 6th place.

With the official labeled test files (7,080 lines of test), we can inspect further metrics such as Precision, Recall, F1:

**Table 4.** Precision, Recall and F1 results primary

| **Primary** | **Precision** | **Recall** | **F1** | **support** |
|---|---|---|---|---|
| 0 | 0.98 | 0.96 | 0.97 | 6944 |
| 1 | 0.11 | 0.24 | 0.15 | 136 |
| Micro avg | 0.95 | 0.95 | 0.95 | 7080 |
| Macro avg | 0.55 | 0.60 | 0.56 | 7080 |
| Weighted avg | 0.97 | 0.95 | 0.96 | 7080 |

**Table 5.** Precision, Recall and F1 results contrastive 1

| **Contr.-1** | **Precision** | **Recall** | **F1** | **support** |
|---|---|---|---|---|
| 0 | 0.98 | 0.97 | 0.98 | 6944 |
| 1 | 0.06 | 0.09 | 0.07 | 136 |
| Micro avg | 0.96 | 0.96 | 0.96 | 7080 |
| Macro avg | 0.52 | 0.53 | 0.52 | 7080 |
| Weighted avg | 0.96 | 0.96 | 0.96 | 7080 |

**Table 6. .** Precision, Recall and F1 results contrastive 2

| **Contr.-2** | **Precision** | **Recall** | **F1** | **support** |
|---|---|---|---|---|
| 0 | 0.98 | 0.99 | 0.99 | 6944 |
| 1 | 0.10 | 0.06 | 0.08 | 136 |
| Micro avg | 0.97 | 0.97 | 0.97 | 7080 |
| Macro avg | 0.54 | 0.52 | 0.53 | 7080 |

| | | | | |
|---|---|---|---|---|
| Weighted avg | 0.96 | 0.97 | 0.97 | 7080 |

In addition, the confusion matrices are:

**Table 7.** Confusion Matrix for primary

| **Primary** | **Predicted No** | **Predicted Yes** |
|---|---|---|
| Actual No | 6694 | 250 |
| Actual Yes | 104 | 32 |

**Table 8.** Confusion Matrix for contrastive 1

| **contr.-1** | **Predicted No** | **Predicted Yes** |
|---|---|---|
| Actual No | 6751 | 193 |
| Actual Yes | 124 | 12 |

**Table 9.** Confusion Matrix for contrastive 2

| **contr.-2** | **Predicted No** | **Predicted Yes** |
|---|---|---|
| Actual No | 6875 | 69 |
| Actual Yes | 128 | 8 |

If we analyze the performance of each system the findings are that the Primary - Naive Bayes has the overall best performance having very high detection rate of phrases which are not worthy of checking and also the best rate of detecting cases which are worthy of fact checking. The confusion matrix also confirm that Naive Bayes is a very performant algorithm, from 7000+ lines only 354 were wrongly labeled.

The next contestant to the place of the best algorithm is the Contrastive-1-Logistic Regression, which seems to have a good capability of predicting non-priority fact checking cases but it is much worse than Naive Bayes at predicting the cases that actually have to be fact checked, thus this would be a close second. The confusion matrix goes hand in hand with the above-mentioned metrics, as the algorithm has wrongly classified 317 lines from the test data.

The final contestant to the place of the best algorithm is the Contrastive-2-Support Vector Machine with the highest detection rate of non-priority cases but unfortunately very low performance in detecting priority cases.

Comparing the Precision, Recall, F1 and Confusion Matrix with the official results it can clearly be seen that the best algorithm is still the primary-Naive Bayes, however the second best is contrastive-2 followed by contrastive-1, this may be because contrastive-1 and contrastive-2 were very close to each other.

## 5 Conclusions and future work

In this paper, we proposed three models to solve the CLEF2019 CheckThat challenge (task 1 Check Worthiness) which deals with the evaluation of the check-worthiness of statements in political debates. We used Naive Bayes, SVM and Logistic Regression to train the models on the extracted features from TF-IDF and CounVectorizer. We got good results with these three models, ranking 5th out of 12. We are currently trying to further improve the feature extraction methods and also insert more data into the algorithms, so we would have even better results, also very important for us is to improve our algorithms and also find new ones, we would like to improve on the Multilayer Perceptron and test everything with a Convolutional Neural Network, so that in the future we would get even better results.

**Acknowledgement.** This work is partially supported by POC-A1-A1.2.3-G-2015 program, as part of the PrivateSky project (P 40 371/13/01.09.2016).

## References

1. Atanasova, P., Nakov, P., Karadzhov, G., Mohtarami, Mitra, Da San Martino, G. (2019) Overview of the CLEF-2019 CheckThat! Lab on Automatic Identification and Verification of Claims. Task 1: Check-Worthiness, CLEF 2019, Working Notes.
2. Beitzel S.M., Jensen E.C., Frieder O. (2009) MAP. In: LIU L., ÖZSU M.T. (eds) Encyclopedia of Database Systems. Springer, Boston, MA
3. Craswell N. (2009) Mean Reciprocal Rank. In: LIU L., ÖZSU M.T. (eds) Encyclopedia of Database Systems. Springer, Boston, MA
4. Cristea, A.G., Savoaia, M.M., Martac, M.A., Pătraș, I.C., Scutaru, A.O. Covrig, C.E., Iftene, A. (2016) Using Machine Learning Techniques, Textual and Visual Processing in Scalable Concept Image Annotation Challenge. In Working Notes of CLEF 2016 - Conference and Labs of the Evaluation forum - ImageCLEF2016. 5-8 September 2016, Evora, Portugal.
5. Dobra A. (2009) Decision Tree Classification. In: LIU L., ÖZSU M.T. (eds) Encyclopedia of Database Systems. Springer, Boston, MA
6. Elsayed, T., Nakov, P., Barron-Cedeno, A., Hasanain, M., Suwaileh, R., Da San Martino, G., Atanasova, P. (2019) Overview of the CLEF-2019 CheckThat!: Automatic Identification and Verification of Claims. In Experimental IR Meets Multilinguality, Multimodality, and Interaction, LNCS, Springer, Lugano, Switzerland, September, 2019.

7. Ho, Tin Kam (1995). Random Decision Forests (PDF). Proceedings of the 3rd International Conference on Document Analysis and Recognition, Montreal, QC, 14–16 August 1995. pp. 278–282. Archived from the original (PDF) on 17 April 2016. Retrieved 5 June 2016.
8. Hosmer, David W.; Lemeshow, Stanley (2000). Applied Logistic Regression (2nd ed.). Wiley.
9. Iftene, A., Gînscă, A.L., Moruz, A., Trandabăţ, D., Husarciuc, M., Boroş, E. (2012) Enhancing a Question Answering system with Textual Entailment for Machine Reading Evaluation. Notebook Paper for the CLEF 2012 LABs Workshop - QA4MRE, 17-20 September, Rome, Italy.
10. Iftene, A. Moruz, A., Ignat, E. (2013) Using Anaphora resolution in a Question Answering system for Machine Reading Evaluation. Notebook Paper for the CLEF 2013 LABs Workshop - QA4MRE, 23-26 September, Valencia, Spain.
11. Iftene, A., Trandabăţ, D., Pistol, I., Moruz, A., Husarciuc, M., Cristea, D. (2009) UAIC Participation at QA@CLEF2008. In Evaluating Systems for Multilingual and Multimodal Information Access, 9th Workshop of the Cross-Language Evaluation Forum, CLEF 2008, Aarhus, Denmark, September 17-19, 2008, Revised Selected Papers. Lecture Notes in Computer Science. Vol. 5706/2009, pp. 448-451.
12. Liu L., Özsu M.T. (2009) SVM. In Encyclopedia of Database Systems. Springer, Boston, MA
13. Rennie, J.; Shih, L.; Teevan, J.; Karger, D. (2003). Tackling the poor assumptions of Naive Bayes classifiers. ICML.