

École polytechnique de Louvain

Robustness of Fake News Detectors

Author: **Quentin CARBONNELLE**

Supervisor: **François-Xavier STANDAERT**

Readers: **Antonin DESCAMPE, Michel VERLEYSEN**

Academic year 2019–2020

Master [120] in Mathematical Engineering

Abstract

The goal of our study is to evaluate the robustness of different fake news detectors, i.e., to determine how difficult it is to slightly modify fake articles so that the detector label them as reliable. For this, we first built a database of articles from both reliable and non-reliable media outlet. On that database, we constructed 5 fake news classifiers using well-known machine learning techniques.

We studied the robustness of two of them, chosen for their high performance and for having inherently different classification strategies: logistic regression combined with tf-idf ignores the order of the words, while Long Short-Term Memory neural network (LSTM) with word embedding doesn't. We evaluated robustness under 3 scenarios of attack: 1) low semantic and syntactic constraints with black box knowledge and full automation, 2) strong semantic and syntactic constraints with human intervention and white box knowledge, and 3) the latter with black box knowledge.

We characterize robustness by the proportion of fake articles that we can successfully change so that they become labeled as reliable. The logistic regression classifier is robust: the proportion is less than 32 percent, under the 3 scenarios of attack. The LSTM classifier is robust to the low constraint attack without human intervention (22 percent), but less so when human intervention is allowed: 3 out of 5 articles were successfully transformed into adversarial articles. We believe however that this last result could be improved with further study.

Finally, as part of our robustness study, we evaluated transferability, i.e., how the adversarial articles created for these two classifiers are classified by the other three classifiers we created at the begin of our study. The naive bayes classifier was the most robust: all of the adversarial articles were still classified as fake.

We conclude that fake news detectors are useful to prevent the spread of fake news.

Acknowledgements

I thank my father for the efficiency and enthusiasm he put into helping me when it was necessary.

I thank my mother for the unexpected milkshakes and other *chouchoutage* during moments of intense focus.

I thank them even more because this intellectual and psychological help was not only during the months of work on this project but during the last five years of study.

I thank my promoter, François-Xavier Standaerd, who took the time to advise me and pointed me in the right direction.

I thank Alice and Simon for animating the house, even if only once a month, and for motivating me when I needed it.

I also thank Papaya, thanks to whom I was able to make a necessary escape for a few weeks.

Contents

I	Motivation and context of the study	1
II	Background	2
1	Database	2
1.1	t-SNE vizualization	2
2	Natural language processing and classification	3
2.1	Preprocessing	3
2.1.1	Cleaning	3
2.1.2	Stemming	4
2.2	Vectorization	4
2.2.1	Bag of words	4
2.2.2	Tf-idf	4
2.2.3	Word2Vec	5
2.3	Classifiers	6
2.3.1	The binary cross-entropy loss function	6
2.3.2	Logistic Regression	6
2.3.3	Bernouilli Naive Bayes	7
2.3.4	Random Forest	8
2.3.5	Bidirectional LSTM	8
3	Evaluation Metrics	10
3.1	Accuracy	10
3.2	Area Under ROC Curve (AUC)	10
3.3	F1 score	10
4	Adversarial Example	11
4.1	Adversarial constraints	11
4.2	Adversarial capabilities	13
III	Database	14
1	Building the database	14
2	Database statistics	15
2.1	Sources exploration	15
2.2	Date exploration	16
2.3	Topic exploration	16
2.4	Visualization with t-SNE	17
3	Conclusion	20

IV	Classifiers	21
1	Implementation	21
1.1	Preprocessing	21
1.2	Vectorization	21
1.2.1	Tf-idf	21
1.2.2	Word2Vec	22
1.3	Classification	22
1.3.1	Traditional methods	22
1.3.2	LSTM	23
1.4	Fine-tuning	23
2	Results	24
3	Conclusion	26
V	Adversarial Example	27
1	Soft constraints with high production rate	28
1.1	Approach	29
1.2	Example	30
1.3	Validation	31
1.4	Conclusion	31
2	Strong constraints with low production rate - White Box	32
2.1	Approach	32
2.1.1	Logistic Regression	33
2.1.2	LSTM	34
2.2	Results and Example	35
2.2.1	Logistic Regression	35
2.2.2	LSTM	36
2.3	Validation	37
2.4	Conclusion	38
3	Strong constraints with low production rate - Black Box	38
3.1	Approach	39
3.2	Results and example	39
3.2.1	Logistic Regression	39
3.2.2	LSTM	40
3.3	Validation	41
3.4	Conclusion	42
4	Transferability	42
5	Further research	44

VI Conclusion	45
References	47
A Parameter Tuning	50
A.1 Logistic regression	50
A.2 Naive Bayes	52
A.3 Random Forest	54
A.4 LSTM	55
B Reliable Hot words	57
B.1 Black Box - Logistic regression	57
B.2 Black Box - LSTM	58
B.3 White Box Logistic regression	59
B.4 White box - LSTM	61

Part I

Motivation and context of the study

We've all heard about fake news. We see one at least once a week but it doesn't bother us and we just go over it. Yet, in some particular cases, fake news can have drastic effects. Extreme examples are the recurrent reports of mob violence generated by false stories circulating on WhatsApp in India [1]. But while it can affect one's life directly it can also have impact on a larger scale, e.g. the impact of fake news in the 2016 US election is often discussed [2][3]. Indeed, the fake news item that claimed that Pope Francis endorsed Donald Trump was the most shared article of all the news - true or false - published during the US election [4]. Hence, the threat of fake news over democratic countries [5] may not to be underestimated: Fake news is not just about journalism, it is putting democracy at risk.

As it is formally defined: a fake news is a fabricated news that takes the appearance of real news with the aim of deceiving people [4]. It has gained a lot of attention the past few years but it has existed for a long time [6]. Indeed, it is an effective tool for propaganda and leaders have taken advantage of that for centuries. However, until recently, fake news was more or less centralized, but with the rise of telecommunication and even more with the invention of the internet and social media, fake news in its modern form can be emitted and propagated by nearly everyone. As a result, never was an individual as exposed to fake news as today.

Fake news circulates freely and very few measures are taken to moderate them. Here are some notable initiatives. In Finland, extra efforts are made in classrooms to raise awareness to misinformation and its spread [7]. In France, a law was passed which aims to empower judges to order the immediate removal of "fake news" during election campaigns [8]. Recently, moderation in the digital world has become subject of research and improvement. Facebook's news feed algorithm is regularly changed to fight the spread of misinformation [9] and Twitter puts warning messages on tweets containing misleading information about Covid-19 [10].

Various strategies are possible for designing an automated fake news detector. Three main approaches stand out: the network approach relies on the study of the propagation of the fake news, a strategy based on fact-checking verifies the news by querying existing knowledge networks, and finally the linguistic approach spots fake articles by observing language patterns[11]. Indeed, a fake news is often designed to be sensational and to incite sharing. Machine learning algorithms and natural language processing can be used together to take advantage of this characteristic and achieve impressive result in fake news detection. Spam detection is based on the same principle [12].

In this document, we build fake news classifiers and then evaluate their robustness against fake news crafted precisely to trick the classifiers (named adversarial examples). Apart from the introduction and conclusion, this document is structured in 4 chapters. Chapter two discusses the theoretical grounds for understanding the work. In chapter three, we discuss the construction and exploration of the database. In chapter four, the classifiers are built and fine-tuned and we compare their performance. In chapter five we study how the classifiers work and try to trick them making fake articles. We then briefly analyze the transferability of the adversarial articles across different classifiers.

Part II

Background

This chapter serves to explain or remind some well-known methods used in this document. Further details can be found in the references.

1 Database

Machine learning algorithms need by definition samples to learn from. Those are collected into a database. In our context, the database will contain articles labelled as fake or as reliable.

1.1 t-SNE vizualization

To analyze a database, it is often interesting to visualize the relation between elements. A common visualization technique is the t-SNE, it takes high-dimensional vectors as input. In the context of natural language, those vectors are obtained after vectorization of the texts, which is explained in section 2.2.

T-distributed Stochastic Neighbor Embedding (t-SNE), introduced in [13], is a nonlinear dimensionality reduction technique well-suited for the visualization of high-dimensional datasets. The t-SNE algorithm has two main steps.

1. t-SNE constructs a probability distribution over pairs of high-dimensional data points such that a pairs probability is proportional to its similarity.
2. t-SNE defines a similar probability distribution over the points in the low-dimensional map

Let's look at it in more details: We have N data points $\mathbf{x}_i$ of high dimension that we want to visualize. **The first step** of t-SNE is to compute probabilities p_{ij} that represent how similar $\mathbf{x}_i$ is to $\mathbf{x}_j$. We start by defining $p_{j|i}$ for $i \neq j$:

$$p_{j|i} = \frac{\exp\left(\frac{-\|\mathbf{x}_i - \mathbf{x}_j\|^2}{2\sigma_i^2}\right)}{\sum_{k \neq i} \exp\left(\frac{-\|\mathbf{x}_i - \mathbf{x}_k\|^2}{2\sigma_i^2}\right)}$$

We set $p_{i|i} = 0$ and we note that the attribute of a probability $\sum_j p_{j|i} = 1$ is verified for all i . Since it uses the euclidian distance $\|\mathbf{x}_i - \mathbf{x}_j\|$, it is affected by the curse of dimensionality. Applying t-SNE on vectors with high dimensionality may have disappointing results.

We can now define the probability p_{ij} as:

$$p_{ij} = \frac{p_{j|i} + p_{i|j}}{2N}$$

The second step is to define a similar probability distribution but in 2-dimension. We thus want to find the N points $\mathbf{y}_i \in \mathbb{R}^2$ that reflects the similarities p_{ij} as well as possible. We compute the similarity between two points $\mathbf{y}_i$ and $\mathbf{y}_j$ with q_{ij} defined as:

$$q_{ij} = \frac{\left(1 + \|\mathbf{y}_i - \mathbf{y}_j\|^2\right)^{-1}}{\sum_{k \neq i} \left(1 + \|\mathbf{y}_i - \mathbf{y}_k\|^2\right)^{-1}}$$

Again with $q_{ii} = 0$.

To measure the similarity between the two probability distributions P (with p_{ij}) and Q (with q_{ij}), we use the Kullback-Leibler divergence (or relative entropy) of the distribution P from the distribution Q .

$$\text{KL}(P, Q) = \sum_{i \neq j} p_{ij} \log \frac{p_{ij}}{q_{ij}}$$

We then minimize the Kullback-Leibler divergence with respect to the points $\mathbf{y}_i$ using gradient descent.

2 Natural language processing and classification

Once we have a suitable database, we can start building the classifiers. Unfortunately, our data, in text format, is hard to manipulate for a program. The manipulation of text is a subfield of Natural Language Processing (NLP) which is an important topic of research in computer science. Before feeding the articles to the classifiers, they will thus be manipulated and then transformed to be easier to handle for the classifier. This is done with the preprocessing and vectorization step.

Once the structure of a classifier is built, it is trained on the major part of the database but not on all of it. Indeed, the database is usually splitted in two unequal parts. The largest subset, called the training set, contains around 80% of the data and is used to train the model and fine-tune its hyperparameters. The smaller subset is the test set and is used to evaluate the classifier. This evaluation may be used to detect overfitting. Overfitting can happen when the model fits too exactly the training data and consequently doesn't generalize well from our training data to unseen data.

2.1 Preprocessing

A text contains a lot of information that is usually considered unimportant for the classification. The goal of the preprocessing phase is to simplify the input texts without losing much information. Of course, it depends a lot on the classification task. We will here present the most common preprocessing operations.

2.1.1 Cleaning

A very common action is to convert every word to lower case for the computer to recognize words independently of their positions in the sentence.

The cleaning can then consist in removing substrings that are assumed insignificant for the classification. This is: removing punctuation, non-alphanumeric characters, multiple whitespaces, websites, numeric characters. It may also remove stop words. Stop words are commonly used words which are often ignored when working on texts. Typical english stop words are: a, in, who, from, on,... The model may also strip short words. Those words can come from typos but also after the above removals, for example: *should've* $\rightarrow$ *should ve*, the *ve* will be removed. Here is a simple example:

He was 22 years old, but John stayed the same! $\rightarrow$ years old john stayed

2.1.2 Stemming

Stemming is the process of reducing inflected words to their word stem, base or root form. The goal is again to simplify the text. On the previous example it becomes:

years old john stayed $\rightarrow$ year old john stay

2.2 Vectorization

Modelling text is hard for a program, a mathematical representation of the text is often required. The text is then converted into a vector. This vectorization can be done in a lot of ways depending on which linguistic property/feature one wants to focus on. We present here three commonly used models.

2.2.1 Bag of words

Bag of words is a simple way to vectorize text. It is flexible and can thus be used in many different ways. It involves two things:

- A vocabulary of known words
- A measure of the occurrence of known words.

Hence, no information about the word order or text structure is kept. This is the reason for calling it *Bag of words*. Once we have the length of the vocabulary by browsing the documents of the dataset, we can compute the vector representation of the articles. This is done by counting the occurrences of the known words in the article.

Here is a simple example. A vocabulary of six words is already built, each document will thus be vectorized as a vector of size 6.

	vocabulary:	<i>Ethan</i>	<i>plays</i>	<i>John</i>	<i>is</i>	<i>strong</i>	<i>football</i>
<i>John plays football, Ethan plays football</i>	$\rightarrow$	1	2	1	0	0	2

The design of the vocabulary and the scoring of words can be modulated in order to reflect wanted linguistic properties.

2.2.2 Tf-idf

The *term frequency-inverse document frequency* or tf-idf technique is similar to the Bag of Words in the way that it uses a fixed vocabulary to form the vector. It differs in its way of scoring for each word. The scoring technique is chosen to reflect how relevant a word is to a document in a collection of documents and is given by this general formula:

$$\text{tfidf}(t, d, D) = \text{tf}(t, d) * \text{idf}(t, D)$$

- The term frequency (tf) is proportional to the word frequency in a document
- The inverse document frequency (idf) is a measure of how much information the word provides, i.e., if it's common or rare across all documents.

For both values, various implementations are possible.

Weighting scheme	tf(t,d)	Weighting scheme	idf(t,D)
binary	0, 1	unary	1
raw count	$n_{t,d}$	inverse document frequency	$\log(\frac{N_D}{N_{D,t}})$
term frequency	$\frac{n_{t,d}}{n_d}$	inverse document frequency smooth	$\log(\frac{1+N_D}{1+N_{D,t}}) + 1$

Table 1: Possible schemes to compute tf(t,d) and idf(t,D) weights

In the Tables 1 the following symbols are used:

- $n_{t,d}$: number of occurrences of word t in document d
- n_d : number of words in document d
- $N_{t,D}$: number of documents containing word t in collection D
- N_D : number of documents in collection D

Table 1 gives a non-exhaustive list of possible schemes. We note that by choosing raw count for tf(t,d) and the unary weighting for idf(t,D) we get the Bag of Words technique. The two techniques are thus closely related. The more complex weighting schemes are widely used in information retrieval or summarization.

2.2.3 Word2Vec

Word embedding is the concept of mapping words to vectors of real numbers. Word2Vec designates a type of word embedding in which the models are shallow, two-layer neural networks that are trained to reconstruct linguistic contexts of words. Word2Vec takes as its input a large corpus of text and produces a vector space, typically of several hundred dimensions, with each unique word in the corpus being assigned a corresponding vector in the space. This vector space is built such that it depicts some semantic meaning and relationship between the words. A recurrent example to illustrate this:

$$\overrightarrow{king} - \overrightarrow{man} + \overrightarrow{woman} \approx \overrightarrow{queen}$$

The similarity between words is computable in the vector space by measuring the cosine similarity. This is the cosine of the angle between two corresponding vectors in multi-dimensional embedding space. If $\mathbf{A}$ is the embedding of *word1* and $\mathbf{B}$ the embedding of *word2*, the similarity is:

$$\text{similarity}(\text{word1}, \text{word2}) = \cos(\theta) = \frac{\mathbf{A} \cdot \mathbf{B}}{\|\mathbf{A}\| \cdot \|\mathbf{B}\|}$$

The reason cosine similarity captures better the semantic similarity than euclidian distance is due to the training process. The word vector magnitudes (which Euclidean distance depends on) are impacted by the occurrence counts of the words while the angles between vectors are less influenced by it. The semantic similarity between a word that appears often and a scarcely occurring word in the training set is thus better captured by an angle measure.

A comprehensive presentation of the Word2Vec model can be found in [14].

2.3 Classifiers

The output of a binary classifier is a value $\hat{y} \in [0, 1]$, called the prediction probability. It represents the confidence of the classifier to label the input as class 1, In our context, class 0 is the reliable class and class 1 is the fake class. Hence, if we have an output $\hat{y} = 0.2$, the classifier has 20% confidence the input belongs to the fake class and thus 80% confidence it belongs to the reliable class. In many cases, the returned probability needs to be converted to a binary value. To do this, one needs to define a classification threshold. In this case, a value above the threshold indicates 'fake' and a value below indicates 'reliable'. The threshold is problem-dependent and in this document, we will fix the threshold to the most common value: 0.5.

2.3.1 The binary cross-entropy loss function

For a classifier to learn from observations, it needs to quantify how close the classifier output $\hat{y}$ is to the correct output y [15]. Such a function is called a loss function.

$$L(\hat{y}, y) = \text{How much } \hat{y} \text{ differs from the true } y$$

For binary classification tasks, the binary cross-entropy loss function is often used. To understand better how it works, we show how to get to its definition.

The probability $p(y|x)$ that the classifier classifies an observation x in the correct class y can be expressed as the following (We observe that if $y = 1$, it simplifies to $\hat{y}$; if $y = 0$, it simplifies to $1 - \hat{y}$):

$$p(y|x) = \hat{y}^y (1 - \hat{y})^{1-y}$$

We thus want to maximize this probability. To make it more easier to work with mathematically, we take the log of both side, it will remain the same maximization problem:

$$\log(p(y|x)) = y \log(\hat{y}) + (1 - y) \log(1 - \hat{y})$$

By definition, a loss function is to be minimized, we will thus simply flip signs and we get the binary cross-entropy loss function:

$$L(\hat{y}, y) = -(y \log(\hat{y}) + (1 - y) \log(1 - \hat{y}))$$

2.3.2 Logistic Regression

The logistic regression assumes a linear relationship between the input $x = (x_1, x_2, \dots, x_n)$ and the log-odds (or logit function) of the prediction probability $\hat{y}$.

$$\text{logit}(\hat{y}) = \log\left(\frac{\hat{y}}{1 - \hat{y}}\right) = wx + b$$

The variables w and b are the weights learned during the training process. By algebraic manipulation we get $\hat{y}$:

$$\hat{y} = \frac{1}{1 + e^{-(wx+b)}} = \sigma(wx + b)$$

The logistic regression is thus a sigmoid function. The weights w_{tot} (w and b) are learned by minimizing the mean of the binary cross-entropy on the training samples, this mean is called the minimization function. To prevent from overfitting, a penalty for weights with big values is added in the minimization function, this is called regularization. A common regularization type is the L2, which consists in adding $\lambda \sum_{i=1}^n w_{tot,i}^2$ to the minimization function. The minimization function becomes:

$$J(w_{tot}) = \frac{1}{m} \sum_{j=1}^m L(\hat{y}_j, y_j) + \lambda \sum_{i=1}^n w_{tot,i}^2$$

with m the number of training samples and i the number of weights.

2.3.3 Bernouilli Naive Bayes

This probabilistic classifier is based on applying Bayes' Theorem:

$$P(B|A) = \frac{P(A|B) * P(B)}{P(A)}$$

Two assumptions are made:

- The features (inputs) are independent. That is: the presence of one particular feature does not affect the other. This assumption is rarely satisfied in reality and is the reason the model is called naive.
- All the features have an equal effect on the outcome.

We have an observation $x = (x_1, x_2, \dots, x_n)$. According to Bayes' Theorem we have thus:

$$P(y|x) = \frac{P(x|y) * P(y)}{P(x)}$$

With the assumption of independence we get

$$P(y|x_1, x_2, \dots, x_n) = \frac{P(x_1|y)P(x_2|y)\dots P(x_n|y)P(y)}{P(x_1)P(x_2)\dots P(x_n)}$$

The denominator doesn't depend on y and is therefore not interesting for our classification.

$$P(y|x_1, x_2, \dots, x_n) \propto P(y) \prod_{i=1}^n P(x_i|y)$$

An observation is classified to the class y with the highest probability.

$$y = \operatorname{argmax}_y P(y) \prod_{i=1}^n P(x_i|y)$$

To better understand how it is applied in the text classification context, let's try to find the probabilities $P(y)$ and $P(x_i|y)$ after a bag of words vectorization:

$$P(y) = \frac{D_y}{D}$$

$$P(x_i|y) = \frac{S_{yi}}{\sum_{t \in V} S_{yt}}$$

With D_y the number of documents in class y , D the number of documents in the database, V the vocabulary and $S_{yi} = \sum_{d \in y} x_i$ ($d \in y$: the documents in class y) is the number of occurrences of the word associated to i in the class y . A problem is raised by a never occurring term i in a class, such that $P(x_i|y) = 0$, making the class y ineligible if i appears in the input. To avoid this, a smoothing factor α (Laplace smoothing) is introduced such that:

$$P(x_i|y) = \frac{S_{yi} + \alpha}{\sum_{t \in V} (S_{yt} + \alpha)}$$

2.3.4 Random Forest

Random Forest Classifier algorithm works on the basis of majority votes of multiple decision trees constructed during the training.

As a reminder, decision trees work by recursively selecting features and splitting the dataset on those features. In order to find the best split, it will try to minimize the gini index or the entropy:

$$G = 1 - \sum_{i=1}^n (p(c_i))^2$$

$$Entropy = - \sum_{i=1}^n p(c_i) \log_2 (p(c_i))$$

With n being the number of classes and $p(c_i)$ is the fraction of samples of class i in the current branch. Both values are a measure of impurity.

Random Forest is based on the concept of the wisdom of crowds: A large number of relatively uncorrelated trees operating as a committee will outperform any of the individual constituent models [16]. To have uncorrelation between the trees, random forest will take advantage of the sensitivity of decision trees to the data they are trained on. Indeed, small changes to the training set can result in significantly different tree structures. Random forest will thus train each tree on a different training set. This is done using Bootstrap Aggregation (Bagging) which generates new training sets by sampling from the database uniformly and with replacement.

2.3.5 Bidirectional LSTM

LSTM or Long Short-Term Memory [17] is a kind of recurrent neural network (RNN) that fits well to temporal or sequential input such as texts. An RNN is an artificial neural network with loops, allowing the information to persist. In Figure 1, we see on the left a RNN with loop and on the right the, often used, unrolled version of it.

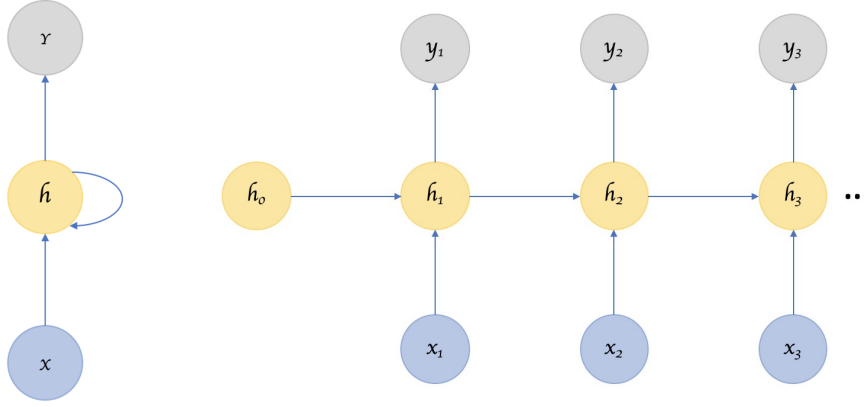


Figure 1: RNN with its unrolled representation, image taken from [18]

In the context of text classification, the words x_i from the article are successively fed into the RNN. When the model processes a word, the processing is affected by the previous words through the inner loop.

LSTM differs from regular RNN by the composition of its units. A common LSTM unit is composed of a cell, an input gate, an output gate and a forget gate. The cell remembers values over arbitrary time intervals and the three gates regulate the flow of information into and out of the cell. It is defined by these equations:

$$\begin{aligned}
 f_t &= \sigma_g(\mathbf{W}_f x_t + \mathbf{U}_f h_{t-1} + b_f) \\
 i_t &= \sigma_g(\mathbf{W}_i x_t + \mathbf{U}_i h_{t-1} + b_i) \\
 o_t &= \sigma_g(\mathbf{W}_o x_t + \mathbf{U}_o h_{t-1} + b_o) \\
 c_t &= f_t \circ c_{t-1} + i_t \circ \sigma_c(\mathbf{X}_c x_t + \mathbf{U}_c h_{t-1} + b_c) \\
 h_t &= o_t \circ \sigma_h(c_t)
 \end{aligned}$$

To evaluate a word and its effects, it is important to know its context, i.e. to consider the words surrounding it. By using RNN, we would only take the previous words into account, but in text classification, we have access to the next words too. This is why a Bidirectional Recurrent Neural Network (BRNN) - in which information flows in both sides - is preferable (Figure 2).

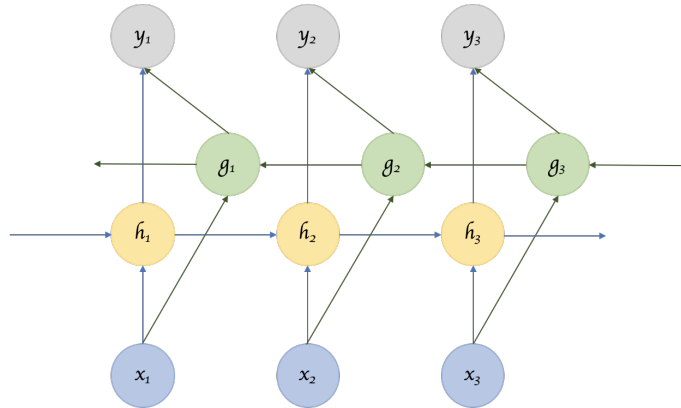


Figure 2: BRNN

We have a Bidirectional LSTM when a BRNN is composed of LSTM units.

3 Evaluation Metrics

3.1 Accuracy

Accuracy is the ratio of number of correct predictions to the total number of input samples.

$$\text{Accuracy} = \frac{\# \text{ correct prediction}}{\# \text{ prediction made}}$$

It is an interesting metric mainly when the number of samples belonging to each class is equal.

3.2 Area Under ROC Curve (AUC)

An ROC curve¹ (Receiver Operating Characteristic curve) is a graph showing the performance of a classification model at all classification thresholds. This curve plots two parameters:

$$\begin{aligned}\text{True Positive Rate (TPR)} &= \frac{\text{TP}}{\text{TP} + \text{FN}} \\ \text{False Positive Rate (FPR)} &= \frac{\text{FP}}{\text{FP} + \text{TN}}\end{aligned}$$

An ROC curve plots TPR vs. FPR at different classification thresholds. Lowering the classification threshold classifies more items as positive, thus increasing both False Positives and True Positives. AUC stands for the area under the ROC curve, it is thus the integral of the ROC curve from 0 to 1 on the FPR-axis. There are two particularities with AUC:

- It is scale-invariant. It measures how well predictions are ranked, rather than their absolute values.
- It is classification-threshold-invariant. It measures the quality of the model's predictions irrespective of what classification threshold is chosen.

3.3 F1 score

F1 score is the harmonic mean between precision and recall.

The **precision** is the number of correct positive results divided by the number of positive results predicted by the classifier.

$$\text{Precision} = \frac{\text{TP}}{\text{TP} + \text{FP}}$$

The **recall** is the number of correct positive results divided by the number of all samples that should have been identified as positive. It is a synonym for TPR.

$$\text{Recall} = \frac{\text{TP}}{\text{TP} + \text{FN}}$$

The harmonic mean is thus

$$\text{F1} = \frac{2}{\frac{1}{\text{precision}} + \frac{1}{\text{recall}}}$$

The F1 score tells you how precise your classifier is (how many instances it classifies correctly), as well as how robust it is (it does not miss a significant number of instances).

¹<https://developers.google.com/machine-learning/crash-course/classification/roc-and-auc>

4 Adversarial Example

Adversarial examples were introduced in [19] as an imperceptible non-random perturbations to the input of a neural network that change its prediction². This type of attack is particularly studied in the image classification task. Figure 3 is a well-known example: for the human eye, we see no differences in both pandas images, however, the classification changes.

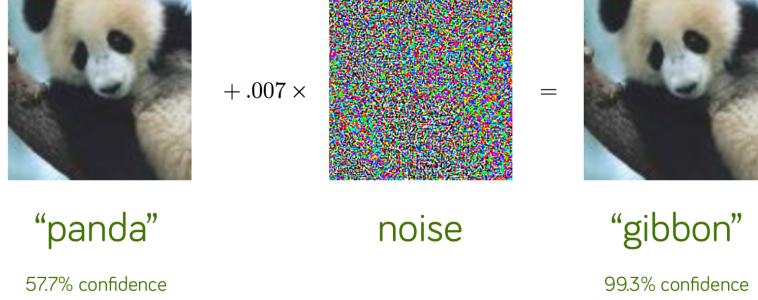


Figure 3: Adversarial example on a panda image

We can think of the classifier as defining a map of the input space, giving the most likely class within each region. The classifier can generally only learn an approximation of the true boundaries. The model error between the approximate and expected decision boundaries is exploited by adversaries. This is illustrated in Figure 4, the dotted line representing the real boundaries and the solid line represents the boundary learned by the model. An adversarial example happens when a modified input goes over only one line. This suggests that inputs close to the decision boundary (what goes with classification with low confidence) are easier targets.

4.1 Adversarial constraints

In image classification, imperceptible perturbations mean a very small change of color of the pixels. We have thus a constraint when building the adversarial example. This concept of constraint can be generalized: given a classifier f , we say that x' is an adversarial example of x targeting class y' if

$$f(x') = y' \quad \text{and} \quad c(x, x') \leq \gamma$$

with c a domain-specific constraint function $c : \mathcal{X} \times \mathcal{X} \rightarrow \mathbb{R}_+^L$ and a vector of bounds $\gamma \in \mathbb{R}^L$ that capture the notion of imperceptible perturbation via $L \geq 1$ constraints[20]. In image classification and continuous problems in general, the constraint will typically be a maximum modification ϵ of each element of the input. But in the case of text classification, the input is discrete which makes the similarity constraint harder to determine. The similarity between two texts can be captured on two levels:

- **Semantic similarity:** The meaning of the adversarial article is the same as for the original article.
- **Syntactic similarity:** The sentence structure in the adversarial article follows the same rules as in the original article.

²<https://towardsdatascience.com/breaking-neural-networks-with-adversarial-attacks-f4290a9a45aa>

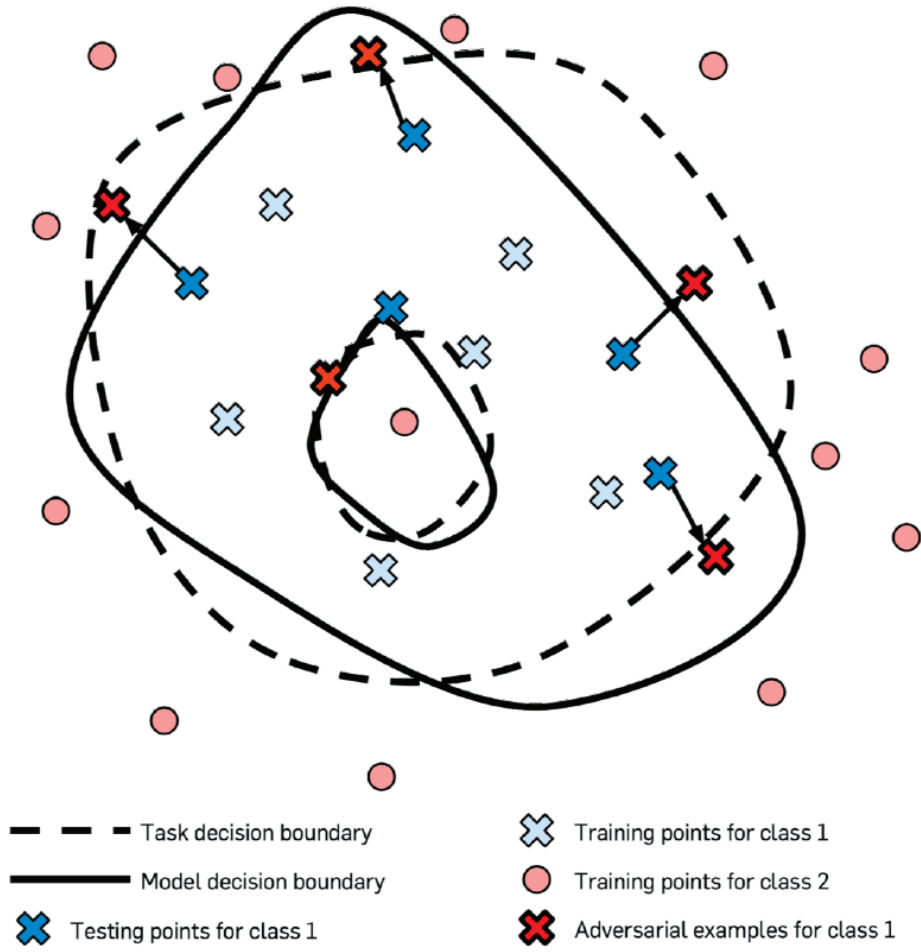


Figure 4: Caption

The strongness of the different constraints depend on the goals of the adversary. An adversary that invent fake news to earn some money may prefer quantity over the quality of the articles and may be more lax on these constraints in order to produce more articles. On the other hand, if the adversary makes a targeted adversarial example, the meaning of the text matters and the constraints will be stronger.

4.2 Adversarial capabilities

As is common in modelling the security of any domain, we can distinguish different types of attacks depending on the capabilities of the adversary. We distinguish two types:

In **White Box** attack on a machine learning model, an adversary has total knowledge of the model. This is: the adversary knows the parameters of the fully-trained model, the training algorithm and the training data.

A **Black Box** attack assumes no knowledge of the model and can exploit a model by providing a series of carefully crafted inputs and observing outputs.

Those types of attacks can further be divided by different degrees of knowledge of the model[21].

Part III

Database

Before making the classifiers, we need a database on which the machine learning algorithms will be trained. It is critical that the collected data reflects the real situation for the models to be applicable outside the dataset. This is why the presentation of the building steps and the exploration of the dataset are not to be overlooked.

1 Building the database

For this project, a database of news articles labeled as fake or as reliable was thus needed. Such a database is hard to built because in a flawless database, each article should be labeled by human beings. Indeed, since the goal of our algorithm is to detect fake news, it would be absurd to build the dataset with fake news detected by an algorithm. But this manual labelization takes time and even for human beings, it can be hard to classify an article as fake or reliable.

No such database were found, we will have to accept to work with data with some flaws. Still, as the quality of the database affects the credibility of the results, extra efforts were made in verifying the consistency of the candidate databases for our problem. For example, after consequent exploration, the Fake News Corpus [22] was judged unfit to our problem due to bad scraping and dissimilarity in the topics of reliable and fake articles. This last aspect is important to avoid classification based on topics instead of reliability. Indeed, if the collected fake articles are mostly about football and the reliable news about politics, the classification will probably be based on that. It will thus be inefficient when applied on articles outside the dataset.

At the end, no satisfying databases were found, we had thus to build one. We built it in 2 steps. The first step consisted in collecting the fake news articles, the second was to scrape the reliable articles.

The fake articles all come from a database from Kaggle [23]. The articles were scraped on websites listed in a blacklist of 207 unreliable websites³. The articles span over a period of 30 days around the 2016 US election (from 26th of October to 25th of November). Due to bad scraping, there was still a lot of unwanted elements not belonging to the real article. An example of those *noises* is typically the date an article was published or other meta-data not present in the original article. This is why we carefully removed specific parts at the beginning and end of some articles. However, this demanded a big effort and to avoid suppressing article parts by mistake, only the most apparent noises were removed. In the end, although much of it was suppressed, there is a relatively significant presence of noise in the database.

The reliable news had to be scraped keeping in mind that they had to be consistent with the already collected fake articles. The main difficulty was to scrape data from more or less the same period of time: scraping news from more than 3 years ago is not an easy task. Fortunately, *The New York Times* and *The Guardian* have developed an API to search for articles which made the scraping possible. Still, those are the only two newspapers found to have an API, hence, the reliable articles only

³The list was made by web activist Daniel Sieradski, it is not accessible anymore

come from these two sources. Out of those sources, we only collected articles that were labelled as *World News* and *US News* to be more consistent with the fake dataset.

For an easier understanding of the results, we aimed for 50% of fake and 50% of reliable articles. It is important to keep in mind that articles are labelled as fake depending on their source, not on their reliability, which means that some reliable articles could be labelled as fake. This is considered as the best alternative to manually labeling each article.

2 Database statistics

The goal of this section is to explore the database and try to spot patterns that could impact the result. If some patterns appear, we try to understand its origin and its impact on the future results. The dataset consists of 7058 articles, exactly half of it is labelled as reliable and the other half as fake.

2.1 Sources exploration

The fake articles come from 207 different websites.

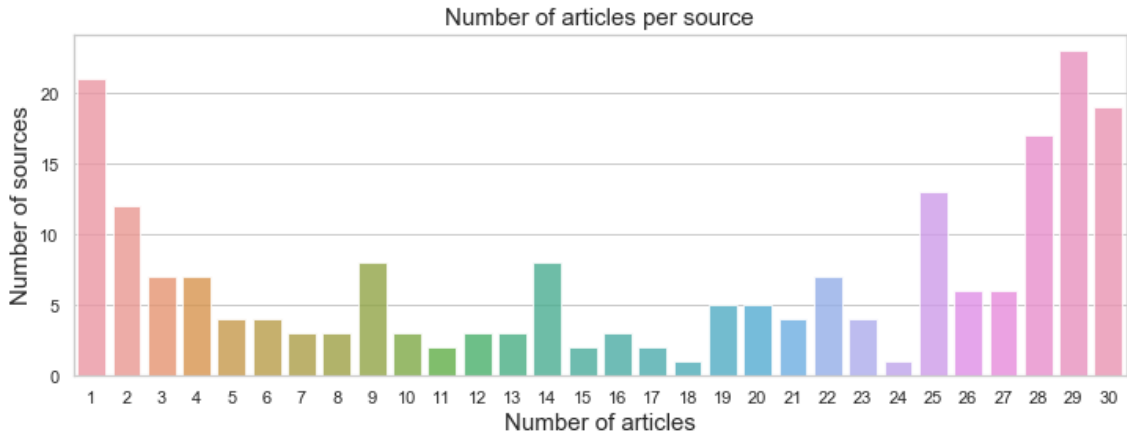


Figure 5: The graph shows how many sites (y-axis) are source of how many articles (x-axis)

As Figure 5 shows, the maximum number of article per source is 30. There is thus no website that contributes more than 1% of the articles in the database. Hence, the database is well diversified and if a problem occurs on a website (bad scraping, bad labelling,...) it doesn't affect too much our database.

There are only two sources for the reliable news, the distribution is given in the table 2. This small diversity in the sources is not considered a problem since the available APIs ensured a good and targeted scraping.

The New York Times	The Guardian	Fake Corpus
1557	1972	3529

Table 2: The distribution of the articles over the sources

2.2 Date exploration

As said earlier, the data spans over a specific period of time, which is rendered in Figure 6.

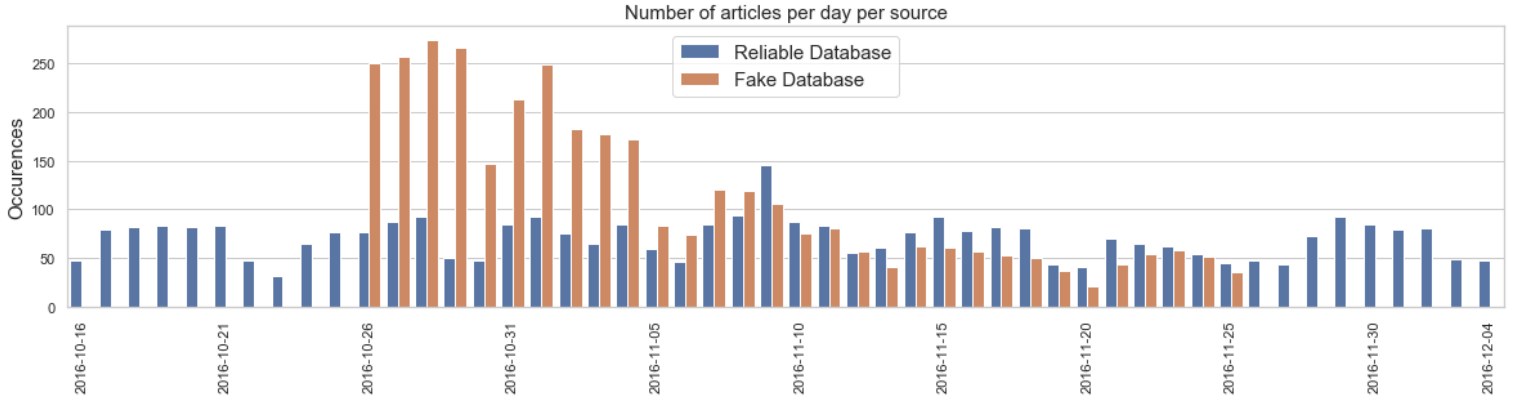


Figure 6: The distribution of the articles over time

We can see in this graph that reliable articles exceed a bit the timezone of the fake articles. As said earlier, scraping news article from 3 years ago is not an easy task and we could only collect a few articles per day. If we only scraped real articles between 26-10-2016 and 25-11-2016 the database would have been too small. We assume here that by adding 10 days before and 8 days after, the exceeding time windows are sufficiently close to not impact our analysis.

We also observe the imbalance in the time distribution of the fake articles. This could have been avoided at the expense of the homogeneity in the sources, i.e. a more equal distribution in time but with sources that contribute significantly more to the database. We judged that the results would be less impacted with the current composition since the uncertainty around the sources is bigger. Yet, this imbalance in the time distribution and the fact that a major part of the fake articles were published before Trump’s election (9-11-2016) may impact our results.

2.3 Topic exploration

The similarity in the topics of the fake and reliable articles in the database is important to ensure that the classification will not be biased. As seen in Chapter II, tf-idf is widely used in information retrieval or summarization, it is thus well suited for studying the major topics of the articles. The idea here is to find the most relevant words according to tf-idf over the whole database, and then compare the score of those words in the reliable and fake subsets. This is done in Figure 7, after preprocessing of the texts (section 2.1). We observe that the distribution is more or less the same but we can make some interesting observations.

- *Trump* is the most important words in both subdatasets
- *Clinton* placed second in the fake database and third in reliable database
- *said* is surprisingly important in the reliable articles. Since it is a neutral word it is of few importance to find topics. However, it will be very interesting for the classification.
- *Hillary* (*hillari* after preprocessing) is important for the fake dataset. This suggests a particular writing style in the fake articles.

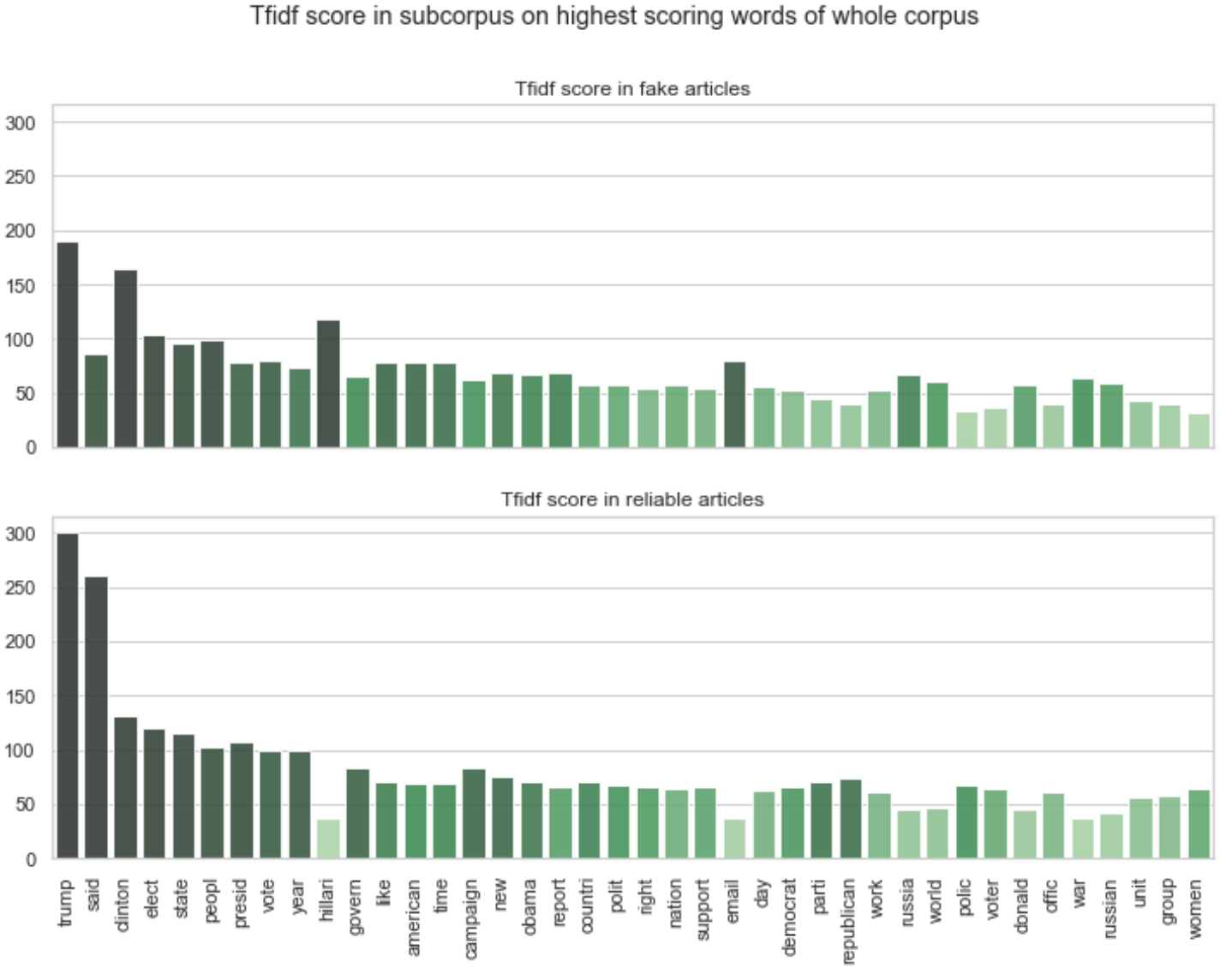


Figure 7: Comparison of the tf-idf scores

- *Email* is here very important due to the Clinton email scandal which seemed to draw more the attention of the fake websites.

Those result hint that the fake articles tend to more sensational topics but overall the topics seem relatively similar. We can already guess some strategies to fool the classifiers: how does the adding of *said* and removal of *Hillary* impact the classification?

2.4 Visualization with t-SNE

Visualization of the data may be interesting to find some hidden patterns in our database. Here, we make the visualization to explore the relation and similarity between the fake and reliable articles. If two clusters of reliable and fake articles stand out, it may mean a bad composed database with a too big dissimilarity between both types of articles. On the other hand, if the distribution is completely homogeneous, it means a very well constructed database but it implies a difficult classification.

T-SNE, presented in Chapter II, takes vectors as input. To transform the texts into vectors, we

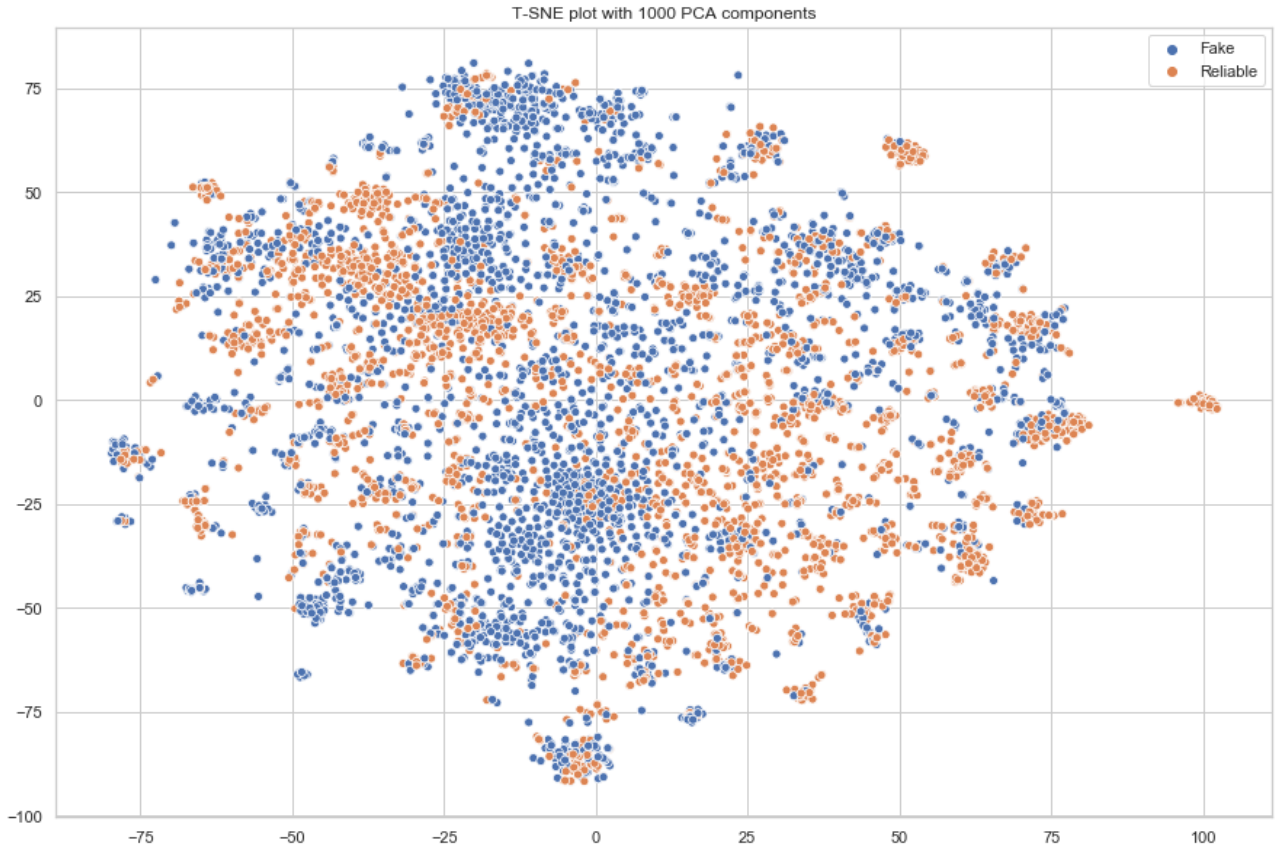


Figure 8: t-SNE with pre-reduction with T-SVD to 1000

use tf-idf since it keeps a sense of the text topics. Applying tf-idf gives us a sparse matrix, with each row representing an article and each column representing a word. Each article is thus represented by a high-dimensional vector and since tf-idf is subject to the curse of dimensionality, we reduce the dimensionality of the vectors with truncated SVD. Contrary to PCA, truncated SVD does not need to center the data first. It is thus particularly efficient for sparse matrices. The difficulty is to find the middle ground between reducing the dimension before applying the t-SNE to avoid the curse of dimensionality, but not reducing too much to keep a maximum of the variance.

In Figure 8, we first reduce the dimension to 1000 which explains 74.6% of the variation. A similar visualisation is obtained in Figure 9 when we reduce to 500 dimensions, it then merely explains 57.2% of the variance. In both figures we observe some spots with more fake or with more reliable articles but overall no clusters of a particular label can be made. On Figure 10 we have the same scatterplot but with the topic of the reliable articles highlighted. We see that the fake articles are indeed about both topics. The important is that it reveals no obvious low-dimensional structure our classifiers can rely on.

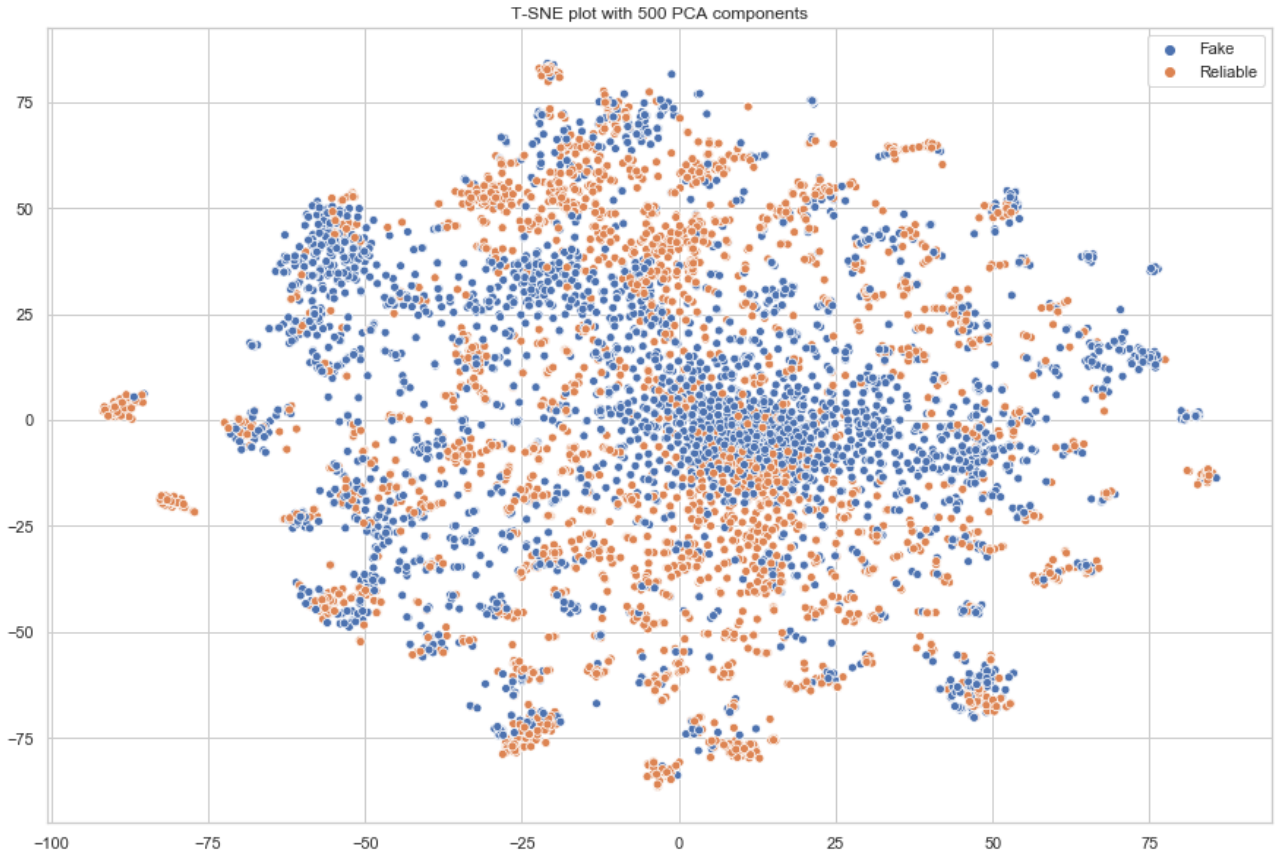


Figure 9: t-SNE with pre-reduction with T-SVD to 500

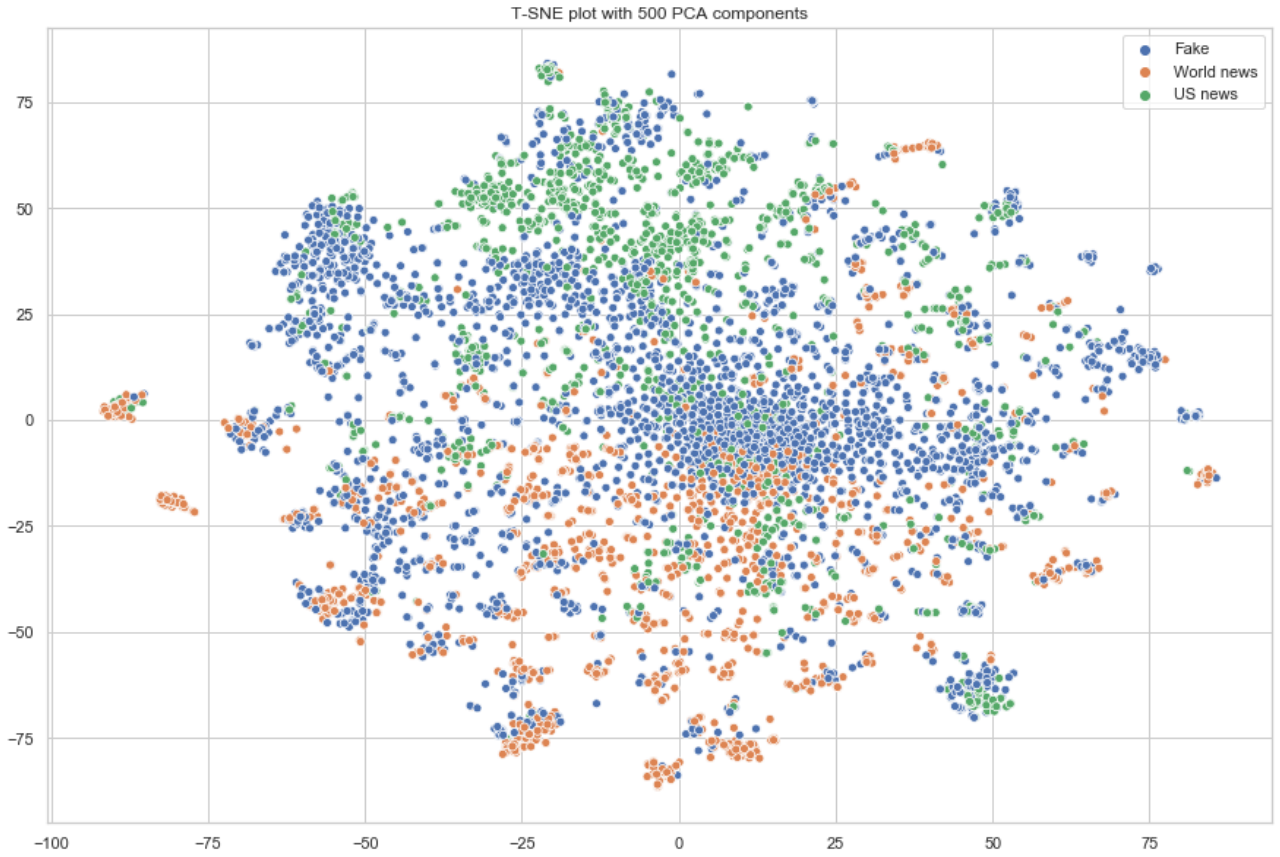


Figure 10: t-SNE with pre-reduction with T-SVD to 500

3 Conclusion

In this chapter, we built a database emphasizing on having data as homogeneous as possible. For this, we had to build a database, relying on an already existing database of fake news. Those articles were not manually labeled but were labeled according to the label of the website that published it. A few articles are thus misclassified. The fake article may also contain unwanted metadata not belonging to the original article. Those two flaws are considered as negligible and are accepted because of the difficulty to obtain a better database. We then explored the constructed database without finding obvious patterns, which suggests homogeneity. During the exploration, we discovered some possible lines of attack against classifiers.

Part IV

Classifiers

A binary classifier takes an observation as input and outputs the probability prediction $\hat{y}$ that the observation belongs to class 1. In this chapter, we apply the classification mechanisms presented in Chapter II, on the fake news detection task and train them on the database constructed in Chapter III.

To better evaluate the models and for the further study of adversarial example, we build classifiers based on 4 inherently different machine learning methods: *logistic regression*, *naive bayes*, *random forest* and *LSTM*. All four are widely used and fit well the binary classification problem.

This chapter is structured in two parts. In the first part, we describe the implementation of the classifiers, presenting the hyper-parameters and fine-tuning the models. The training and fine-tuning is done on the training set which consists of 80% of the database (5646 articles). In the second part, we compare the results on the test set which is composed of the remaining 20% of the dataset.

1 Implementation

The classifiers built are all structured in three steps.

1. Preprocessing
2. Vectorization
3. Classification

The preprocessing step is exactly the same for all the classifiers. The vectorization depends on the classifier. Therefore, we will have 4 pairs as follow: tf-idf with logistic regression, tf-idf with naive bayes, tf-idf with random forest and Word2Vec with LSTM. In this section, we first describe the implementation choices and the hyperparameters in each step and then we proceed to the fine-tuning.

1.1 Preprocessing

During this step, the model first cleans the text by removing punctuation, non-alphanumeric characters, multiple whitespaces, websites, numeric characters, stop words and short words. Afterwards, it converts to lower case and stems every word.

This step will not be fine-tuned. It is assumed to be coherent with our text classification problem since the majority of our models are based on tf-idf. Indeed, tf-idf doesn't take word order into account and it downscales the importance of words or characters that appear in many documents. Nonetheless, this preprocessing may have stronger effect on the LSTM model.

1.2 Vectorization

1.2.1 Tf-idf

The weighting scheme used is:

$$\text{tf-idf}(t, d) = \frac{n_{t,d}}{n_d} \cdot \left(\log \left(\frac{1 + N_D}{1 + N_{D,t}} \right) + 1 \right)$$

Where $n_{t,d}$ is the number of occurrences of word t in document d , n_d is the number of words in document d , $N_{t,D}$ is the number of documents containing word t in collection D and N_D is number of documents in collection D .

Tf-idf has 3 hyperparameters meant to shape the vocabulary:

- **min_df**: The minimum permitted appearances of a word in the documents. If a word occurs less often than the tolerance, it will be ignored. This dismisses a large number of typos and scarcely ever used words or names.
- **max_df**: The maximum permitted appearances of a word in the documents. In our collection the preprocessing, no words appear more than in 85% of the articles after the preprocessing step.
- **ngram_range**: To specify the N-grams allowed in the vocabulary. A N-gram is a contiguous sequence of N words. If only unigrams are extracted, no group of consecutive words will be in the vocabulary.

1.2.2 Word2Vec

Various Word2Vec models are possible. A candidate could be a pre-trained model like Google News Corpus with a vocabulary of 3 billion words embedded in a 300-dimension space [24]. Another possibility is to use a model trained on our dataset. It has the advantage it can be trained after the preprocessing and it takes thus the stemming of words better into account. We will thus favor this tailor-made model. Like google’s **Word2Vec**, the embedding is fixed to 300-dimensional vectors. This size seems fair: as shown in [25], the performance of low dimensional word embedding is poor until the embedding size reaches a lower bound specific to the collection, it then stabilizes. After the training, its vocabulary contains 12950 words. Table 3 compares the word similarities of *oil* found with the two models.

Trained Word2Vec		Pre-trained Word2Vec	
oil	gas	oil	crude_oil
	mine		petroleum
	frack		gas
	coal		Oil
	spill		natural_gas

Table 3: Most similar words for oil

1.3 Classification

1.3.1 Traditional methods

We build three traditional classifiers whose hyperparameters are already presented in section 2.3 of Chapter II. Here is a quick reminder of the hyperparameters.

Logistic Regression: The parameter C . It is the inverse regularization strength: $C = \frac{1}{\lambda}$.

Naive Bayes: The smoothing factor α .

Random Forest: The number of trees in the forest `n_estimator` and the maximum depth of the trees `max_depth`. Also the criterion of the split is to be chosen between `gini` or `entropy`.

1.3.2 LSTM

The idea of LSTM is to take advantage of the information about the word order. We it is thus paired with `Word2Vec`. Since the input is a tensor, the input has to have a predefined fixed size. The input dimension is `(batch_size, num_steps, features)`. The `num_steps` is fixed to 200 which means that only the first 200 words after preprocessing are kept and if the article is shorter, zero vectors are appended. This `num_steps` was chosen because fake articles are usually significantly shorter than reliable ones. Indeed, after preprocessing, the median of the lengths of fake articles is 263 while it is 426 for reliable articles. The number of articles with less than 200 words is 1364 for the fake articles and 390 for the reliable articles. To prevent an influence of the text size, we chose a `num_steps` of 200 words. The number of features is fixed at 300, since the embedding of the used `Word2Vec` is to a 300 dimensional space. Here is the architecture of the RNN:

Layer (type)	Output Shape
input_3 (InputLayer)	[(batch_size, 200, 300)]
bidirectional_2 (Bidirection	(batch_size, 200, 2*neuronsLSTM)
global_max_pooling1d_2 (Glob	(batch_size, 2*neuronsLSTM)
dense_4 (Dense)	(batch_size, neuronsDense)
dense_5 (Dense)	(batch_size, 1)

We fixed the parameters in order to reduce the tuning to only 4 parameters, namely:

For the architecture:

- `neuronsLSTM`: gives the number of unit in one LSTM layer (we have two same-sized LSTM layer in our bidirectional layer
- `neuronsDense`: gives the number of units in the first dense layer

For the training:

- `batch_size`: The number of samples to work through before updating the internal model parameters.
- `epoch`: The number times that the learning algorithm will work through the entire training dataset.

1.4 Fine-tuning

During the validation, we optimize our models by fine-tuning the models hyperparameters. This is done by a 5 cross-validation⁴ grid-search over a parameter grid. The parameters are evaluated on the

⁴3 cross-validation for the LSTM

						acc.
Tfidf	Logistic Regression 1	min_df=20	max_df=0.85	ngram_range=(1,1)	C=500	95.6%
	Logistic Regression 2	min_df=20	max_df=0.5	ngram_range=(1,2)	C=500	94.7%
	Naive Bayes	min_df=20	max_df=0.6	ngram_range=(1,2)	$\alpha=0.01$	84.4%
	Random Forest	min_df=20	max_df=0.6	ngram_range=(1,2)		92.6%
		criterion=gini	max_depth=None	n_estimators=1000		
Word2Vec	LSTM	neuronsLSTM=100		neuronsDense=400		90.2%
		batch_size=None		epochs=3		

Table 4: Validation of the different classifiers

	Accuracy	F1 score	Recall	Precision	AUC
Tfidf - Logistic Regression with unigrams	92.0%	92.0%	92.44%	91.79%	97.37%
Tfidf - Logistic Regression with unigrams and bigrams	92.56%	92.56%	92.86%	92.47%	97.67%
Tfidf - Naive Bayes	80.95%	80.85%	87.25%	77.78%	88.59%
Tfidf - Random Forest	88.03%	88.03%	85.85%	90.0%	95.56%
W2V - LSTM	89.52%	89.52%	89.22%	89.97%	96.17%

Table 5: Evaluations of the models

accuracy. The results can be found in appendix A. We keep two logistic regression classifier. Even if the first one scores slightly lower, we decided to keep it because of its simplistic nature and its negligence for the word order.

We thus built 5 models. In the next parts of this document, they may be mentioned by the name of their classification algorithm. *Logistic regression* then stands for the model processing only unigrams and *logistic regression 2* stands for the model working with unigram and bigrams.

2 Results

After the training of the 5 models, we can now compare the results. Figure 5 resumes the general evaluation scores. The ranking of each evaluation score is always topped by Logistic Regression 2 and closely followed by Logistic Regression with unigram. We also note the relatively low scores for the Naive Bayes. In Figure 11, we see the ROC-curves on a graph, the two logistic regression models and the LSTM form the three best scores on AUC. Figure 12 shows the learning curve, all the classifiers seem to have enough data to learn from.

Finally, the distribution of the confidence of the classification might be an important characteristic for the robustness of a model. Table 6 shows the proportion of fake articles labeled as fake with more than 95% and 99% of confidence. We notice that the Naive Bayes classifier is particularly confident, this is typical for Naive Bayes. It is explained by the fact that the Naive Bayes algorithm doesn't compute a probability directly, rather it computes a *raw score* for each class. The *raw scores* are then converted into a prediction probability and are almost always either very close to zero or very close to one. For the Naive Bayes, it is thus difficult and risky to interpret the prediction probability. Contrary to Naive

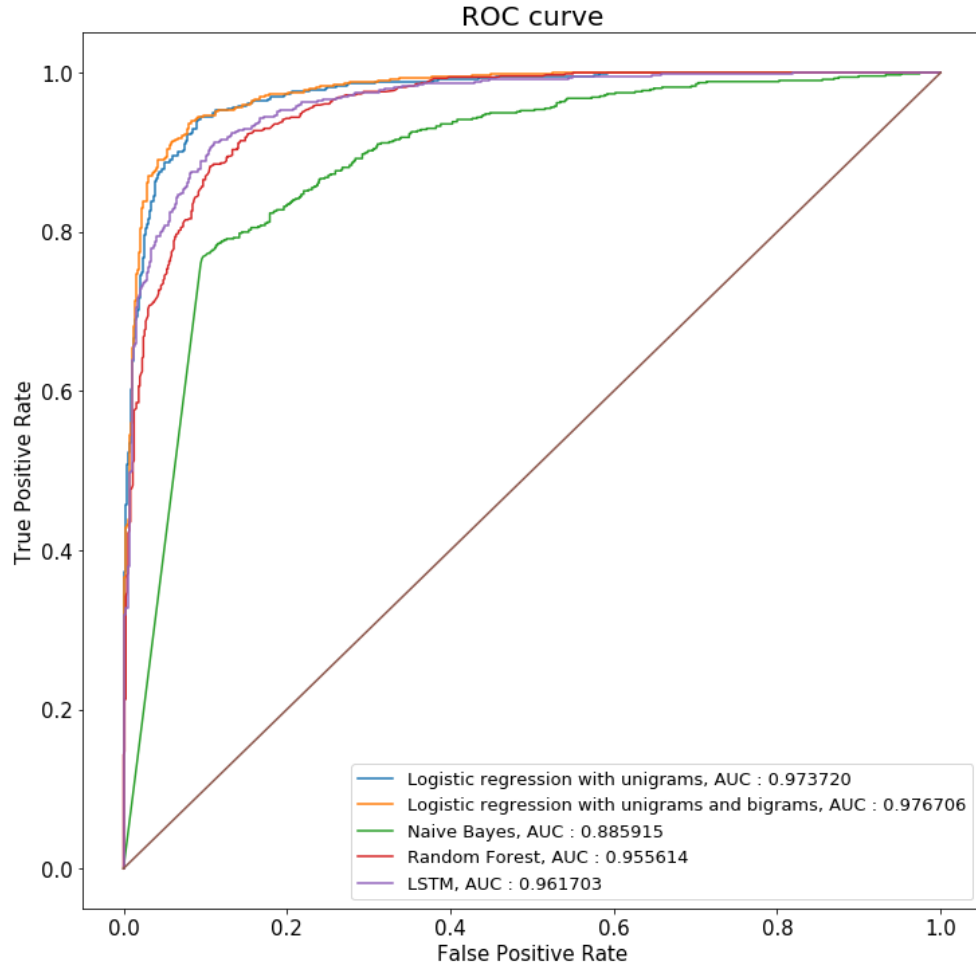


Figure 11: ROC curve

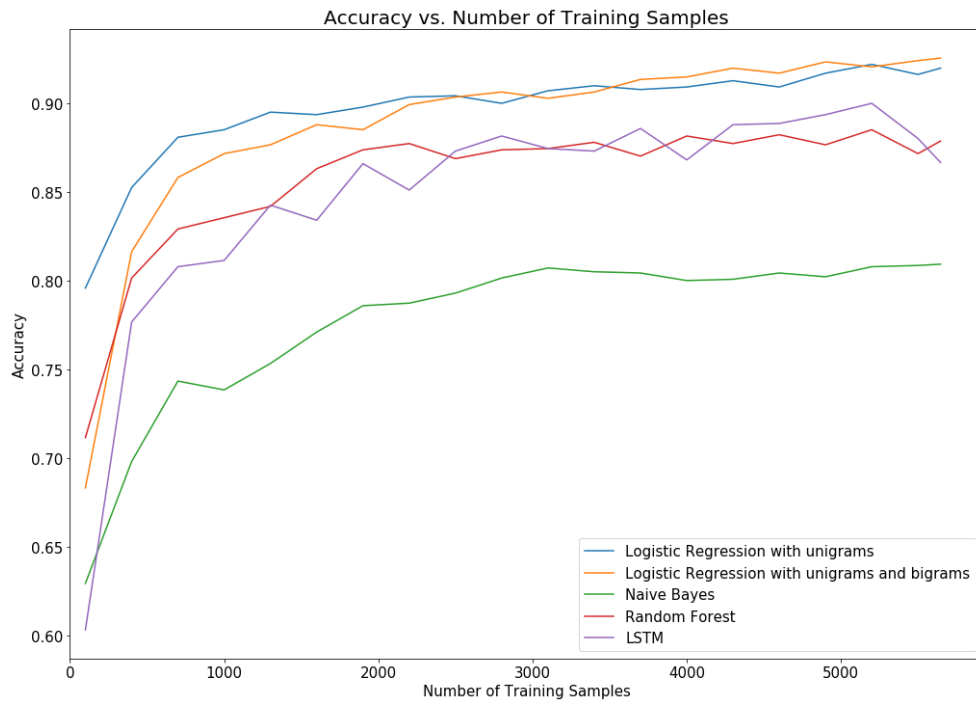


Figure 12: Learning rate

	>95%	>99%
Logistic regression	79.7%	66.9%
Logistic regression 2	76.3%	62.8%
Naive Bayes	86.1%	85.5%
Random Forest	1.2%	0.3%
LSTM	63.86%	43.13%

Table 6: Distribution of the confidence for fake classification on fake articles

Bayes, the Random Forest has only 1.2% of fake article classifications above 95% confidence, this is due to its strategy of taking the the mean of the prediction of multiple decision trees. We note also that both Logistic Regressions are significantly more confident than the LSTM. Again, both logistic regression have similar results. However, one has to keep in mind that because the construction of the prediction probability differs strongly between the models, the interpretation of this value should be specific to each classifier. The comparison of the confidence between classifiers has to be done with caution.

3 Conclusion

In this chapter, we implemented and trained 5 classifiers on the dataset. The different models were chosen for their reputation as classifiers but also for their inherently different strategies for classification. The LSTM takes only 200 words because the implementation required a fixed input. We then saw that all the classifiers have scores in the same range except for the Naive Bayes, which scores significantly lower. In all the evaluation metrics, the logistic regressions score the best, followed by the LSTM. Finally, we looked at the confidence in classifications and we noticed that the confidence of the Naive Bayes when classifying an article as fake is usually close to 100%. We will thus be careful when interpreting its results. The Random Forest generally has a lower confidence.

Part V

Adversarial Example

In the previous chapter, we built effective classifiers that label articles with an accuracy of up to 92.56%. But beside the accuracy and other scores found in table 5, an important characteristic of a classifier is its robustness against adversarial examples. In the context of fake news detection, an adversarial example is an article resulting from small modifications on a fake article correctly classified so that it becomes classified as reliable, i.e. so that the prediction probability is lower than 0.5. The feasibility of these attacks and the possible patterns in the reactions of the classifiers are what we're going to explore in this chapter.

To go deeper in the understanding in the time available, we're going to restrict the study to only two of the five classifiers we created in the previous section. We will use two classifiers with good performances and with fundamentally different classification strategies. The first classifier is the traditional logistic regression combined with tf-idf. We will use the logistic regression that only processes unigrams since it scores well and because it doesn't considers the word order at all. It is thus a model that relies on superficial cues without understanding language. The second classifier is the LSTM with Word2Vec, a recurrent neural network that can take intrinsic patterns of languages into account. The other three classifiers will solely be used to study transferability and will thus not be direct targets of attacks.

The strategy of the adversary depends on its knowledge of the targeted classifier. We will consider 2 levels of knowledge:

- **Black Box:** The adversary can provide a series of inputs and observe the outputs and he has access to the test set.
- **White Box:** The adversary has the black box knowledge plus he knows the architecture and all the parameters of the classifier and has access to the training set.

The strategy of the adversary also depends of its goals, 2 situations are considered. In the first situation, the adversary, having black box knowledge, wants to produce a lot of fake news without being too strict on the preserving of the meaning and the syntax of the adversarial article. This adversary goal with white box knowledge is left for further work.

The second situation is when the adversary wants to preserve the meaning of the original text and does not need a high production rate. In this situation, the adversarial article has to respect stronger constraints and will thus need human intervention to make the text modifications. We build a strategy with white box knowledge and then with black box knowledge. Both strategies rely on the identification of words, called Reliable Hot Words (RHW), that influence the classification towards the reliable class and the identification of words within the targeted article that influence its classification towards the fake class, we will call them Article's Fake Hot Words (AFHW).

The three strategies are resumed in Table 7.

In what is being described as another ‘bizarre’ attempt to sabotage her own campaign, Hillary Clinton has desecrated a series of beloved US symbols, including punching a bison, setting fire to the Stars & Stripes and spitting at Jerry Seinfeld. The Presidential hopeful seems determined to make a series of unprovoked errors, not least of which was agreeing to Bill hosting a sleepover for a group of Girl Guides. Short of dressing the Statue of Liberty in a Burka, Mrs Clinton has lurched from one PR blunder to another. Commented one journalist: ‘The Presidential race is entering the final furlong and if Mrs Clinton was horse – and before you can say Benghazi – she’s gone from bookie’s favourite to an ingredient at the local glue factory’. Having already become the unwitting focus of various health scares and FBI investigations, Mrs Clinton’s campaign is as orderly as a Marx Brothers movie. Her lead in the polls has been cut as video emerges of her lighting a cigar with a rolled up Bill of Rights, then proceeding to take a dump on the White House lawn. Hillary’s erratic behaviour has seen her sing the Star-Spangled Banner in Korean, dress as Oprah Winfrey for Halloween and pebble-dash Mount Rushmore. Remarkd a flummoxed advisor: ‘She keeps doing the unthinkable – like making Donald Trump electable’. Share this story...

Example 8: Original text of main example

	Knowledge	Goals
1	Black Box	Producing a lot of adversarial articles at the expense of their quality
2	White Box	Making few but coherent articles that preserve the meaning of the original texts
3	Black Box	Making few but coherent articles that preserve the meaning of the original texts

Table 7: Description of the different situation of the adversary

To compare the different results using each scheme, we will try to make an adversarial example out of the same article with each strategy. This main example is the article in Example 8. It is a fake article from the test dataset that is correctly classified by both classifiers as we can see in Table 9. This article was chosen for its small size which simplifies the overview of the multiple text modifications and for the prediction probability relatively close to the classification threshold, this may make it easier to make the prediction probability go below the 0.5 threshold.

	$\hat{y}$ on Original text
Logistic regression	0.89
LSTM	0.64

Table 9: Prediction probabilities on original text

Once an adversarial strategy is presented, it will be tested on multiple target articles. Finally, we will briefly study the transferability of the constructed adversarial examples with the classifiers built in the previous chapter.

1 Soft constraints with high production rate

The first scheme supposes that the adversary wants to produce automatically adversarial examples from a list of articles. To do this, the only knowledge it has on the classifier is the prediction probability $\hat{y}$ of sent articles (black box). The quantity of adversarial articles takes precedence over the semantic and

syntactic constraints. The constraints will thus be softer and the preservation of the article’s meaning is not requisite. To meet these needs and conditions, we will built a fully automated algorithm.

1.1 Approach

This method follows the straightforward strategy of replacing one by one each word by similar words and observe how the classification changes. More exactly, it is an iterative method that considers at each step all valid word substitutions and chooses the one for which the prediction probability drops the most. It does this until the classification changes or until no word substitution lowers the prediction probability. The pseudo algorithm is in Algorithm 1.

Algorithm 1: Greedy algorithm

Data: Article A to modify

$A_{best} = A;$

while Prediction probability of $A_{adv} > 0.5$ **do**

$A_{adv} = A_{best};$

for each word w in A_{adv} **do**

for $syn \in \text{similar}(w)$ **do**

if $\text{correct_syntax}(A_{adv}, syn)$ **then**

$A_{temp} = \text{replace}(A_{adv}, syn);$

if Prediction probability of $A_{temp} < \text{Prediction probability of } A_{best}$ **then**

$A_{best} = A_{temp}$

 Stops if no word substitution were found

return A_{adv}

Two important functions in this algorithm need some further explanation:

similar(w): This function tackles the semantic constraints. It returns words that are semantically similar to the word w . As we saw in section 2.2.3, those words can be found with Word2Vec’s cosine similarity. To do this, we use google’s pre-trained **word2vec** [26]. However, sometimes similar words mean exactly the opposite, for example *Black* and *White* are cosine similar in the embedding space, same for *democrats* and *republicans*. Another idea would be to use NLTK’s WordNet database [27] where we can easily find synonyms for the word w , we can then avoid **word2vec**’s problem of getting antonyms. However, it still doesn’t settle the semantic correctness problem since words can have different meanings depending on their context. Because we assume a soft semantic constraint, we will use **word2vec** in the validation.

correct_syntax(A,syn): This function verifies the syntactic correctness of the replacement. Indeed, replacing a word by a similar word is not always syntactically correct. For example replacing *came* by *gone* or *install* to *installing* can lead to some weird sentences. To try to prevent this, we will use a bigram language model. A language model learns to predict the probability of a sequence of words. In a bigram model, the probability is simply $P(w_k|w_{k-1})$, which is found by a statistical study of the available documents. The probability of *to eating* will thus be very low which prevents the case of substituting *eat* to *eating*. Since it learns the probabilities on given documents, those documents have a great impact on the language model. The language model used here learned from the documents

of the test set since those are the available documents in black box. A bigram only watches a word and its neighbours but a word can depend on words further than the direct neighbours. The problem is that to make a N-gram, the bigger the N, the more data is needed, and the available documents are here too scarce.

The while loop stops when the prediction probability of the current adversarial article is lower than 0.5, i.e. when the modified article is classified as reliable by the classifier. When no more substitution can lower the prediction probability, the algorithm stops and it is estimated that it is not possible to make an adversarial example by replacement only. To make sure the final article doesn't differ too much from the original, we can add the condition in the while loop that no more than 5% of the words are modified.

This greedy algorithm has some major limitations.

- It can only replace words, there is no adding nor removing.
- It may take a lot of time, especially for large articles with a high prediction probability. At each iteration, the algorithm has to go through a lot of words and since the confidence is higher, the number of iteration will probably be bigger.
- It doesn't control well the semantics or syntax which makes inappropriate if the constraints are stronger.

1.2 Example

Let's try this on our example when the classifier is the logistic regression.

...various health scares and FBI investigations , Mrs Clinton's...	→	...various health scares and FBI inquiry , Mrs Clinton's...
... sleepover for a group of Girl Guides. Short of dressing...	→	... sleepover for a group of Woman Guides. Short of dressing...
...beloved US symbols, including punching a bi- son , setting fire...	→	...beloved US symbols, including punching a buffalo , setting fire...

Example 10: Greedy algorithm on logistic regression

We see in Example 10 that it only has to modify three words to drop the confidence from 89.8% to 47.58%. The syntax is correct except for the first modification where inquiry is singular which makes the sentence a little bit weird. The semantics is a more subjective part but overall the adversarial article makes sense and the message stays the same.

While the algorithm is pretty convincing for the logistic regression, making a drop of 42% of confidence with only three reasonable alteration, the result are less satisfying trying it on the LSTM as we can see in Example 19. Four modifications are made but neither the syntax nor the semantics make sense. Even if we can not generalize this observation, it is worth precising that if we prohibit the replacement of *hillary* and *mrs*, no adversarial example is found.

...her own campaign, Hillary Clinton has dese-	→	...her own campaign, Gop Clinton has dese-
crated...		crated...
... White House lawn. Hillary's erratic be-	→	... White House lawn. Gop's erratic be-
haviour...		haviour...
...and FBI investigations, Mrs Clinton's cam-	→	...and FBI investigations, Elizabeth Clinton's
paign...		campaign...
...one journalist: 'The Presidential race is en-		...one journalist: 'The Presidency race is en-
tering the...		tering the...

Example 11: Greedy algorithm on LSTM

1.3 Validation

To evaluate the greedy algorithm on a Logistic Regression classifier, we use the algorithm on 100 fake articles classified as originally fake by the classifier. Of the 100 articles, 32 were successfully transformed into adversarial examples. The average number of words modified in those 32 adversarial examples is 27 words. The article having the highest original $\hat{y}$ transformed to adversarial article was labelled as was fake with 99.9992%. As we can see in Table 12, all the articles under 95% were successfully transformed.

Original $\hat{y}$	
<95%	11 of the 11 articles were successfully transformed
95% < <99%	5 of the 10 articles were successfully transformed
>99%	16 of the 79 articles were successfully transformed

Table 12: Distribution of adversarial example over original $\hat{y}$ for logistic regression

The same test is done on LSTM on 100 articles. Of the 100 articles, 22 were transformed into adversarial examples. The average number of words modified in these 22 articles is 6 words. The article with the lowest initial confidence that was not successfully transformed was initially labeled with 61.4% of confidence to be fake. As we can see in Table 13, only one article with original $\hat{y} > 99\%$ was successfully transformed.

Original $\hat{y}$	
<95%	18 of the 27 articles were successfully transformed
95% < <99%	3 of the 13 articles were successfully transformed
>99%	1 of the 60 articles were successfully transformed

Table 13: Distribution of adversarial example over original $\hat{y}$ for LSTM

1.4 Conclusion

We thus find that the LSTM classifier is more robust to the greedy algorithm than the Logistic Regression, even though the Logistic Regression had higher confidence in the classification of the original articles. This is not surprising, the greedy algorithm is better fitted to attack a model for which the word order has no importance, since the only modification it can do in the text is replacing words one by one. Another reason may be the fact that the greedy algorithm relies on the Word2Vec

model to find similar words which is precisely the model used for the word embedding by the LSTM. We can suggest a possible consequence of that. The similar words are chosen by the greedy algorithm due to their similar embedding. If the embedding is similar between two words, the replacement of one by the other will just slightly modify the embedding of the text and thus the output of the LSTM will only slightly vary. However, the influence of this is not clear, because the Word2Vec models used in the greedy algorithm is not the same as in the LSTM. We also observe that the average number of words modified in Logistic Regression is higher. This is because LSTM only takes the first 200 words into account which reduces the needed substitutions.

2 Strong constraints with low production rate - White Box

In this second situation, we assume that the adversary wants a few coherent adversarial examples that keep the same meaning as the original articles. Hence, the semantic and syntactic constraints are strong and the idea of a fully automated algorithm is dropped. This strategy relies thus on a human intervention in the modification of the article, the adversary is assisted by the machine to know where and what to modify. In this section, the adversary has a white box knowledge on the targeted classifier, we will take advantage of that by examining the gradient of the loss function with respect to the inputs. The main ideas are inspired by [28].

2.1 Approach

An usual way to built a white box adversarial example is the Fast Gradient Sign Method (FGSM). The method uses the gradient of the loss function with respect to the input to create an adversarial example that maximizes the loss function.

$$\begin{aligned}\Delta x &= \epsilon \operatorname{sign}(\nabla_x J(F, x, y)) \\ x_a &= x + \Delta x\end{aligned}$$

with

x	The input
x_a	The adversarial input
ϵ	Has to be small enough to make perturbation Δx undetectable
F	The classifier
y	The target values i.e. the class

FGSM works well for image classification or similar continuous problems for which the imperceptibility constraint is respected by using a small ϵ . But FGSM is not directly applicable in the context of natural language. First, because other and more complex constraints apply. But more importantly because text is discrete. Depending on the vectorization method in the preprocessing part, adding a word could completely change the vectorization of the text. We will thus customize the FGSM for our problem and since the vectorization method used in the two targeted classifiers are fundamentally different, the white box strategy for LSTM will differ from the white box strategy for Logistic Regression.

Nonetheless, the two adversarial methods rely on the same idea and have thus the same structure:

1. Identify Reliable Hot Words (RHW) over the whole available collection of documents. Those words are supposed to influence the classification towards the reliable class

2. Identify the Article's Fake Hot Words (AFHW) specific to the targeted article. Those words influence the current classification towards the fake class.
3. Human intervention to replace AFHW by RHW in a semantic preserving and syntactic correct way.

2.1.1 Logistic Regression

The first important information is that the word order is of no importance for the classification. Indeed, the vectorized input vector fed to the Logistic Regression is a vocabulary with an increment or decrement in one dimension when we respectively add or remove the corresponding word independently of the position. From a FGSM-strategy point of view, we only need to compute the gradient of each input for the targeted article and then accordingly add or remove specific words no matter the position.

The gradient of a tfidf - logistic regression model can be computed analytically. Here are the equations we know about the model with x the bag of words of the document d :

$$\begin{aligned}
 z &= wP \frac{x}{n_d} + b \\
 \hat{y} = a = \sigma(z) &= \frac{1}{1 + e^{-z}} \\
 L(\hat{y}, y) &= -y * \log(\hat{y}) - (1 - y) * \log(1 - \hat{y})
 \end{aligned}$$

With P being a diagonal matrix with the idf values on the diagonal and n_d the number of words in document d ($\frac{x}{n_d}$ is thus the term frequency tf). We want to compute the gradient of the loss function with respect to the inputs x :

$$\frac{\delta L}{\delta x_1} = \frac{\delta L}{\delta a} * \frac{\delta a}{\delta z} * \frac{\delta z}{\delta x_1}$$

Which each give:

$$\begin{aligned}
 \frac{\delta L}{\delta a} &= \left(\frac{-y}{a} \right) + \left(\frac{1-y}{1-a} \right) \\
 \frac{\delta a}{\delta z} &= a * (1 - a) \\
 \frac{\delta z}{\delta x_1} &= \frac{P_{11}}{n_d} * w_1
 \end{aligned}$$

Putting it together:

$$\begin{aligned}
 \frac{\delta L}{\delta x_1} &= \left(\left(\frac{-y}{a} + \frac{1-y}{1-a} \right) * a * (1 - a) \right) * \frac{P_{11}}{n_d} * w_1 \\
 &= (a - y) * \frac{P_{11}}{n_d} * w_1 \\
 \frac{\delta L}{\delta x} &= \frac{(a - y)}{n_d} * Pw
 \end{aligned}$$

The relative magnitudes of the gradient of each input is independent of n_d and $(a - y)$, however the sign $(a - y)$ is important to know the role of a word. Since $a \in [0, 1]$ and $y = 1$ for articles labeled as fake, the words corresponding to the most negative product $P_{ii} * w_i$ will increase the loss function the most and are thus Reliable Hot Words. Alternatively, words with the biggest $P_{ii} * w_i$ are Fake Hot

Words. Since the gradient does not depend on the input, the occurring Fake Hot Words in the article are the Article’s Fake Hot Words.

The adversary will then try to manually remove, replace or paraphrase AFHW by RHW.

2.1.2 LSTM

Contrary to the logistic regression, having a white box knowledge of a LSTM algorithm doesn’t mean having a perfect understanding of the classifier. The importance of patterns and the relation between the inputs are highly more intricate. The foundation of our strategy is an observation made in [29]. In this article, it is shown that in image classification with neural networks, the inputs with large gradient magnitude often contribute more to the current prediction. Knowing this, they successfully provide adversarial examples by only modifying those inputs. The identification of RHW and AFHW in the white box case will rely on this observation.

Before starting to design an adversarial strategy, we first try to understand the relationship between the Article’s Hot Word and its gradient. Hence, we have to compute the gradient of the loss function with respect to the inputs, i.e. each element of each word’s embedding. An analytical equation of this gradient is hard, we will thus work with a numerical estimation of the gradient specific to each article⁵. The method computes a gradient with the shape of the input i.e. 200x300 which is thus 300 values for each word. This makes the obtained gradient hard to decipher. Table 14 explores some basic statistics on the gradient of a random article. We see that the word with the maximum gradient has also the minimum gradient. It is thus difficult to deduct the role of a word based on the signs of its gradients, like we did for the Logistic Regression. However, we can deduct the importance of the words for the classification based on the magnitude of its gradient, according to the key observation discussed earlier.

	Ranking by maximum gradient	Rank by minimum gradient	Ranking by mean of gradient magnitudes
1	<i>stood</i>	<i>stood</i>	<i>stood</i>
2	<i>rock</i>	<i>rock</i>	<i>rock</i>
3	<i>return</i>	<i>return</i>	<i>return</i>

Table 14: Study of the gradients of each word in the selected article

Knowing this, we can try to identify the RHW for the corpus. The algorithm to achieve this is composed in two parts. First, we iterate over all the available articles classified as reliable and we keep the word with the maximum gradient in absolute value (and words with maximum absolute gradient greater than 98% that gradient). Once those hot words are collected, we rank the words based on their occurrence counts. We then proceed the same way to find the Fake Hot Words in the articles classified as fake. The problem is that FHW often appear with big gradients in articles classified as reliable. Hence, they have a high score as both RHW and FHW. To prevent this, we need a second part for the RHW identification.

⁵computed with `tensorflow.gradientTape`

We suppose that FHW appear more often in articles classified as fake. Assuming this, we compute for each word the relative occurrences in both lists and consider it as a hot word for the class in which it relatively appears the most.

Now we know which words influence the classification to the reliable class. But we don't know where to insert those words in an article to make an adversarial example. Indeed, the consequence of adding, removing or modifying a word is far more complex than for the logistic regression. For example, inserting (without replacing) *trump* in our main example in the fourth position of the article increases the prediction probability to 0.72 while appending *trump* to the end of the text makes the prediction probability drop to 0.50. We thus need to find the Article's Fake Hot Words.

The strategy to find the AFHW is also based on the maximum gradient. We identify the 15 words with the highest absolute gradient. Those words are supposed to influence a lot the prediction probability and therefore, if modified, they could have a big impact on the confidence of the classification. The problem is that we don't know in which direction it influences the classification. To find this out, we remove them one by one and observe how the classifier reacts. Since LSTM takes into account the context of a word, if neighbouring words appear in the 15 hot words, we will remove them together and study the results. The word removals that decreases the prediction probability are the AFHW.

Knowing this, the adversary can now modify manually the article. The attacker computes the AFHW of the modified article after each modification and continues the modification accordingly.

2.2 Results and Example

2.2.1 Logistic Regression

The first step consists to find the Reliable Hot Words. The top 10 RHW and FHW for logistic regression are given in Table 15.

RHW		FHW	
1	guardian	1	gop
2	said	2	est
3	mrs	3	tremor
4	zealand	4	sen
5	flore	5	parker
6	snowflak	6	rep
7	chaplain	7	trunew
8	chester	8	kany
9	woodley	9	dgc
10	rightw	10	aca

Table 15: Hot words found with the white box strategy for logistic regression

A couple of remarks to better understand the results:

1. gop stands for Grand Old Party and is the nickname of the Republican Party.

2. est stands probably for Eastern Standard Time
3. kany is the stemming of Kanye
4. rightw is the stemming of rightwing
5. guardian appears often in articles of the guardians.
6. trunew comes from trunews and is a media outlet.

Now that we have the Reliable Hot Words, we easily find the Article’s Fake Hot Words based on the FHW. The biggest AFHW are *advisor* and then *hillary* (Found in full FHW list in B.3). Let’s try to replace them by a RHW. We thus replace *advisor* by *minister*. This makes the prediction probability $\hat{y}$ drop to 0.63. We can then replace *hillary* by *mrs*. This drops $\hat{y}$ to 0.21.

...Mount Rushmore. Remarked a flummoxed	→	...Mount Rushmore. Remarked a flummoxed
advisor : ‘She keeps...		minister : ‘She keeps...
... sabotage her own campaign, Hillary Clinton	→	... sabotage her own campaign, Mrs Clinton
has desecrated...		has desecrated...

Example 16: Article modifications with logistic regression

We see that only two well thought replacements were sufficient to decrease the prediction probability of 0.687.

2.2.2 LSTM

Table 17 gives the RHW and FHW found on the corpus. The problem of the presence of FHW as RHW is not completely solved. Indeed, after some text modifications, we will quickly notice that *report* and *comment* are FHW. We just need to change their label to solve the problem. Those bad labelizations are in most adversary’s situations not a major drawback.

RHW		FHW	
1	said	1	novemb
2	report	2	octob
3	told	3	news
4	donald	4	post
5	comment	5	say
6	trump	6	articl
7	say	7	share
8	campaign	8	publish
9	interview	9	video
10	debat	10	sourc

Table 17: Hot Words found with the White Box algorithm for LSTM

We now want to find the AFHW of our main example. The first step is to identify the words with the highest gradient:

	share	stori	comment	agre	campaign		journalist	sing	elect	star	sabotag	host
Index:	105	106	37	24	4	107	38	86	104	87	3	25

They are ordered from higher to lower maximum gradient and the number under each word is its position in the text. We observe that the 3 last words and the vacant space at the end of the article have a great impact. To know if the impact is positive or negative we proceed to the removals of each word or sequence of consecutive words. In Table 18, the 5 biggest AFHW are given.

share stori	-0.269
elect share stori	-0.227
share	-0.189
elect share	-0.138
sabotage	-0.024

Table 18: AFHW of main article

As we can see, removing the last sentence is sufficient, dropping the prediction probability to 0.379 (Example 25).

...Donald Trump electable'. Share this story... → ...Donald Trump electable'.
--

Example 19: Article modifications with logistic regression

If we don't want to loose the message of the last sentence, we can simply replace *share* by *send* (drops to 0.439) and if we want to take advantage of the influence of adding a word (at index 107) we add the word *friend* (Example 20).

...Donald Trump electable'. Share this story... → ...Donald Trump electable'. Send this story to your friends...
--

Example 20: Article modifications with logistic regression

The prediction probability drops then to 0.377.

2.3 Validation

To evaluate these strategies, we will try to make 5 adversarial examples from 5 randomly chosen fake articles in the test set that are correctly classified by both models. This small number of validation samples is due to the time consuming nature of the method. One intervention is equivalent to adding, modifying or removing a word or a sequence of consecutive words. We stop after 15 interventions estimating that the adversary is not willing to spend more time in the making of the adversarial article. We can see the results Table 21. We have to be cautious analyzing the results since the human intervention in the algorithm is very aleatory and the small number of targeted articles may not be a good representation of reality.

	text size	Logistic Regression			LSTM		
		Initial $\hat{y}$	Interventions	Final $\hat{y}$	Final $\hat{y}$	Interventions	Final $\hat{y}$
Article 1	1100	0.99997	15	0.996	0.9998	15	0.983
Article 2	1279	0.9996	15	0.97	0.70	1	0.44
Article 3	295	0.989	7	0.4	0.996	4	0.36
Article 4	1280	0.998	15	0.79	0.82	5	0.40
Article 5	162	0.9999	15	0.92	0.9995	15	0.985

Table 21: Results for logistic regression

We can immediately notice the high number of interventions needed to make an adversarial example in the Logistic Regression. There is a direct correlation between the text size and the number of interventions and between the original $\hat{y}$ and the number of interventions. Both relations are deductible from the equation of the gradients found in section 2.1.1 in Chapter II:

$$\frac{\delta L}{\delta x} = \frac{(\hat{y} - y)}{n_d} * Pw$$

The gradients are lower when $\hat{y}$ is close to 1 and when n_d is large which means a lower impact when modifying the input.

LSTM is generally fooled after fewer interventions. A first reason for this is the lower initial $\hat{y}$. A second reason is that LSTM only takes the first 200 words⁶ into account, all the used information is thus concentrated in fewer words, hence, modifying them will have greater impact. In particular, LSTM accords more quickly high importance to some patterns, for example in our main example the pair *share story*. Our white box algorithm takes advantage of this characteristic by examining the impact of removing those consecutive words jointly.

2.4 Conclusion

While our understanding of the logistic regression is great, it is still hard to fool. Two main reasons for this: its high confidence for classification of the original article and the reduced impact of the interventions when the article contains a lot of words. On the other hand, the understanding of LSTM is difficult, even in the white box situation. Still, we succeed to achieve satisfying results. Next to the lower initial confidence, another main reason is that it only takes into account the first 200 words, our trial and error strategy will thus be more effective.

Contrary to the previous scenario with softer constraints, the Logistic Regression seems here harder to fool. Indeed, even if we know exactly the best possible changes to make, the stronger constraints reduce the possibilities. Furthermore, we restricted ourselves to 15 interventions while the greedy algorithm made on average 27 substitutions to obtain an adversarial article. On the other hand, the possibility of modifying multiple words at once made it easier to fool the LSTM in this scenario.

3 Strong constraints with low production rate - Black Box

In this situation, the needs of the adversary are the same as in the previous chapter, i.e. the constructed adversarial examples need to be syntactically correct and need to preserve the meaning. But contrary

⁶after preprocessing

to the previous presented method, the adversary is now unaware of the classifiers structure. It only knows the prediction probability $\hat{y}$ of sent articles (black box). The strategy is thus the same independently of the targeted classifier.

3.1 Approach

In the same way as the white box algorithms, the strategy is structured in three parts, first identifying the RHW then identifying the AFHW followed by the manual text modifications.

The idea, shown in Algorithm 2, is to iterate over all the available documents classified by the targeted model as reliable, in each document it removes each word one by one and feeds the new article to the classifier. It then compares for each word the prediction probability of the modified text and the original text. The word whose removal increases the prediction probability the most is added to a list (as well as words with variation greater than 98% the greatest variation). The most occurring words in the list are considered as RHW. The same can be done to obtain the Fake Hot Word iterating over the documents classified as fake and collecting the words whose removal decreases the prediction probability the most.

Algorithm 2: Black Box

Data: Collection D of documents

D_{rel} = documents classified as reliable;

for *each* d *in* D **do**

for *each* word w *in* d **do**

$d_temp = d$ without word x ;

 Compute deviation in prediction probabilities of d_temp

$reliables \cup$ word with biggest deviation;

$RHW = \text{mostoccurringwords}(reliables)$;

return RHW

Once we have the RHW, we can proceed to the second article-specific step: the identification of Article's Fake Hot Words. This is made in a very similar way by generating test samples by occluding the words of the targeted article one by one. Like in the first step, we then compare the classification results with the results of the original article. The larger the decrease of the prediction probability, the more important the word for the fake classification. Those words are considered as AFHW.

In the final part of the process, the text will be manually modified by replacing, removing or paraphrasing the AFHW using RHW.

3.2 Results and example

3.2.1 Logistic Regression

As usual, we start by looking at the obtained RHW and FHW (full list in Appendix B.3). The numbers in parenthesis give, as a comparison, the rank of the word obtained with the optimal white box Hot Words identification and the far right column gives the number of times a word was considered as hot word.

Reliable Hot Words				Fake Hot Words			
1	said	(2)	498	1	sourc	(20)	155
2	guardian	(1)	96	2	hillari	(34)	104
3	mrs	(3)	30	3	novemb	(33)	94
4	debat	(22)	7	4	articl	(45)	47
5	zealand	(4)	6	5	massiv	(26)	39
6	age	(74)	5	6	nov	(12)	34
7	argu	(80)	4	7	oct	(16)	30
8	monday	(54)	3	8	gop	(1)	23
9	year	(190)	3	9	probe	(15)	23
10	tuesday	(85)	3	10	kid	(31)	19

Table 22: Hot Words found with the Black Box algorithm for logistic regression

The result for the RHW with the black box are satisfying, it has the same three first words as in the white box. The very specific and scarcely used words present in the top 10 RHW of the white box do not appear at all in the black box list. This doesn't matter that much since those rarely used words are hard to insert in a text and are thus difficult to use in our context.

The second step is finding the Fake Sample Hot Words. Table 23 shows the 10 words whose removal decreases the prediction probability the most. The index next to each words gives which occurrence of the word, 1 being the first occurrence:

Logistic regression		LSTM	
1	advisor	1	share
2	hillary_2	2	hillary_1
3	hillary_1	3	fbi
4	fbi	4	orderly
5	entering	5	health
6	commented	6	hillary_2
7	girl	7	scares
8	video	8	movie
9	sing	9	investigations
10	donald	10	rights

Table 23: Article's Fake Hot Words

We can now proceed to text modification. We try to replace AFHW by RHW and we come to the same conclusion as in the white box case: Replacing *advisor* by *minister* and *hillary* by *mrs*.

3.2.2 LSTM

The RHW and FHW found are given in Table 24.

RHW		Fake Hot Words	
1	said	1	novemb
2	donald	2	octob
3	year	3	media
4	guardian	4	post
5	presid	5	articl
6	minist	6	hillari
7	world	7	share
8	trump	8	report
9	campaign	9	read
10	north	10	comment

Table 24: Hot Words found with the Black Box algorithm for LSTM

Having the AFHW in table 23, we can now try to fool the LSTM. It is quiet hard to find replacement to *share* in the RHW list. We can thus start with replacing hillary (second AFHW) by mrs. This drops the prediction probability to 0.504. We can then replace the fourth AFHW *orderly* by a synonym *neat*, making sure the synonym is not a Fake Hot Word. The $\hat{y}$ drops then to 0.45.

...her own campaign, Hillary Clinton has dese-	→	...her own campaign, Mrs Clinton has dese-
crated...		crated...
...campaign is as orderly as a Marx Brothers...	→	...campaign is as neat as a Marx Brothers...

Example 25: Article modifications with logistic regression

3.3 Validation

For the analysis of the validation of the black box method, we have to keep in mind the author’s total knowledge of the targeted classifier. The results in Table 26 may thus be biased and for the same reason as in the white box case, we have to be careful when analyzing these results.

	text size	Logistic Regression			LSTM		
		Initial $\hat{y}$	Interventions	Final $\hat{y}$	Final $\hat{y}$	Interventions	Final $\hat{y}$
Article 1	1100	0.99997	15	0.9994	0.9998	15	0.994
Article 2	1279	0.9996	15	0.98	0.70	13	0.47
Article 3	295	0.989	9	0.47	0.996	8	0.43
Article 4	1280	0.998	15	0.92	0.82	14	0.48
Article 5	162	0.9999	15	0.96	0.9995	15	0.997

Table 26: Results for logistic regression

As expected, making an adversarial attack is less effective compared with the white box case. A big limitation is that the RHW lists are significantly smaller reducing the obvious modifications. The adversary will then have to modify words with synonyms by trial and error. The increase in the number of needed interventions is especially higher for LSTM. The adversary has here indeed no information about the effect of removing multiple consecutive Hot Words at once.

3.4 Conclusion

As we have seen with the main example, the adversarial examples on a logistic regression with white box and black box are very similar. In fact, the AFHW obtained with both methods will always be exactly the same. The difference between the two methods is that although the RHW list in the black box contains only influential words, it is significantly smaller than the RHW list obtained with white box. Hence, the adversary has less obvious possibilities when modifying the text and will have to try to substitute AFHW with synonyms, not knowing in advance the influence on the classification.

We observe some strong similarity between the black box and white box algorithm: relying on occluding words to study their influence on the classification. Yet, there are two major differences. The first one is the time-efficiency: finding RHW with black box on LSTM requires a consequent amount of time because it will have to classify the article as many times as there are words in the article⁷. The comparison of the time to find RHW based on 200 articles⁸ is given in Table 27. The same problem occurs when computing AFHW.

	Black Box	White Box
Time	17h48min	5min46

Table 27: Time to compute RHW for LSTM based on 200 articles

Secondly, the white box strategy analyzes the effect of the removal of a sequence of words to find the AFHW, which is more consistent with the context property of the LSTM.

4 Transferability

In some cases, adversarial examples can be transferred across classifiers. A black box model can then be fooled by an adversarial article constructed against another classifier, possibly in white box. This interesting feature of transferability is analyzed in this section: can the obtained adversarial examples also attack the classifiers constructed in the previous chapter?

Studying the transferability of the adversarial example requires a lot of adversarial samples. Since our second approach is not designed to produce a lot of adversarial articles and is therefore significantly more time consuming, we will solely study the transferability on adversarial articles constructed with the greedy algorithm.

In Table 28, we observe the reactions of the classifiers to the 32 adversarial examples obtained with the greedy strategy on Logistic Regression. As we may have expected, the adversarial examples made for Logistic Regression with unigrams are the most transferable to the Logistic Regression with unigrams and bigrams, the classifiers work fundamentally in the same way with a slightly modified vocabulary. The worst transferability is with the Naive Bayes where no article became an adversarial example after modification. We also observe that the modifications in general decrease the prediction probabilities. As mentioned in Chapter IV, the prediction probability of Naive Bayes is by nature often extreme, i.e. close to 0 or to 1. It is thus difficult to draw conclusion from the magnitude of change. How-

⁷This problem is present but in far lesser proportions with logistic regression

⁸Computed on a computer with 4 core, 2.60 GHz

	Original text classified as reliable	Adv. Ex. classified as reliable	Adv. Ex. classified as fake	Mean difference of $\hat{y}_{adv} - \hat{y}_o$	Articles with $\hat{y}_{adv} - \hat{y}_o > 0$
Log. Reg. 2	1	15	17	-0.40	1
Naive Bayes	0	0	32	-0.002	0
Random Forest	2	3	29	-0.05	3
LSTM	2	6	26	-0.13	0

Table 28: Transferability for adversarial articles made with greedy algorithm on Logistic Regression

	Original text classified as reliable	Adv. Ex. classified as reliable	Adv. Ex. classified as fake	Mean difference of $\hat{y}_{adv} - \hat{y}_o$	Articles with $\hat{y}_{adv} - \hat{y}_o > 0$
Log. Reg. 1	2	2	20	-0.041	3
Log. Reg. 2	2	4	18	-0.043	3
Naive Bayes	0	0	22	$-5.2 \cdot 10^{-11}$	1
Random Forest	4	5	17	-0.019	6

Table 29: Transferability for adversarial articles made with greedy algorithm on LSTM

ever, the negative sign of $\hat{y}_{adv} - \hat{y}_o$ for Naive Bayes shows that the modifications influence towards a reliable classification. In lower proportions than for the Logistic Regression 2, there is a relatively high decrease in prediction probability for LSTM on average. Finally, we observe that for Logistic Regression 2 and Random Forest, some modifications increase the prediction probability, this may be an important parameter for the adversary.

We now do the same for the 22 adversarial articles made by the greedy algorithm on the LSTM in Figure 29. Considering the original classifications, no articles in Logistic Regression moved from a fake to reliable classification after modifications. This happened twice with Logistic Regression 2 and once with Random Forest. Hence, the relatively high transferability of Random Forest and the Logistic Regressions is mainly due to the classification of the original articles as reliable. The number of adversarial examples classified as reliable (2nd column) may thus differ significantly on a different or bigger test set.

We observe that, also on these 22 adversarial articles, the prediction probabilities decrease on average. But the decrease is lesser than with adversarials created against Logistic Regression. In particular, the prediction probabilities $\hat{y}$ of LSTM on adversarial articles against Logistic Regression has a decrease of 0.13 on average, in the inverse scenario, the $\hat{y}$ of Logistic Regression only decreases of 0.04 on average. As we saw during the validation of the greedy algorithm, LSTM modifies on average 21 words less than Logistic Regression to built an adversarial article, due to the fact that it only takes the first 200 words into account. This is probably the main reason why the transferability of adversarial articles for LSTM is lower compared to the transferability of adversarial articles for Logistic Regression: because the greedy algorithm for LSTM modifies less the original document. Another factor may also be relied to the classification strategy itself. Indeed, as we can see in the results in Table 29, for significantly more articles there is an increase in their prediction scores after modification compared to Table 28. In particular, for Random Forest, for 6 of the 22 articles the confidence towards falsehood increased.

This study of transferability solely applies for adversarial examples constructed with our greedy algo-

rithm. Adversarial examples obtained by other means may present other transferability characteristics. The above analysis of transferability is only a brief study, deeper analysis is left for further work.

5 Further research

Our results are strongly impacted by the fact that LSTM only considers the first 200 words. An analysis on a LSTM taking all the words into account without needing padding could generate different results. Unfortunately, implementing such a classifier is difficult.

An important drawback to both our first and second approach is the lack of consideration for important consecutive words sequences. In the greedy algorithm, a solution would be to go over all the possible sequences or all the targeted sequences of maximum N words and use word2vec to find similar N-grams. In the second approach, the existence of multiple word patterns is a bit taken into account in the White Box scheme for LSTM when searching Samples Hot Words. This is done by observing the impact of removing specific consecutive words together. The same idea could be used to find the RHW of the whole database by analyzing the effect of removing jointly neighbouring RHW. This technique may require additional sample articles.

Part VI

Conclusion

The aim of this document was to study the robustness of fake news classifiers against articles built specifically to fool the classification. The first step was to find a database. Since we encountered no existing database that met our needs, we built our own based on an already existing database of fake news. While it has some imperfections, the exploration showed that it was of sufficient quality for our requirements. We then implemented 5 different classifiers, and chose two of them on which we will focus our adversarial attacks. We chose the Logistic Regression and the LSTM because of their good performance and because they work in inherently different ways. The first one vectorizes the article with tf-idf before applying the logistic regression and the second one embeds the first 200 words with a Word2Vec model and then feeds it into a bidirectional LSTM. Once we had our database and our classifiers, we could start the design of adversarial attacks.

Since the design of an adversarial attack depends on the goals and capabilities of the adversary, we considered 3 different situations. In the first situation, the adversary needs to produce a lot of adversarial examples without being too strict on the semantic and syntactic constraint. We considered that the adversary had a black box knowledge of the classifier. To meet the needs of the attacker, we proposed a fully automated greedy algorithm.

In the second and third situation, the adversary wants to produce coherent adversarial articles that preserve the meaning of the original text. The attacker does this with white box knowledge (second situation) and then black box (third situation). Both strategies are similar and consists in three steps. First, the adversary identifies the Reliable Hot Words (RHW) over all the corpus. Those are the words that influence the classification towards the reliable class. The second step is to find the words in the targeted article that influence its classification towards the fake class. Those words are called the Article's Fake Hot Words (AFHW). The final step requires human intervention. The adversary replaces AFHW by RHW in order to modify the text such that it stays syntactically and semantically correct. The validation showed that the resulting adversarial examples were similar from white box to black box. Table 30 shows the result of the validation of our three strategies.

	Logistic Regression	LSTM
Greedy algorithm	32/100	22/100
Human intervention with white box	1/5	3/5
Human intervention with black box	1/5	3/5

Table 30: Validation results

As we can see, we could not successfully transform more than 1/3 of the targeted articles classified by the Logistic Regression. To fool the Logistic Regression, it often requires a lot of modification, this is why it is more exposed to attacks with low semantic constraints. The LSTM was more robust to the greedy algorithm, partly because the naive algorithm ignores the importance of word sequences. On the other hand, LSTM was particularly vulnerable to attacks with human intervention. With all the used information concentrated in the first 200 words, it was easier to make impactful modifications. In

that sense, the LSTM could be improved by taking every word of an article into account. This requires a complicated implementation.

Finally, we briefly explored the transferability of the constructed adversarial examples to the other classifiers we built. Because a large amount of adversarial articles was needed to analyse the transferability, we restricted the study to adversarial articles constructed with the greedy algorithm. It showed that the classifiers have a relatively low transferability between each other except for between Logistic Regression with unigrams and Logistic Regression with unigrams with bigrams, nearly half of the adversarial examples fooled the second classifier. We also noticed the zero transferability for Naive Bayes.

As a final conclusion, the two studied classifiers have proven to be relatively robust, especially the Logistic Regression which, even if we understand entirely its working, is hard to fool.

References

- [1] Timothy McLaughlin, Wired. How WhatsApp Fuels Fake News and Violence in India. <https://www.wired.com/story/how-whatsapp-fuels-fake-news-and-violence-in-india/>, December 2020. Accessed: 03-06-2020.
- [2] Hannah Jane Parkinson. Click and elect: how fake news helped donald trump win a real election. *The New York Times*, Novembre 2016.
- [3] Theresa Waldrop, Dave Alsup and Elliott C. McLaughlin, CNN. Majority of Americans were not exposed to 'fake news' in 2016 U.S. election, Twitter study suggests. <https://www.sciencemag.org/news/2019/01/majority-americans-were-not-exposed-fake-news-2016-us-election-twitter-study-suggests>, January 2019. Accessed: 03-06-2020.
- [4] What is "fake news"? <https://30secondes.org/en/module/what-is-fake-news/>, 2019. Accessed: 03-06-2020.
- [5] Laura Chinchilla. Post-truth politics afflicts the global south, too. *The New York Times*, October 2019.
- [6] Jacob Soll, Politico. The Long and Brutal History of Fake News. <https://www.politico.com/magazine/story/2016/12/fake-news-history-long-violent-214535>, December 2016. Accessed: 03-06-2020.
- [7] Jon Henley, The Guardian. How Finland starts its fight against fake news in primary schools. <https://www.theguardian.com/world/2020/jan/28/fact-from-fiction-finlands-new-lessons-in-combating-fake-news>, January 2020. Accessed: 03-06-2020.
- [8] Michael-Ross Fiorentino, Euronews. France passes controversial 'fake news' law. <https://www.euronews.com/2018/11/22/france-passes-controversial-fake-news-law>, November 2018. Accessed: 03-06-2020.
- [9] Wired. Facebook Is Changing News Feed (Again) to Stop Fake News. <https://www.wired.com/story/facebook-click-gap-news-feed-changes/>, October 2019. Accessed: 03-06-2020.
- [10] BBC. Coronavirus: Twitter will label Covid-19 fake news. <https://www.bbc.com/news/technology-52632909>, May 2020. Accessed: 03-06-2020.
- [11] Victoria L. Rubin Nadia K. Conroy and Yimin Chen. Automatic Deception Detection: Methods for Finding Fake News. February 2016. <https://doi.org/10.1002/pr2.2015.145052010082>.
- [12] Volodymyr Mesyura Mykhailo Granik. Fake News Detection Using Naive Bayes Classifier. *IEEE First Ukraine Conference on Electrical and Computer Engineering*, pages 900–903, 2017.
- [13] Laurens van der Maaten and Geoffrey Hinton. Visualizing data using t-SNE. *Journal of Machine Learning Research*, 9:2579–2605, 2008.

- [14] Tomas Mikolov, Ilya Sutskever, Kai Chen, Greg Corrado, and Jeffrey Dean. Distributed Representations of Words and Phrases and their Compositionality. *arXiv e-prints*, page arXiv:1310.4546, October 2013.
- [15] Dan Jurafsky and James H. Martin. *Speech and Language Processing*. Third edition draft edition, October 2019.
- [16] Tony Yiu, Towards data science. Understanding Random Forest. <https://towardsdatascience.com/understanding-random-forest-58381e0602d2>, June 2019. Accessed: 20-05-2020.
- [17] Hochreiter Sepp and Schmidhuber Jürgen. Long Short-Term Memory. *Neural Computation*, 9(8):1735-1780, 1997.
- [18] Mahendran Venkatachalam , Towards data science. Recurrent Neural Networks. <https://towardsdatascience.com/recurrent-neural-networks-d4642c9bc7ce>, March 2019. Accessed: 20-05-2020.
- [19] Christian Szegedy, Wojciech Zaremba, Ilya Sutskever, Joan Bruna, Dumitru Erhan, Ian Goodfellow, and Rob Fergus. Intriguing properties of neural networks. *arXiv e-prints*, page arXiv:1312.6199, December 2013.
- [20] Anonymous authors. Adversarial examples for natural language classification problems. *Conference paper at ICLR*, 2018.
- [21] Ian Goodfellow, Patrick McDaniel, and Nicolas Papernot. Making machine learning robust against adversarial inputs. *Communications of the ACM*, pages 56–66, June 2018. doi: 10.1145/3134599. url: <http://doi.acm.org/10.1145/3134599>.
- [22] several27. Fake News Corpus. <https://github.com/several27/FakeNewsCorpus>, January 2020. Accessed: 3-03-2020.
- [23] Meg Risdal. Getting real about fake news. <https://www.kaggle.com/mrisdal/fake-news>, Nov 2016. Accessed on 15-02-2020.
- [24] Google. Word2Vec, Tool for computing continuous distributed representations of words. <https://code.google.com/archive/p/word2vec/>, July 2013. Accessed: 25-03-2020.
- [25] Kevin Patel and Pushpak Bhattacharyya. Towards lower bounds on number of dimensions for word embeddings. In *Proceedings of the Eighth International Joint Conference on Natural Language Processing (Volume 2: Short Papers)*, pages 31–36, Taipei, Taiwan, November 2017. Asian Federation of Natural Language Processing.
- [26] Tomas Mikolov, Kai Chen, Greg Corrado, and Jeffrey Dean. Efficient Estimation of Word Representations in Vector Space. *arXiv e-prints*, page arXiv:1301.3781, January 2013.
- [27] George A. Miller. Wordnet: A lexical database for english. *Commun. ACM*, 38(11):39–41, November 1995.
- [28] Bin Liang, Hongcheng Li, Miaoqiang Su, Pan Bian, Xirong Li, and Wenchang Shi. Deep Text Classification Can be Fooled. *arXiv e-prints*, page arXiv:1704.08006, April 2017.

- [29] Bin Liang, Hongcheng Li, Miaoqiang Su, Xirong Li, Wenchang Shi, and Xiaofeng Wang. Detecting Adversarial Image Examples in Deep Networks with Adaptive Noise Reduction. *arXiv e-prints*, page arXiv:1705.08378, May 2017.

A Parameter Tuning

A.1 Logistic regression

param_clf_C	param_tfidf_max_df	param_tfidf_min_df	param_tfidf_ngram_range	mean_test_score	rank_test_score ▲
500	0.5	20.0	(1, 2)	0.956961	1
500	0.6	20.0	(1, 2)	0.956607	2
50	0.5	20.0	(1, 2)	0.956253	3
50	0.6	20.0	(1, 2)	0.955899	4
500	0.85	20.0	(1, 2)	0.955013	5
50	0.85	20.0	(1, 2)	0.953419	6
500	0.6	0.01	(1, 2)	0.951825	7
500	0.6	50.0	(1, 2)	0.951825	8
50	0.6	0.01	(1, 2)	0.951471	9
50	0.5	0.01	(1, 2)	0.951471	9
500	0.85	0.01	(1, 2)	0.951471	11
50	0.6	50.0	(1, 2)	0.951294	12
500	0.85	50.0	(1, 2)	0.951294	13
50	0.5	50.0	(1, 2)	0.95094	14
50	0.85	50.0	(1, 2)	0.950585	15
50	0.85	0.01	(1, 2)	0.950585	16
500	0.5	0.01	(1, 2)	0.950408	17
500	0.5	50.0	(1, 2)	0.9497	18
500	0.85	20.0	(1, 1)	0.947398	19
50	0.85	20.0	(1, 1)	0.946866	20
50	0.85	0.01	(1, 1)	0.944209	21
50	0.85	50.0	(1, 1)	0.944032	22
50	0.6	20.0	(1, 1)	0.943323	23
500	0.6	20.0	(1, 1)	0.942968	24
500	0.85	0.01	(1, 1)	0.942792	25
50	0.5	20.0	(1, 1)	0.941906	26

param_clf_C	param_tfidf_max_df	param_tfidf_min_df	param_tfidf_ngram_range	mean_test_score	rank_test_score 
50	0.5	20.0	(1, 1)	0.941906	26
500	0.5	20.0	(1, 1)	0.941729	27
500	0.85	50.0	(1, 1)	0.941375	28
50	0.6	0.01	(1, 1)	0.940135	29
50	0.6	50.0	(1, 1)	0.93978	30
500	0.6	0.01	(1, 1)	0.938895	31
500	0.6	50.0	(1, 1)	0.937655	32
50	0.5	50.0	(1, 1)	0.937124	33
50	0.5	0.01	(1, 1)	0.93677	34
500	0.5	50.0	(1, 1)	0.936593	35
1	0.5	20.0	(1, 2)	0.936062	36
1	0.5	0.01	(1, 2)	0.935885	37
500	0.5	0.01	(1, 1)	0.93553	38
1	0.6	20.0	(1, 2)	0.935177	39
1	0.6	50.0	(1, 2)	0.935176	40
1	0.5	50.0	(1, 2)	0.934999	41
1	0.6	0.01	(1, 2)	0.934468	42
1	0.85	20.0	(1, 2)	0.93252	43
1	0.85	0.01	(1, 2)	0.931988	44
1	0.85	50.0	(1, 2)	0.931102	45
1	0.85	20.0	(1, 1)	0.930925	46
1	0.85	0.01	(1, 1)	0.929686	47
1	0.85	50.0	(1, 1)	0.929508	48
1	0.6	0.01	(1, 1)	0.925435	49
1	0.5	0.01	(1, 1)	0.925258	50
1	0.6	50.0	(1, 1)	0.925081	51
1	0.5	20.0	(1, 1)	0.924904	52
1	0.5	50.0	(1, 1)	0.924195	53
1	0.6	20.0	(1, 1)	0.923841	54

Figure 13: Logistic Regression

A.2 Naive Bayes

param_clf_alpha	param_tfidf_max_df	param_tfidf_min_df	param_tfidf_ngram_range	mean_test_score	rank_test_score 
0.01	0.6	20	(1, 2)	0.844845	1
0.01	0.85	20	(1, 2)	0.842897	2
0.1	0.6	20	(1, 2)	0.842188	3
0.1	0.85	20	(1, 2)	0.84024	4
1	0.6	20	(1, 2)	0.834041	5
1	0.85	20	(1, 2)	0.833687	6
4	0.85	20	(1, 2)	0.818987	7
4	0.6	20	(1, 2)	0.818987	8
0.01	0.6	20	(1, 1)	0.816684	9
0.01	0.85	20	(1, 1)	0.816508	10
0.01	0.6	0.01	(1, 2)	0.816153	11
0.1	0.6	0.01	(1, 2)	0.815799	12
0.1	0.85	20	(1, 1)	0.815622	13
0.01	0.85	0.01	(1, 2)	0.815267	14
1	0.6	0.01	(1, 2)	0.814736	15
0.1	0.6	20	(1, 1)	0.814381	16
0.1	0.85	0.01	(1, 2)	0.814204	17
0.01	0.6	50	(1, 2)	0.813142	18
0.01	0.85	50	(1, 2)	0.812433	19
1	0.85	0.01	(1, 2)	0.812433	20
1	0.85	20	(1, 1)	0.812257	21
0.1	0.6	50	(1, 2)	0.812256	22
0.1	0.85	50	(1, 2)	0.812079	23
1	0.6	50	(1, 2)	0.810308	24
1	0.85	50	(1, 2)	0.810131	25

param_clf_alpha	param_tfidf_max_df	param_tfidf_min_df	param_tfidf_ngram_range	mean_test_score	rank_test_score
4	0.6	50	(1, 2)	0.80464	29
4	0.85	50	(1, 2)	0.80411	30
4	0.85	20	(1, 1)	0.800036	31
4	0.6	20	(1, 1)	0.799327	32
0.01	0.6	0.01	(1, 1)	0.794899	33
0.01	0.85	0.01	(1, 1)	0.794546	34
0.1	0.6	0.01	(1, 1)	0.794368	35
0.1	0.85	0.01	(1, 1)	0.794015	36
0.01	0.85	50	(1, 1)	0.792775	37
0.01	0.6	50	(1, 1)	0.792774	38
0.1	0.85	50	(1, 1)	0.79242	39
0.1	0.6	50	(1, 1)	0.792243	40
1	0.85	0.01	(1, 1)	0.791889	41
1	0.6	0.01	(1, 1)	0.791711	42
1	0.6	50	(1, 1)	0.790649	43
1	0.85	50	(1, 1)	0.789941	44
4	0.85	0.01	(1, 1)	0.786575	45
4	0.85	50	(1, 1)	0.786221	46
4	0.6	50	(1, 1)	0.78445	47
4	0.6	0.01	(1, 1)	0.784096	48

Figure 14: Naive Bayes

A.3 Random Forest

param_cif_criterion	param_cif_max_depth	param_cif_n_estimators	param_tfidf_max_df	param_tfidf_min_df	param_tfidf_ngram_range	mean_test_score	rank_test_score
gini	None	1000	0.6	20	(1, 2)	0.926851	1
entropy	None	1000	0.6	20	(1, 2)	0.924549	2
gini	500	1000	0.6	0.01	(1, 2)	0.924548	3
entropy	None	1000	0.6	0.01	(1, 2)	0.924372	4
gini	500	1000	0.6	20	(1, 2)	0.924195	5
gini	500	1000	0.85	0.01	(1, 2)	0.924016	6
entropy	500	1000	0.6	0.01	(1, 2)	0.923841	7
gini	None	1000	0.6	0.01	(1, 1)	0.922954	8
entropy	500	1000	0.6	20	(1, 2)	0.922777	9
gini	None	1000	0.6	0.01	(1, 2)	0.922777	10
gini	500	1000	0.6	0.01	(1, 1)	0.921891	11
gini	None	1000	0.85	0.01	(1, 2)	0.921538	12
entropy	500	1000	0.85	0.01	(1, 2)	0.920652	13
gini	None	1000	0.85	0.01	(1, 1)	0.919943	14
gini	500	1000	0.85	20	(1, 1)	0.919943	15
gini	500	1000	0.85	0.01	(1, 1)	0.919943	16
gini	None	1000	0.85	20	(1, 2)	0.91959	17
gini	500	1000	0.85	20	(1, 2)	0.919235	18
entropy	None	1000	0.85	20	(1, 1)	0.919235	19
entropy	500	1000	0.85	20	(1, 2)	0.919058	20
gini	None	1000	0.85	20	(1, 1)	0.918881	21
entropy	None	1000	0.85	20	(1, 2)	0.918881	21
entropy	None	1000	0.85	0.01	(1, 1)	0.918704	23
gini	500	100	0.85	0.01	(1, 1)	0.918173	24
gini	None	1000	0.6	20	(1, 1)	0.918172	25
entropy	None	1000	0.6	20	(1, 1)	0.917818	26
gini	500	1000	0.6	20	(1, 1)	0.917818	27
entropy	500	1000	0.85	0.01	(1, 1)	0.917465	28
entropy	None	1000	0.85	0.01	(1, 2)	0.917464	29
entropy	500	1000	0.85	20	(1, 1)	0.916933	30
gini	None	100	0.6	0.01	(1, 2)	0.916932	31
entropy	None	100	0.6	0.01	(1, 2)	0.916578	32
entropy	500	100	0.6	0.01	(1, 2)	0.916225	33
entropy	None	1000	0.6	0.01	(1, 1)	0.916223	34
gini	500	100	0.6	20	(1, 2)	0.915515	35
gini	500	100	0.85	20	(1, 2)	0.915339	36
entropy	500	1000	0.6	0.01	(1, 1)	0.91516	37
gini	500	100	0.85	0.01	(1, 2)	0.914983	38
entropy	500	100	0.85	0.01	(1, 2)	0.914808	39
gini	500	100	0.6	0.01	(1, 2)	0.91463	40
entropy	500	1000	0.6	20	(1, 1)	0.914275	41
entropy	500	100	0.6	20	(1, 2)	0.914099	42
gini	None	100	0.6	20	(1, 2)	0.913921	43
gini	None	100	0.85	20	(1, 2)	0.913744	44
entropy	500	100	0.85	0.01	(1, 1)	0.913391	45
entropy	None	100	0.85	0.01	(1, 1)	0.913213	46
gini	None	100	0.85	20	(1, 1)	0.912506	47
gini	500	100	0.6	0.01	(1, 1)	0.912505	48

Figure 15: Random forest

param_clf_criterion	param_clf_max_depth	param_clf_n_estimators	param_tfidf_max_df	param_tfidf_min_df	param_tfidf_ngram_range	mean_test_score	rank_test_score
gini	500	100	0.6	0.01	(1, 1)	0.912505	48
entropy	500	100	0.85	20	(1, 1)	0.912326	49
entropy	None	100	0.6	20	(1, 2)	0.911972	50
gini	None	100	0.85	0.01	(1, 2)	0.911796	51
entropy	None	100	0.85	0.01	(1, 2)	0.911619	52
entropy	None	100	0.85	20	(1, 2)	0.910735	53
entropy	None	100	0.85	20	(1, 1)	0.910733	54
gini	None	100	0.85	0.01	(1, 1)	0.910024	55
gini	None	100	0.6	0.01	(1, 1)	0.909848	56
entropy	500	100	0.85	20	(1, 2)	0.909847	57
gini	500	100	0.85	20	(1, 1)	0.908788	58
entropy	None	100	0.6	0.01	(1, 1)	0.908785	59
entropy	500	100	0.6	20	(1, 1)	0.906484	60
entropy	None	100	0.6	20	(1, 1)	0.905951	61
gini	None	100	0.6	20	(1, 1)	0.905774	62
gini	500	100	0.6	20	(1, 1)	0.905774	63
entropy	500	100	0.6	0.01	(1, 1)	0.90471	64

Figure 16: Random forest

A.4 LSTM

param_clf_neuronsDense	param_clf_neuronsLSTM	mean_test_score	rank_test_score
400	100	0.902136	1
200	400	0.901169	2
200	100	0.897272	3
200	200	0.893376	4
100	100	0.893022	5
400	200	0.891428	6
100	200	0.886291	7
400	400	0.88204	8
100	400	0.879207	9

Figure 17: Tuning NeuronsDense and NeuronsLSTM with batchsize=None and epoch=2

param_clf_batch_size	param_clf_epochs	mean_test_score	rank_test_score ▲
None	3	0.894439	1
None	2	0.889479	2
None	4	0.882217	3
400	4	0.854942	4
400	3	0.828905	5
400	2	0.823592	6

Figure 18: Tuning batchsize and epochs for neuronsDense=400 and neuronsLSTM=100

B Reliable Hot words

B.1 Black Box - Logistic regression

Index	word	size	Index	word	size
0	said	532	25	tibetan	1
1	guardian	106	26	remain	1
2	mrs	33	27	cairo	1
3	debat	17	28	stone	1
4	zealand	7	29	stanc	1
5	year	7	30	argument	1
6	argu	6	31	slogan	1
7	age	5	32	select	1
8	monday	4	33	clown	1
9	wrote	3	34	room	1
10	tuesday	3	35	tanden	1
11	suggest	3	36	elig	1
12	oxford	3	37	recount	1
13	minist	3	38	globalis	1
14	huge	3	39	guantánamo	1
15	affect	3	40	dish	1
16	latino	2	41	hamilton	1
17	wednesday	2	42	die	1
18	criticis	2	43	india	1
19	inquiri	2	44	irish	1
20	rightw	2	45	jersey	1
21	worri	1	46	join	1
22	walloon	1	47	defenc	1
23	batch	1	48	millenni	1
24	beij	1	49	danc	1

Figure 19: Reliable Hot Words

B.2 Black Box - LSTM

Index	word	size	Index	word	size
0	said	125	25	director	4
1	donald	48	26	senat	3
2	year	30	27	journal	3
3	guardian	27	28	role	3
4	presid	23	29	leader	3
5	minist	16	30	rais	3
6	world	15	31	night	3
7	trump	14	32	week	3
8	north	12	33	sunday	3
9	campaign	12	34	decad	3
10	wednesday	7	35	spokesman	2
11	debat	7	36	journalist	2
12	south	6	37	american	2
13	govern	6	38	professor	2
14	news	5	39	confer	2
15	univers	5	40	republican	2
16	polit	5	41	long	2
17	social	5	42	court	2
18	hour	5	43	kong	2
19	monday	4	44	york	2
20	newspap	4	45	franc	2
21	secretari	4	46	hong	2
22	email	4	47	thursday	1
23	mrs	4	48	tuesday	1
24	month	4	49	women	1

Figure 20: Reliable Hot Words

B.3 White Box Logistic regression

Inde ▲	Type	Size	Value	Inde ▲	Type	Size	Value
0	str	1	guardian	24	str	1	tanden
1	str	1	said	25	str	1	whale
2	str	1	mrs	26	str	1	servicemen
3	str	1	zealand	27	str	1	oxford
4	str	1	flore	28	str	1	castro
5	str	1	snowflak	29	str	1	criticis
6	str	1	chaplain	30	str	1	meme
7	str	1	chester	31	str	1	pluto
8	str	1	woodley	32	str	1	cairo
9	str	1	rightw	33	str	1	cathol
10	str	1	schwartz	34	str	1	clown
11	str	1	amp	35	str	1	spain
12	str	1	defenc	36	str	1	arafat
13	str	1	chick	37	str	1	del
14	str	1	guantánamo	38	str	1	guardianwit
15	str	1	globalis	39	str	1	bataclan
16	str	1	sikh	40	str	1	hawkin
17	str	1	inqui	41	str	1	feinstein
18	str	1	palmer	42	str	1	walloon
19	str	1	rat	43	str	1	millenni
20	str	1	slager	44	str	1	hersh
21	str	1	debat	45	str	1	monk
22	str	1	samsung	46	str	1	mustafa
23	str	1	flint	47	str	1	sunni

Figure 21: Reliable Hot Words

Inde ▲	Type	Size	Value	Inde ▲	Type	Size	Value
0	str	1	gop	24	str	1	dixon
1	str	1	est	25	str	1	massiv
2	str	1	tremor	26	str	1	reuter
3	str	1	sen	27	str	1	pirat
4	str	1	parker	28	str	1	print
5	str	1	rep	29	str	1	jungl
6	str	1	trunew	30	str	1	kid
7	str	1	kany	31	str	1	hostel
8	str	1	dcg	32	str	1	novemb
9	str	1	aca	33	str	1	hillari
10	str	1	getti	34	str	1	buffer
11	str	1	nov	35	str	1	kilomet
12	str	1	advisor	36	str	1	tpp
13	str	1	franconi	37	str	1	allah
14	str	1	probe	38	str	1	yahoo
15	str	1	oct	39	str	1	photo
16	str	1	url	40	str	1	looni
17	str	1	courtesi	41	str	1	soro
18	str	1	ceta	42	str	1	cohen
19	str	1	sourc	43	str	1	malik
20	str	1	ayott	44	str	1	articl
21	str	1	salam	45	str	1	rinf
22	str	1	tmz	46	str	1	witchcraft
23	str	1	gabbard	47	str	1	sil

Figure 22: Fake Hot Words

B.4 White box - LSTM

Index	word	size	relative	Index	word	size	relative
0	said	272	0.0226837	25	year	34	0.00283546
1	report	155	0.0129264	26	group	33	0.00275206
2	told	100	0.00833959	27	base	32	0.00266867
3	donald	71	0.00592111	28	state	30	0.00250188
4	comment	68	0.00567092	29	come	30	0.00250188
5	trump	66	0.00550413	30	announc	29	0.00241848
6	say	65	0.00542073	31	immigr	29	0.00241848
7	campaign	55	0.00458677	32	week	29	0.00241848
8	interview	49	0.0040864	33	washington	29	0.00241848
9	debat	49	0.0040864	34	elect	28	0.00233508
10	unit	48	0.004003	35	confer	28	0.00233508
11	day	48	0.004003	36	world	28	0.00233508
12	wrote	46	0.00383621	37	claim	27	0.00225169
13	guardian	46	0.00383621	38	wednesday	27	0.00225169
14	british	43	0.00358602	39	presidenti	26	0.00216829
15	accord	43	0.00358602	40	die	26	0.00216829
16	univ	42	0.00350263	41	friday	25	0.0020849
17	candid	41	0.00341923	42	latest	25	0.0020849
18	media	41	0.00341923	43	win	25	0.0020849
19	hillari	40	0.00333584	44	call	25	0.0020849
20	newspap	38	0.00316904	45	anti	25	0.0020849
21	monday	36	0.00300225	46	support	25	0.0020849
22	press	36	0.00300225	47	statement	24	0.0020015
23	stori	36	0.00300225	48	common	23	0.00191811
24	tuesday	35	0.00291886	49	tell	23	0.00191811

Figure 23: Reliable Hot Words

UNIVERSITÉ CATHOLIQUE DE LOUVAIN
École polytechnique de Louvain

Rue Archimède, 1 bte L6.11.01, 1348 Louvain-la-Neuve, Belgique | www.uclouvain.be/epl