École Publique d'Ingénieurs en 3 ans

Report

EXPLAINABLE NEAR REAL-TIME FAKE NEWS DETECTION FROM TWITTER

August 30, 2021

Engineer internship 2nd year – 2020/2021

Matthias ROUSSEL
matthias.roussel@ecole.ensicaen.fr
College year 2020/2021
INFO / ISIA

Supervisor TALTECH: Pr. Sadok BEN YAHIA
Supervisor ENSICAEN: Alain LEBRET

ENSI CAEN
ÉCOLE PUBLIQUE D'INGÉNIEURS
CENTRE DE RECHERCHE

TAL TECH

www.ensicaen.fr

Acknowledgment

I would like to express my sincere thanks and gratitude to Pr. Sadok BEN YAHIA for giving me the opportunity to do this internship face-to-face in Tallinn. I also thank him for his help and supervision during these sixteen weeks.

In addition, I would like to thank my mentors Wissem INOUBLI and Imen BEN SASSI, for their time and precious guidance that have made my work much easier, and my office neighbor Nzamba BIGNOUMBA for his precious advice.

I also thank Alain LEBRET from ENSICAEN for the time he took to supervise my internship in France.

Finally, I would like to thank all the members of the International Relations Office at ENSICAEN for their guidance in finding and preparing an internship abroad in these challenging times.

Contents

Acknowledgment 2

REPORT **5**

1. Introduction 5
2. Subject of the internship 5
3. Related work 6
4. Personal work 7
   - 4.1. Work organization 7
   - 4.2. Big data architecture 8
     - 4.2.1. Twitter stream 9
     - 4.2.2. Classification and post-processing engines 12
     - 4.2.3. Visual analytics module 12
   - 4.3. Machine learning models 13
     - 4.3.1. Presentation of the datasets used 14
     - 4.3.2. Models using numerical features 15
     - 4.3.3. Models using textual data 17
   - 4.4. Incremental learning 19
     - 4.4.1. Experiment design 20
     - 4.4.2. Results 21
5. Conclusion 23

Appendix 29

A Details about machine learning models 29

Summary 33

Résumé 33

REPORT

1. Introduction

As part of my engineering studies at ENSICAEN, I was supposed to do an internship abroad at the end of my second year. After some research, I had the opportunity to do an internship under the supervision of Professor Sadok BEN YAHIA at the Tallinn University of Technology (TalTech) in the Data Science group. Founded in 1918, TalTech is the only university entirely dedicated to technical subjects in Estonia and is the most well-known Estonian university among non-Estonians.

I was interested in doing an internship in data science and machine learning. Indeed, the computer science curriculum at ENSICAEN features some courses in data mining, but I wanted to broaden my skills in this discipline. This internship at TalTech fills this expectation, being centered on machine learning and near real-time big data analytics.

In this report, I will first present the subject and background of the project of this internship. Then, I will detail the architecture of the big data platform that was created. Finally, I will explain the different experiments conducted to build this system's machine learning model and detail their results.

2. Subject of the internship

This internship aims to use machine learning techniques to detect fake news on the internet, particularly on Twitter. Over the past few years, the phenomenon of fake news has become a more and more prevalent problem. It affects the public debate in all areas and has been suspected to influence elections. Worse, misinformation can mislead people on topics such as health or adverse events, threatening lives. In particular, the COVID–19 pandemic period has seen fake news related to the epidemic flourish, fueled by the uncertainty and fear felt by the citizens during this unprecedented period. These fake news pieces may include fake treatments or misleading advice, false information on the origin or severity of the virus, and sometimes conspiracy theories. These false claims can have a tangible impact on the public response to the pandemic, for example, impacting vaccination intent [5], or prompting people to take unverified drugs that can lead to their deaths [7]. These fake news often spread on social media, where people can share them quickly, allowing some claims to reach a vast audience in a very short amount of time while debunking such claims can take a lot of time and research. Because of its limit of characters, Twitter fosters short and catchy claims without lengthy explanations. In particular,

people are very likely to share a link to an article after reading only its title, without reading the entire article that could nuance and refine the claim.[1] This contributes to the spread of inexact, incomplete, or downright false pieces of information.

In this context, it is crucial to detect these misinformation pieces quickly and debunk them to prevent the spread of false claims that could have serious and impactful consequences. However, detecting fake news manually is an almost impossible task, mainly because of the gigantic amount of information shared on the internet nowadays and the speed at which it spreads. Moreover, short claims like those we are likely to encounter on Twitter can be vague without citing any source, making them very difficult to prove or disprove. Therefore, debunking fake news requires much more time, energy, and financial resources than spreading it. With these constraints, it seems natural to try to detect fake news with the help of machine learning techniques that can operate at a very large scale and in near real-time.

The goal of this project is to build a platform that can detect fake news related to COVID–19 in tweets in near-real-time using machine learning techniques.

3. Related work

Using machine learning techniques to detect online fake news has attracted the interest of industry [10] and academia for several years already. In particular, several works have been conducted on the detection of fake news related to COVID–19, with exciting results.

Four types of methods exist to detect fake news: *knowledge-based* methods that compare the claim with existing factual knowledge; *style-based* methods, which focus on the writing style; *propagation-based* methods that observe the spreading patterns of the news pieces; and *source-based* methods, that base their predictions on the different sources of the news pieces [11]. However, these methods are not siloed. Most of the attempts of fake news detection by machine learning and deep learning rely on text classification. The latent features created by the models contain both content/knowledge-based and style-based information. Several surveys have shown that many traditional machine learning algorithms, including *Support Vector Machines* (SVM), *Decision Trees*, *Random Forests*, and *Logistic Regressions* have been used to detect fake news and rumors on social media with promising results, as well as deep learning algorithms such as *Convolutional Neural Networks* (CNN), *Recurrent Neural Networks* (RNN) and in particular *Long Short-Term Memory* (LSTM) networks. Several approaches use a mix of both CNN, and RNN [1]. More recent work using newer pre-trained language models such as BERT has outperformed state-of-the-art results [8].

Research has also been conducted to attempt to provide explainable fake news detection, using in particular social media comments to provide explanations on whether a news article is detected as real

[1] Twitter has recently implemented a feature that suggests users open a link and read its content before sharing it, aiming to reduce this problem by making users more responsible for the links they share.

or fake news [9].

However, although academic research is focused on using machine and deep learning to detect fake news, the engineering process of integrating it with a large-scale near real-time architecture has not been covered.

4. Personal work

The product I built during this internship with the help of my supervisor and mentors is a software platform that must analyze social media messages to classify them between factual claims and falsehoods. This project is divided into two parts: The first step is to design a machine learning model that can detect COVID-related fake news in messages with satisfying accuracy. Then, the big data infrastructure that allows this model to be used at scale, and most importantly in near real-time, must be set up.

It was also an initial goal to provide explanations for the predictions made by the machine learning model, using popular explainable artificial intelligence (XAI) methods. However, because of a lack of time, this part of the internship has been scrapped.

4.1. Work organization

Being the only person dedicating 100% of my time to this project, it was unnecessary to adopt any precise framework for project management. I had bi-weekly meetings with my supervisor and my mentors to update them on my progression and discuss the orientations of my work. I would also contact them any time I had interrogations or issues that would impede my progress.

My work during this internship can roughly be split down into four main phases:

- **Weeks 1-2**: Self-training on the basic concepts of machine learning, deep learning, and natural language processing using online resources.
- **Weeks 3-7**: Building models and comparing the performances of different machine learning and deep learning algorithms on different datasets.
- **Weeks 8-11**: Experiments on incremental learning; benchmarks and comparisons with the classic approach.
- **Weeks 12-15**: Installation of tools on the cluster of machines, integration of the predictive model into a streaming job, development of the full system with near-real-time tweet streaming and visualizations.
- **Week 16**: Final modifications, writing documentation, and internship report.

4.2. Big data architecture

One important constraint is that the system must ingest many publications and make predictions with a near-real-time latency. These constraints make the software impossible to run on a single server. To run this software at scale, the data processing must be executed on a cluster of machines in a distributed way. Several technologies are required for that:

- Apache Spark is a distributed framework for large data processing. The various libraries it incorporates (Spark SQL and Spark Streaming) allow, in particular, to process structured data either in one batch or inside a streaming job.
- Apache Kafka is a distributed streaming platform conceived as a messaging system that enables the transmission of data between several application components with high throughput and low latency. It aims to reduce the need for individual coupling between the application components that share data.

In addition to Apache Spark and Apache Kafka, we use the Hadoop ecosystem to operate the cluster of machines. The two components of the Hadoop ecosystem that we use in this project are:

- Hadoop Distributed File System (HDFS): a distributed file system that handles data storage on the cluster.
- Hadoop YARN: A resource negotiator responsible for allocating resources on the cluster for the distributed tasks.

All these tools are open-source and maintained by the Apache Software Foundation. The programming language chosen for this project is Python. Although it is possible to use Apache Spark with other programming languages such as Scala or Java, Python is a significant language in data science and allows fast prototyping, particularly in an interactive way on Jupyter notebooks. The general architecture of this fake-news detection platform is described in Figure 1.

The messages are pulled from the various APIs by a Python job that pulls them into Kafka. The several components that need to process the data can then consume the messages from Kafka and put the result of their processing into another Kafka topic. This architecture is designed to be easily extensible: it is possible to add new producers that input publications from other sources. In this case, the project was designed primarily to process tweets, so the term “tweet” used in this report is interchangeable with “message” or “publication”.

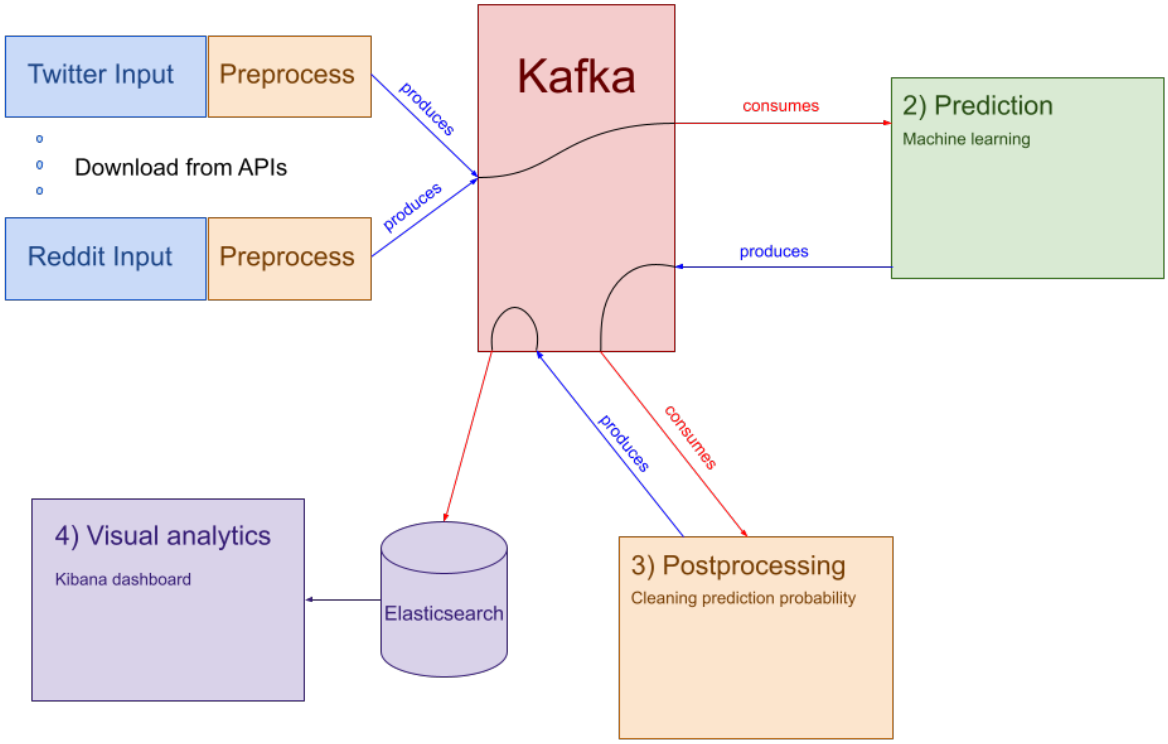


Figure 1. General architecture of the system

4.2.1. Twitter stream

The first components of this architecture are the scripts that download the data in near real-time and push them into Kafka. An arbitrary number of such components can be added to the system to get data from several sources. In this example, I will only detail the component that downloads tweets. A module that downloads comments from Reddit was also added to show how to extend the project, but the machine learning model was not adapted to reliably process this content.

The tweet retrieval module uses the `python-twitter-v2` library to access Twitter's API. The filtered streaming endpoint of the Twitter API[2] allows getting tweets that match one or several filtering rules. Once the stream is launched, all the tweets that match the rules are sent as soon as they are tweeted. The filtering rule used in this application is:

```
(COVID OR covid OR coronavirus) lang:en -is:retweet -is:reply
```

[2] The documentation of this API endpoint is available at `https://developer.twitter.com/en/docs/twitter-api/tweets/filtered-stream/api-reference/get-tweets-search-stream`

Let's explain this rule:

- `(COVID OR covid OR coronavirus)` means that we select tweets that contain at least one of the keywords "COVID", "covid" or "coronavirus".
- `lang:en` means that we select only tweets in English (Twitter tries to automatically detect the language of a tweet).
- `-is:retweet` means that we exclude retweets, as we want each original tweet to appear only once in the results.
- `-is:reply` means that we exclude tweets made in response to other tweets. We, therefore, keep only normal tweets and quote tweets[3].

All the components of this rule are cumulative: only tweets that match all conditions are included in the stream.

When making a query to the Twitter API to get a filtered stream, the desired fields must be provided. Indeed, the API response does not contain all the fields by default, as the information associated with a tweet is sizable[4]. To get the interesting fields, the query is made with the following parameters (This sample shows the code to make the query with `python-twitter-v2`):

```
stream.search_stream(
    tweet_fields=["text", "created_at"],
    expansions=["author_id"]
)
```

Let's explain this query:

- By default, the fields `id` (the tweet's unique id) and `text` (the text of the tweet) are included in the response.
- We also request the fields `created_at` (the tweet's timestamp).
- We ask the API to make an expansion on the field `author_id`. This means that the response will include the related objects. In this case, the User objects[5] are included. They contain the fields `id`, `username` (the user's handle) and `name` (the user's displayed name) by default. Since we don't want any other information about the users, we don't need other parameters. Otherwise, we would need to include the parameter `user.fields` (`user_fields` in `python-twitter-v2`) to the query.

The API response for one tweet is as follows (the JSON is indented for readability)[6]:

[3] A quote tweet is a tweet that references another tweet. This kind of tweet appears when a user adds a comment to a retweet.

[4] The complete reference of a Tweet object in the API response can be found at: `https://developer.twitter.com/en/docs/twitter-api/data-dictionary/object-model/tweet`

[5] Reference: `https://developer.twitter.com/en/docs/twitter-api/data-dictionary/object-model/user`

[6] The data has been changed for privacy.

Listing 1. Sample of the filtered stream API response

```
{
    "data": {
        "author_id": "1234567",
        "created_at": "2021-08-01T15:44:36.000Z",
        "id": "1234567891234567891",
        "text": "#COVID19 Thread https://t.co/xxxxxxxx"
    },
    "includes": {
        "users": [
            {
                "id": "1234567890",
                "name": "Twitter User 1",
                "username": "user1"
            },
            {
                "id": "123456780",
                "name": "Twitter User 2",
                "username": "user2"
            }
        ]
    },
    "matching_rules": [
        {
            "id": 0000000000000000000,
            "tag": "covid EN tweets"
        }
    ]
}
```

The `data` part contains the requested fields, the `includes` part has the requested expansions. In this case, all the users associated with the tweet are included in this part. This is the case of the author of the tweet and other users, such as in this case, the author of the quoted tweet. Finally, the `matching_rules` part tells us which rule this tweet matched to end up in the stream. This could be informational if several rules were set, which is not the case here.

Pre-processing

As shown in Listing 1, the data from the API response is very nested and contains uninteresting information. Since the data can come from various sources and in different formats, it is important to preprocess the data into a common JSON schema. Each modules that gets data from a services formats its data properly before pushing it into Kafka. In the case of the Twitter module, the preprocessing function keeps the fields `id`, `text`, `created_at` and computes the fields `author_username` and `author_name` from the `includes` part of the API response. Although the machine learning model needs only the tweet's text, the other fields are kept to perform analytics in the visualization module. The preprocessed data is pushed into Kafka under the topic `preprocessed_tweets`[7].

[7] The preprocessed data from all input components is pushed under this topic, despite the fact that not all the data comes

4.2.2. Classification and post-processing engines

The next stage through which the tweets[8] pass is the classification engine. This engine is responsible for making the prediction on each tweet. The machine learning model used in this stage is logistic regression. This model was chosen because it is natively available in Apache Spark and, therefore, easy to deploy on the cluster. Its performance is also acceptable, albeit slightly lower than the best model in section 4.3.3. The datasets used to train this models are the CMU-MisCOV19 and HeRA datasets. They are presented in section 4.3.1. The classification engine adds three fields to each tweet (since all the processing is done in Spark, it actually adds three columns to the streaming dataframe[9]):

- `prediction`: the predicted class: `0.0` for a real news and `1.0` for a fake statement.
- `class`: this is also the predicted class, but in a human-readable format: "REAL" or "FAKE".
- `probability`: this is the array of the probabilities that the tweet belongs to each class. Since there are only two classes, the second probability is redundant, but the probability is extracted in the next stage for technical reasons.

The resulting data frame is then pushed into Kafka under the topic `predictions`. A post-processing engine then pulls the data and extracts the probability of the prediction as a single floating-point number. The final data frame is pushed into Kafka under the topic `postprocessed_predictions`.

4.2.3. Visual analytics module

Finally, the results of the predictions are displayed on a dashboard. This dashboard is created with Kibana, which is a visualization add-on for the Elasticsearch search engine. This module works as follows:

1. Elasticsearch pulls the tweets with predictions from Kafka[10] and stores them into an index[11].
2. The dashboard created in Kibana[12] perform queries on the data of the index and display diagrams showing statistics about the tweets and the associated predictions.

The resulting dashboard is shown in Figure 2.

from Twitter. The topic name remained the same despite the system being extended to other services.

[8] Once again, since the machine learning model was trained to process tweets only, the term "tweet" is used here to denote any text that passes through the system.

[9] In Apache Spark, the data is available through a structure called *dataframe*, which is very similar to a table in a relational database, with the exception that the data types are not enforced. When working with Spark Streaming, a streaming data frame is simply an unbounded table in which new rows are added when they come from the stream of data.

[10] This is made possible by Kafka Connect, a component of Kafka that allows to stream data between Kafka and other data systems. A component called connector is provided to describe the communication with Elasticsearch. Using this connector, Kafka Connect pushes data into Elasticsearch using its REST API.

[11] An index in Elasticsearch is a collection of documents. To make a parallel with relational databases, an index corresponds roughly to a table, but the data in Elasticsearch is much less structured.

[12] Kibana allows creating dashboards from its web interface without writing any code.

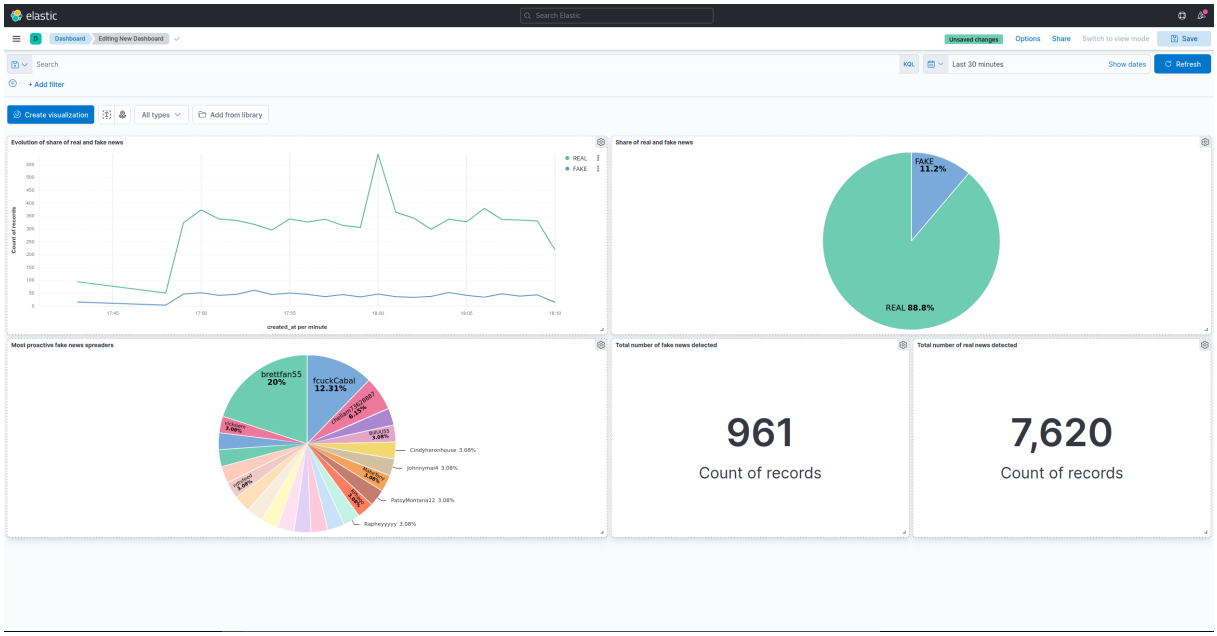


Figure 2. Kibana dashboard showing various statistics about the tweets and predictions, in near real-time.

4.3. Machine learning models

The core piece of this system is the prediction module that is based on a machine learning model. This machine learning model must, given some information about a tweet, predict whether the content of this tweet is true or false. A machine learning model must be fed with *features*, i.e., variables that describe each sample of data. The simplest models can use only simple features like the numerical features associated with a tweet (like the number of retweets, for example) since they are the easiest to feed into a model. Features that are not numerical must be preprocessed to be converted into numerical features because machine learning and deep learning models can only work with numbers. In the case of tweets, the text of the tweets is obviously a relevant feature that brings most of the information necessary to make good predictions. Therefore, the text must be cleaned and preprocessed with Natural Language Processing (NLP) tools or algorithms to be converted to a meaningful set of numerical features. More complex features could be used, such as the answers to a tweet, the previous behavior, or the social connections of its author. Such features are not taken into account in this project. It is important to note that the approach chosen here relies only on classifying the tweets based on the features they contain only. The models do not attempt to interpret the claims in the tweets to compare them to existing sources of information[13].

To find the best model I could train for this project, I experimented with several algorithms and different features. I first tried to train models using only the numerical features that are associated with tweets. Then, I dug into Natural Language Processing-oriented approaches that focus on the content of the tweet.

[13] Such an approach would be called *knowledge-based fake news detection*, and would rely on the construction of a knowledge base.

Category	Count	Binary category
Irrelevant	131	None
Conspiracy	924	FAKE
True Treatment	0	REAL
True Prevention	175	REAL
Fake Cure	141	FAKE
Fake Treatment	34	FAKE
False Fact or Prevention	321	FAKE
Correction or Calling out	1,331	None
Sarcasm or Satire	476	None
True Public Health Response	163	REAL
False Public Health Response	3	FAKE
Politics	512	None
Ambiguous or Difficult to Classify	143	None
Commercial Activity or Promotion	37	None
Emergency Response	17	None
News	95	REAL
Panic Buying	70	None

Table 1. Categories of the CMU-MisCOV19 dataset annotations and their associated tweet count and binary class.

4.3.1. Presentation of the datasets used

Before describing the experiments I conducted, I must describe the datasets I used to train and test the models. I used the public dataset that contains tweets related to COVID–19, with annotations.

CMU-MisCOV19

The CMU-MisCOV19 dataset [6] contains 4,573 manually annotated tweets from 3,629 distinct users. The tweets were collected on March 29, 2020, June 15, 2020, and June 24, 2020. Due to how the Twitter Search API works, the tweets were posted up to seven days before each date. The tweets were collected using a set of keywords and hashtags related to the pandemic, such as *#COVID19* or *doctor*, but also to the popular fake news claims of the time, such as *bleach*, *5G* or *bioweapon*. The annotation classifies each tweet into one of 17 possible classes. Since I only wanted to make a binary classification, I mapped each of these 17 categories into the *REAL* or *FAKE* categories. Some categories did not fit in this binary classification, so I discarded them in this experiment. Although the *Correction or Calling out* category would fit in the *REAL* class, I discarded it since it was over-represented in this dataset. The list of the 17 categories with their tweet count and the associated binary class is shown in Table 1.

Category	Count	Binary category
Note severe	1,851	FAKE
Possibly severe	439	FAKE
Highly severe	568	FAKE
Refutes/Rebuts	447	REAL
Real News/Claims	57,981	REAL

Table 2. Categories of the Covid-HeRA dataset annotations and their associated tweet count and binary class.

Covid-HeRA

The Covid-HeRA dataset (for Health Risk Assessment) [3] contains 61,286 tweets. The tweets are taken from the CoAID dataset [2]. The CoAID dataset contains news article about COVID–19 annotated with binary labels. It also contains tweets that are associated with each news article without labeling these tweets. The HeRA dataset annotates these tweets between five categories to get a finer scale than a simple binary classification. Alongside with *Real News/Claims* and *Refutes/Rebuts* that corresponds to users correcting false claims, the fake news is split into three categories based on their estimated severity. This severity reflects how harmful the claim can be for its audience by assessing the risk of dangerous behavioral change (for example, a tweet falsely claiming that a celebrity tested positive to COVID–19 is considered not severe. In contrast, a tweet claiming that bleach can cure COVID–19 is considered highly severe because it promotes a behavior with health risk). Once again, I mapped these categories to the binary labels *REAL* and *FAKE*. Table 2 shows the categories of the HeRA dataset with their tweet count and associated binary labels. Since the binary classes are highly unbalanced, I only used a fraction of these tweets in my training process.

A mix of these two datasets was used to train the final model used in the prediction engine. It must be noted that other datasets were used in the experiments I conducted, but they were either not focused on COVID–19 , or the annotations were imperfect. Therefore, I present here only the datasets that were used to train the production model.

4.3.2. Models using numerical features

The first experiments I conducted at the beginning of this internship consisted of making predictions using the numerical features of tweets. They are the easiest to work with since they don't require any particular preprocessing used by a machine learning model. The numerical features that I used are:

- Number of retweets
- Number of likes
- Number of people followed by the author of the tweet
- Number of followers the author has

Algorithm	AUC
Random forest	0.61
Linear SVC	0.52
Logistic regression	0.53

Table 3. Results of the models trained on numeric features.

- The total number of tweets written by the author
- Followers/following ratio of the author

The five first features are available through the Twitter API, and the author's follower/following ratio is computed from the number of followers and people followed[14]. I decided to include this ratio and the raw features because it is an excellent indicator of a Twitter account's popularity and behavior. Popular or famous people tend to have much more followers than subscriptions. On the contrary, some aggressive accounts subscribe to a lot of accounts to try to gain popularity. This ratio might therefore be an exciting feature to determine if a tweet contains fake news.

I selected three machine learning algorithms that I trained using these features. These algorithms are as follows:

- Random forest
- Linear SVC
- Logistic regression

I chose these algorithms because they were easy to use with the skills I had at the beginning of this internship and are popular machine learning algorithms that all have their advantages and disadvantages. However, in hindsight, the linear SVC is probably a bad choice since we have no reason to suppose that our data is linearly separable.

The Area Under ROC[15] was the metric computed for each model. The results are shown in Table 3. It is important to note that this experiment was done on a different dataset than those presented above, the MM-COVID dataset [4], but this dataset was not kept for the following experiments and the training of the production model because its annotations are imperfect. Since 0.5 would be the score of a random classifier, we can see that none of these models can predict the class a tweet belongs to correctly.

[14] Computing new features as a combination or reordering of existing ones is called *feature engineering*.

[15] The area under the ROC curve, or area under the curve (AUC), is a metric to evaluate the performance of a binary classifier. An AUC of 0.5 means that the classifier cannot separate the classes, while an AUC of 1 means the classifier can perfectly separate the two classes. Compared to other metrics such as accuracy or F1-score, the AUC has the advantage of being insensitive to the potential lack of balance between the two classes.

4.3.3. Models using textual data

To get more reliable predictions, I then trained models using the textual contents of tweets. To be inputted into a machine learning algorithm, a piece of text must be converted to numerical features by a Natural Language Processing technique. Several techniques exist:

- **Bag-of-words** creates one feature per word present in the corpus of samples and takes the number of occurrences of the word in each sample as the value of the features. In the case of short texts like tweets, the feature vector of a sample is very sparse because each tweet contains only a few words. Alternatively, features can be created for the n-grams, i.e., the sequences of n adjacent words. Both approaches can also be combined.
- **TF-IDF**, short for "term frequency-inverse document frequency," is similar to bag-of-words. However, instead of taking the raw number of occurrences of a word, it takes its frequency in the sample divided by its frequency in the whole corpus. This aims to reduce the score of words that are very common in the corpus but give little information about the sample, such as *the* or *in*[16], or, in this case, *covid*.
- **Word2Vec** uses a pre-trained neural network to transform a word into a fixed-length vector that represents this word (this is called *word embedding*). This vector represents the meaning of the word, and words that have similar purposes are close in the vector space. To get the Word2Vec representation of a sample of text, the Word2Vec representations of its words are simply averaged.
- **fastText** is another word embedding model created by Facebook AI Research lab.

Once the texts are converted into numerical feature vectors, they can be inputted in any machine learning or deep learning algorithm. Choosing the right machine learning technique for a task of text classification, therefore, involves selecting the suitable model *and* the proper embedding technique.

To get the best model for the prediction engine, I performed a benchmark between five associations of embedding and algorithms:

- **BoW + FFNN** A bag-of-words vectorization associated with a simple feed-forward neural network[17].
- **Bi-directional LSTM** is a bi-directional Long Short-Term Memory (LSTM) network, which is popular in NLP. A word embedding is performed before the network[18].
- **Word2Vec + FFNN** is the Word2Vec model used for word embeddings associated with the same feed-forward neural network than in the "Bow + FFN" approach.

[16] It must be noted that these words (called *stop words*) are generally excluded from the text during a preprocessing step (so before the embedding) since they are not informative.

[17] This feed-forward neural network's layers are detailed on Figure 7 in the Appendix section.

[18] The word embedding algorithm used is the one built in the Keras library.

Algorithm	PHEME 9 T/F	PH.5Lc bin	PH.5Lc multi	Gossipcop	Average rank
BoW + FFNN	91.7 ±1.2	77.7	20.3	84.8 ±1.8	2.75
Bi-directional LSTM	91.1 ±0.4	76.6	29.4	84.9 ±1.1	**2.5**
Word2Vec + FFNN	90.9 ±1.0	71.4	23.3	85.2 ±1.6	3.25
BERT	87.5 ±1.1	77.5	27.4	56.1 ±1.1	**2.5**
fastText	87.2 ±1.3	60.2	39.8	82.7 ±0.8	4

Table 4. Results of the benchmark of NLP models. The metric is F1-score.

- **BERT** is a language model developed by Google. It includes all stages of text processing.
- **fastText** is simply the fastText language model that once again takes care of all the steps.

I benchmarked these five approaches on two datasets not related to COVID–19: The **PHEME** dataset and a reduced version of the **Gossipcop** dataset. The PHEME dataset contains tweets related to five breaking news events in the late 2010s, including the Paris terror attacks in January 2015, and the Gossipcop dataset contains tweets about celebrities. The PHEME dataset was split in three different ways to get the same datasets as in Pelrine, Danovitch, and Rabbany's article [8]:

- **PHEME9 T/F** contains only the tweets labeled as *true* or *false* (the PHEME dataset also contains tweets labeled as *unverified*). A 70-10-20 train-dev-test split is done.
- **PHEME5 Lc bin** is the dataset using only *true* and *false* labels split by events (training on four events and testing on the last event, the Charlie Hebdo terror attack).
- **PHEME5 Lc multi** is the same split as PHEME5 Lc bin but also using the *unverified* label.

The results of this benchmark are presented in Table 4. The column "Average rank" shows the average of the ranks of the model against all datasets. Cross-validation was performed on the PHEME 9 T/F and Gossipcop datasets. The standard deviation of the F1 scores is therefore reported. The PHEME 5 Lc dataset has only one split, and consequently, no cross-validation is done. We notice that the performance is much lower on the PHEME 5 Lc dataset, especially in the three-class classification. This is logical: as the model was tested on tweets related to an event that no tweet from the training set is related to, the testing data has notable differences with the training data. This is interesting because it is a similar situation that our system will encounter in production. We can see that except fastText, the models don't perform too poorly on this dataset. We can see that the bidirectional LSTM and the BERT models perform this task the most by looking at the rank. I, therefore, chose the bidirectional LSTM model for the incremental learning experiments in section 4.4.

4.4. Incremental learning

By nature, a machine learning model takes its knowledge from the data it has been trained on. In the case of text classification, the model's prediction is linked to how similar the submitted text sample is to the training samples of each class. This similarity can be of various types: for example, a word or sequence of the word can appear with a high frequency in a particular class, or the wording and tone of the text can be different among classes (for example, tweets from news outlets typically use a distinctive wording). In the case of fake news detection, the classification of a text sample can only be reliable if other samples regarding the same claim or subject were present in the training data. Otherwise, since the models we have here don't understand the meaning of the sentences and their relations to the news context, the model is likely to make its prediction based on the knowledge given by text samples that contain the exact words or latent features but are related to a different claim. Moreover, the "news" being ever-changing by definition, what is true one day may be incorrect the following week.

To properly detect fake news with machine learning, the model must be updated constantly to take into account the most recent knowledge, without forgetting the previous data, as new tweets containing old fake news can still appear from time to time. There are two possible approaches to update a model:

1. Train the model from scratch using a training dataset that contains both old and new data. Each time the model is trained, the training set is bigger, and the training process is longer.
2. Train the model only with the new data, but start the training process with the weights[19] of the existing model. Each step of the training process is made only on the newest data and is therefore relatively short. However, depending on the nature of the model and its hyperparameters[20], knowledge from earlier data may be gradually forgotten.

The first approach has the advantage that since all data is given the same weight, knowledge from older data does not risk being erased. However, as the training set size grows, so does the training time. To detect fake news as efficiently as possible, we need to update the model as often as possible, preferably every 24 hours. Therefore, the training process must remain fast enough and never take more than a few hours, even as the training set gets very large.

Since we operate on a cluster of a machine, we can imagine training the model in a distributed way to reduce the training time and repeat the whole training process every day. Several libraries allow performing incremental machine learning and deep learning in a distributed way on Apache Spark. However, this approach has two drawbacks:

- These libraries are add-ons to Apache Spark. As they are experimental and not widely use, they are difficult to integrate with the cluster installation.

[19] The weights of a model are the values of the parameters it contains. These values are found during the training process and are the representation of the "knowledge" of the model.

[20] The hyperparameters are the global parameters that can be set to a model to tune it, like the learning rate or the number of units in a neural network. They are called "hyperparameters" to distinguish them from the trainable parameters or weights.

- The cluster already carries the task to make predictions on incoming tweets in near real-time and is likely to be sized accordingly. Adding the computational charge of the training process a few hours a day could slow down the prediction process and add latency in the prediction, which is not acceptable.

4.4.1. Experiment design

Because of these difficulties, it was interesting to see how an incremental training approach could be beneficial[21]. To really measure the differences in predictive power and training time between this approach and the *from-scratch* approach, I benchmarked these two methods. The goal of the experiment was to show that:

1. The incremental training approach learned from the most recent data without forgetting too much earlier knowledge.
2. The training time gain using the incremental approach was significant.

To prove the first point, I needed to train the model with data that evolved over time. To simulate such evolving data, I used two datasets related to COVID–19: The HeRA dataset [3] and a reduced version of the CMU-MisCOV19 dataset [6]. Although these datasets both contain tweets about COVID–19, they have not been collected at the same dates nor with the same methods and therefore contain tweets with different claims and themes. The word clouds of these datasets (Figure 3) can show that although the main theme of both dataset is COVID–19, with words such as *COVID*, *COVID19*, and *coronavirus*, they also have specific words or sequences of words such as *flu* or *Contains HIV* for the HeRA dataset and *bioweapon*, *cure* or *immune system* for the CMU-MisCOV19 dataset.

To simulate evolving data, I created several batches of training samples, alternating between the two datasets. The model would then be trained successively on each batch[22] without resetting the weights. The composition of the batches is shown in Figure 4.

The two training processes are done separately. First, the incremental approach, where the model is trained on each batch without resetting the weights between the batches. Then, the classic approach (or *from-scratch* approach) where the model is trained at each step *i* on the cumulative union of the batches from *0* to *i* as shown on Figure 5. These two training processes are repeated five times, with samples ordered differently from the same source. However, the division of the samples between the training batches and testing sets remains the same in the five trials.

[21] Even if this incremental approach was not selected in the end.

[22] These batches must not be confused with the batches (or "mini-batches") used during the training process with mini-batch gradient descent. The batches we are talking about here are the equivalent of the entire dataset in the classic training process. Still, this process is successful on these datasets that we call "batches" here for convenience. The mini-batches are also used in this experiment, but they are handled automatically by the deep learning library, TensorFlow.

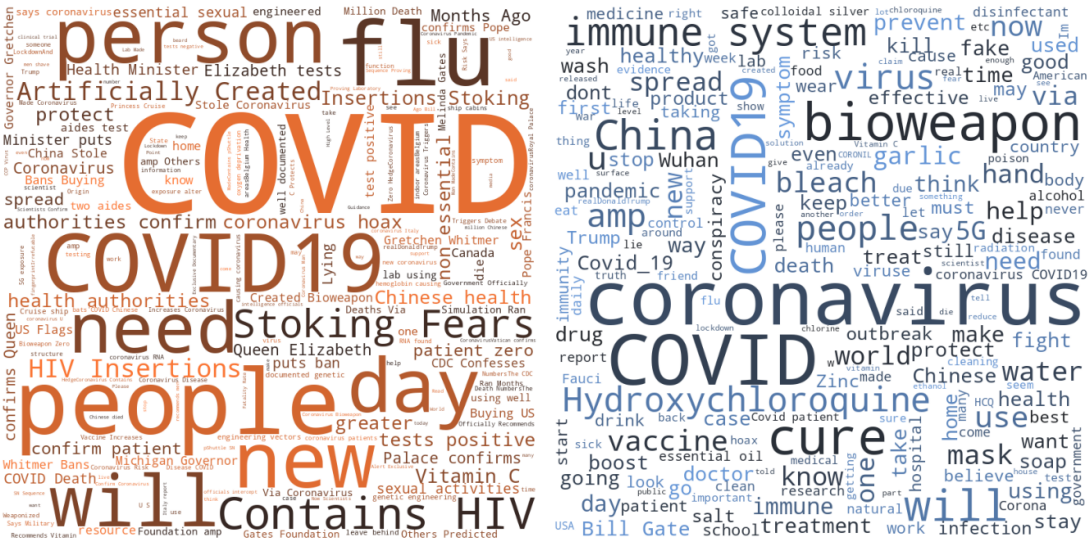


Figure 3. Most common words in the HeRA dataset (left) and the selected subset of the CMU-MisCOV19 dataset (right).

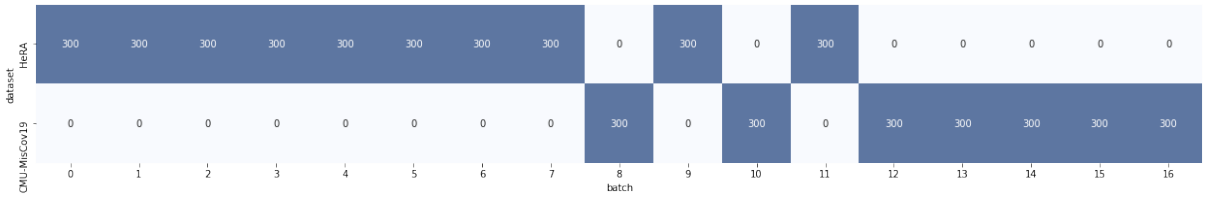


Figure 4. Distribution of the source datasets among the training batches.

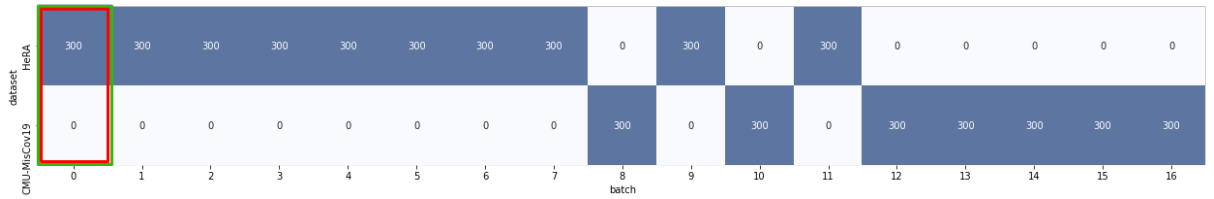
4.4.2. Results

After each training step, the model is evaluated on three test sets: One that contains samples from the HeRA dataset, and one that includes samples from the CMU-MisCOV19 dataset, and the union of the two. The testing samples are naturally not present in any of the training batches. The metric used for the evaluation is the F1-score[23]. The results after each training step are shown in Figure 6 (The results at step 0 are the results before any training of the model, they are included for comparison). The training time in the incremental approach was roughly the same as in the *from-scratch* approach.

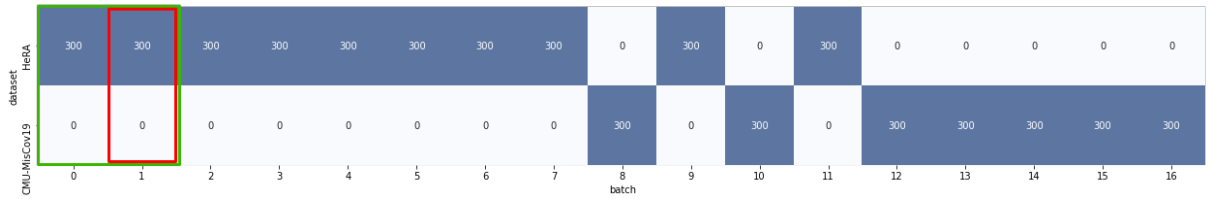
In these results, we can clearly see that the model quickly becomes very good at making a prediction on the HeRA test dataset but performs poorly on the CMU-MisCOV19 test dataset. However, starting from step 9, when the model encounters training data from the CMU-MisCOV19 dataset, its predictions on the related test dataset become better. However, in the incremental approach, its prediction on the HeRA test dataset becomes slightly worse.

More generally, we can see that the incremental approach is very sensitive to changes like the

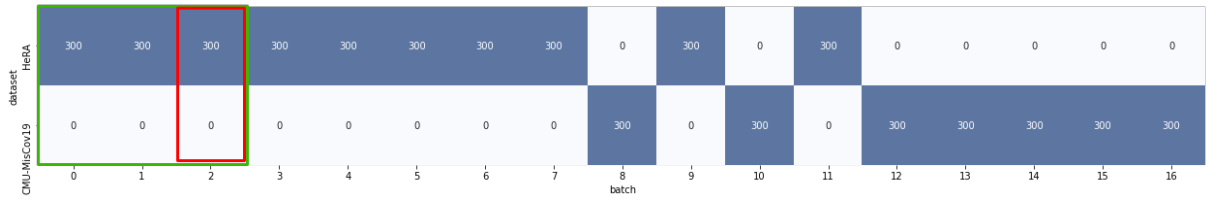
[23] The F1-score is a metric to evaluate a binary classifier. It combines the precision (the fraction of true positives among the total number of predicted positives) and the recall (the number of true positives among the total number of actual positive elements).



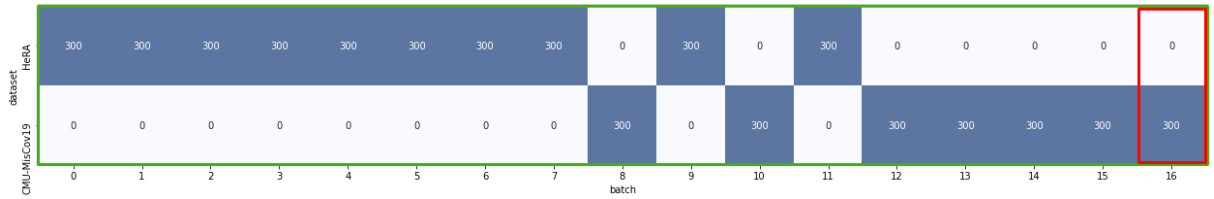
(a) Step 1



(b) Step 2



(c) Step 3



(d) Step 17

Figure 5. Training sets at different steps. The composition of sets of the incremental approach is in red and the composition of the from-scratch *approach in green.*

training data and that the model prioritizes knowledge from recently seen data. The big spikes of performance in the results of the incremental approach can support this, as they are not present in the results of the classic approach. At the end of the training process, the model trained incrementally is better at making predictions on the recent data that the model trained at once but worse on the older data. It is interesting to note that the performance on the union of the two test sets is virtually the same for the two training approaches.

It might seem surprising that despite these variations of performance that are correlated with the contents of the batches, the performance on the CMU-MisCOV19 test dataset is always lower than the performance on the HeRA test dataset. But the total number of training batches from the CMU-MisCOV19 dataset is lower (because the source dataset is smaller). Also, experiments on these two datasets separately show that most models are better at making predictions on the HeRA dataset. Therefore, we can assume that the classes of this dataset are naturally easier to separate than the classes on the CMU-MisCOV19 dataset. It must also be remembered that the binary classes in the

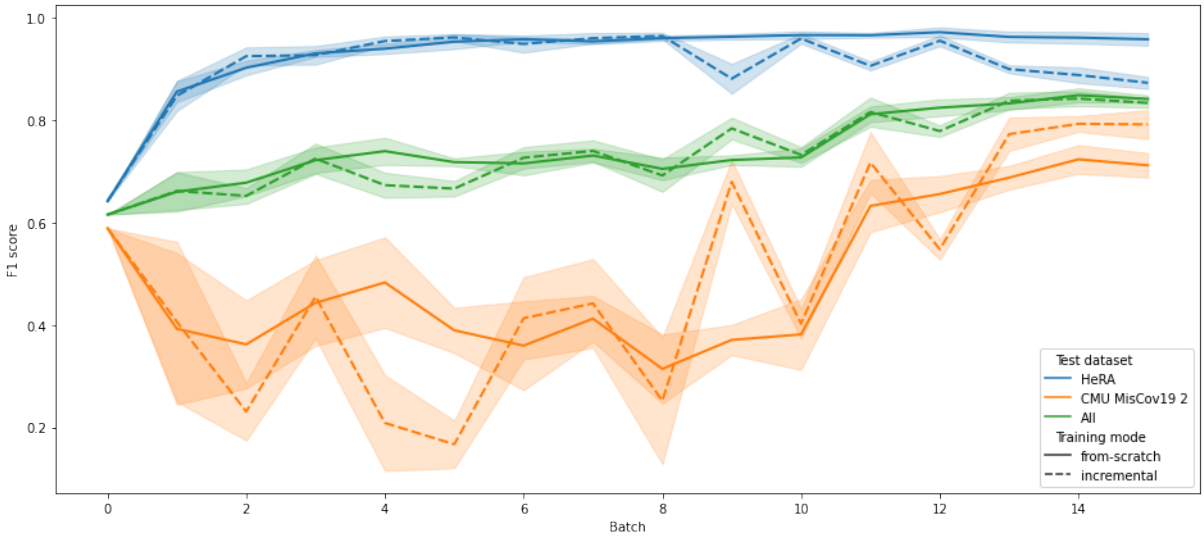


Figure 6. Comparison of the results of the incremental and from-scratch *approaches at each step. The score before any training is included. The error bands show the standard deviation over the five trials.*

CMU-MisCOV19 dataset result from several classes that have been merged, which is not the case in the HeRA dataset.

5. Conclusion

During the sixteen weeks of this internship, I had the opportunity to develop my knowledge and skills in data science. In particular, I was able to see the difference in workflow between this discipline and typical software development: the former is much more made of trial and error, as building a machine learning model requires testing a lot of parameters to see what gives the best performance. This is also one of the challenges in this field: I realized that it is essential to keep track of the experiments made in a very structured way, or we risk getting lost in our experiments and lose time redoing things that have already been made. In a software development project, I'm used to using version control to properly manage projects, and I realized that in the same way, it is crucial to use experiment tracking tools when doing a data science project. Although such tools exist, I didn't have to use any in this internship, probably because they are much less ubiquitous than version control tools in software development.

It was also an opportunity to discover the field of data engineering/big data. Indeed, while I had a basic overview of the field of machine learning before this internship, the technologies behind big data and large-scale data processing were utterly unknown to me. This internship allowed me to better understand this area and develop skills in some of its core technologies. I found myself very interested in this aspect because it has a good balance between software engineering, on which my curriculum at ENSICAEN was focused until now, and data science. I like being able to have a software engineering-

like structured workflow while working on an interesting data-oriented project. In the future, I would like to dive deeper into this field.

This internship was also an opportunity to discover the world of academic research. Interacting with researchers, post-doctoral students, and Ph.D. students every day and reading several research articles to conduct the project gave me some insight into the process of doing research and publishing papers.

Finally, this internship was also an opportunity to travel and discover the country of Estonia and to improve my English language skills by practicing the language both in a professional environment and in my daily life, which are two circumstances that allow developing very different aspects of language mastery that are hard to learn in academic language courses. I was also able to live in a foreign country for several months, which was a new and enriching experience for me.

List of Figures

1 General architecture of the system 9
2 Kibana dashboard showing various statistics about the tweets and predictions, in near real-time. 13
3 Most common words in the HeRA dataset and the selected subset of the CMU-MisCOV19 dataset. 21
4 Distribution of the source datasets among the training batches. 21
5 Training sets at different steps. 22
6 Comparison of the results of the incremental and *from-scratch* approaches. 23
7 The feed-forward neural network used in the benchmark. 29

List of Tables

1 Categories of the CMU-MisCOV19 dataset annotations and their associated tweet count and binary class. 14
2 Categories of the Covid-HeRA dataset annotations and their associated tweet count and binary class. 15
3 Results of the models trained on numeric features. 16
4 Results of the benchmark of NLP models. The metric is F1-score. 18

List of Code Listings

1 Sample of the filtered stream API response 11

Appendix

A Details about machine learning models

Layer (type)	Output Shape	Param #
batch_normalization (BatchNo	(None, 2460)	9840
dense (Dense)	(None, 256)	630016
batch_normalization_1 (Batch	(None, 256)	1024
dropout (Dropout)	(None, 256)	0
dense_1 (Dense)	(None, 256)	65792
batch_normalization_2 (Batch	(None, 256)	1024
dropout_1 (Dropout)	(None, 256)	0
dense_2 (Dense)	(None, 1)	257

Total params: 707,953
Trainable params: 702,009
Non-trainable params: 5,944

Figure 7. The feed-forward neural network used in the benchmark.

BIBLIOGRAPHY

[1] Alessandro Bondielli and Francesco Marcelloni. "A survey on fake news and rumour detection techniques". In: *Information Sciences* 497 (2019), pp. 38–55. ISSN: 0020-0255. DOI: 10.1016/j.ins.2019.05.035. URL: https://www.sciencedirect.com/science/article/pii/S0020025519304372.

[2] Limeng Cui and Dongwon Lee. *CoAID: COVID-19 Healthcare Misinformation Dataset*. 2020. arXiv: 2006.00885 [cs.SI].

[3] Arkin Dharawat et al. *Drink bleach or do what now? Covid-HeRA: A dataset for risk-informed health decision making in the presence of COVID19 misinformation*. 2020. arXiv: 2010.08743 [cs.CL].

[4] Yichuan Li et al. *MM-COVID: A Multilingual and Multimodal Data Repository for Combating COVID-19 Disinformation*. 2020. arXiv: 2011.04088 [cs.SI].

[5] Sahill Loomba et al. "Measuring the impact of COVID-19 vaccine misinformation on vaccination intent in the UK and USA". In: *Nature Human Behaviour* 5 (3 2021), pp. 337–348. ISSN: 2397-3374. DOI: 10.1038/s41562-021-01056-1. URL: https://doi.org/10.1038/s41562-021-01056-1.

[6] Shahan Ali Memon and Kathleen M. Carley. "Characterizing COVID-19 Misinformation Communities Using a Novel Twitter Dataset". In: *Proceedings of The 5th International Workshop on Mining Actionable Insights from Social Networks (MAISoN 2020), co-located with CIKM, virtual event due to COVID-19* (2020). arXiv: 2008.00791.

[7] Scott Neuman. "Man Dies, Woman Hospitalized After Taking Form Of Chloroquine To Prevent COVID-19". In: *NPR* (Mar. 2020). URL: https://www.npr.org/sections/coronavirus-live-updates/2020/03/24/820512107/man-dies-woman-hospitalized-after-taking-form-of-chloroquine-to-prevent-covid-19?t=1628604706299.

[8] Kellin Pelrine, Jacob Danovitch, and Reihaneh Rabbany. "The Surprising Performance of Simple Baselines for Misinformation Detection". In: *Proceedings of the Web Conference 2021*. WWW '21. Ljubljana, Slovenia: Association for Computing Machinery, 2021, pp. 3432–3441. ISBN: 9781450383127. DOI: 10.1145/3442381.3450111. URL: https://doi.org/10.1145/3442381.3450111.

[9] Kai Shu et al. "DEFEND: Explainable Fake News Detection". In: *Proceedings of the 25th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining*. KDD '19. Anchorage, AK, USA: Association for Computing Machinery, 2019, pp. 395–405. ISBN: 9781450362016. DOI: 10.1145/3292500.3330935. URL: https://doi.org/10.1145/3292500.3330935.

[10] *Using AI to detect COVID-19 misinformation and exploitative content*. May 2020. URL: https://ai.facebook.com/blog/using-ai-to-detect-covid-19-misinformation-and-exploitative-content.

[11] Xinyi Zhou and Reza Zafarani. “A Survey of Fake News: Fundamental Theories, Detection Methods, and Opportunities”. In: *ACM Comput. Surv.* 53.5 (Sept. 2020). ISSN: 0360-0300. DOI: 10.1145/3395046. URL: https://doi.org/10.1145/3395046.

Summary

This report details my internship at the Tallinn University of Technology during the summer of 2021. The goal of this internship was to build a system that aims to detect COVID–19-related misinformation on Twitter in near real-time. Tweets related to COVID–19 are streamed from Twitter and fed into a machine-learning model that performs a binary classification. Statistics about the resulting predictions are displayed on a near real-time dashboard. As it is implemented in a distributed way on a cluster, this system is easily scalable. This report details the architecture of the system, as well as the development of the machine learning model, the core piece of the prediction engine.

Keywords: Machine learning, deep learning, big data, near real-time, COVID–19.

Résumé

Ce rapport présente mon stage à l'Université de Technologie de Tallinn durant l'été 2021. Le but de ce stage était de développer un système de détection de désinformation au sujet du COVID–19 sur Twitter en quasi temps réel. Le flux de tweet est téléchargé depuis Twitter et envoyé dans un modèle d'apprentissage machine qui effectue une classification binaire. Des statistiques sur prédictions qui en résultent sont affichées sur un tableau de bord en quasi temps réel. Ce système étant implémenté de manière distribuée sur une grappe de machines, il passe facilement à l'échelle. Ce rapport détaille l'architecture du système, ainsi que le développement du modèle de prédiction, qui constitue la pièce maîtresse du module décisionnel.

Mots-clés: Apprentissage machine, apprentissage profond, *big data*, quasi temps réel, COVID–19.

ENSI CAEN

ÉCOLE PUBLIQUE D'INGÉNIEURS
CENTRE DE RECHERCHE

École Publique d'Ingénieurs en 3 ans

6 boulevard Maréchal Juin, CS 4505
14050 CAEN cedex 04

ministère éducation nationale enseignement supérieur recherche
Liberté • Égalité • Fraternité
RÉPUBLIQUE FRANÇAISE

Normandie Université

Cti
Commission des titres d'ingénieur

cdefi
Conférence des Directeurs des Écoles Françaises d'Ingénieurs

European Accreditation of Engineering Programmes
EUR-ACE®

CONFÉRENCE DES GRANDES ÉCOLES