Fake News Detection Using Machine Learning

Author: **Simon Lorent**
Supervisor: **Ashwin Itoo**

A thesis presented for the degree of
Master in Data Science

University Of Liège
Faculty Of Applied Science
Belgium
Accademic Year 2018-2019

Contents

Master thesis

Fake news detection using machine learning

Simon Lorent

Acknowledgement

I would start by saying thanks to my family, who have always been supportive and who have always believed in me.
I would also thanks Professor Itoo for his help and the opportunity he gave me to works on this very interesting subject.
In addition I would also thank all the professors of the faculty of applied science for what they taught me during these five years at the University of Liège.

Master thesis

Fake news detection using machine learning

Simon Lorent

Abstract

For some years, mostly since the rise of social media, fake news have become a society problem, in some occasion spreading more and faster than the true information. In this paper I evaluate the performance of Attention Mechanism for fake news detection on two datasets, one containing traditional online news articles and the second one news from various sources. I compare results on both dataset and the results of Attention Mechanism against LSTMs and traditional machine learning methods. It shows that Attention Mechanism does not work as well as expected. In addition, I made changes to original Attention Mechanism paper[1], by using word2vec embedding, that proves to works better on this particular case.

Chapter 1

Introduction

1.1 What are fake news?

1.1.1 Definition

Fake news has quickly become a society problem, being used to propagate false or rumour information in order to change peoples behaviour. It has been shown that propagation of fake news has had a non-negligible influence of 2016 US presidential elections[2]. A few facts on fake news in the United States:

- 62% of US citizens get their news for social medias[3]
- Fake news had more share on Facebook than mainstream news[4].

Fake news has also been used in order to influence the referendum in the United Kingdom for the "Brexit".

In this paper I experiment the possibility to detect fake news based only on textual information by applying traditional machine learning techniques[5, 6, 7] as well as bidirectional-LSTM[8] and attention mechanism[1] on two different datasets that contain different kinds of news.
In order to work on fake news detection, it is important to understand what is fake news and how they are characterized. The following is based on *Fake News Detection on Social Media: A Data Mining Perspective*[9].

The first is characterization or what is fake news and the second is detection. In order to build detection models, it is need to start by characterization, indeed, it is need to understand what is fake news before trying to detect them.

Fake news definition is made of two parts: authenticity and intent. Authenticity means that fake news content false information that can be verified as such, which means that conspiracy theory is not included in fake news as there are difficult to be proven true or false in most cases. The second part, intent, means that the false information has been written with the goal of misleading the reader.

1.1.2 Fake News Characterization

Definition 1 *Fake news is a news article that is intentionally and verifiable false*

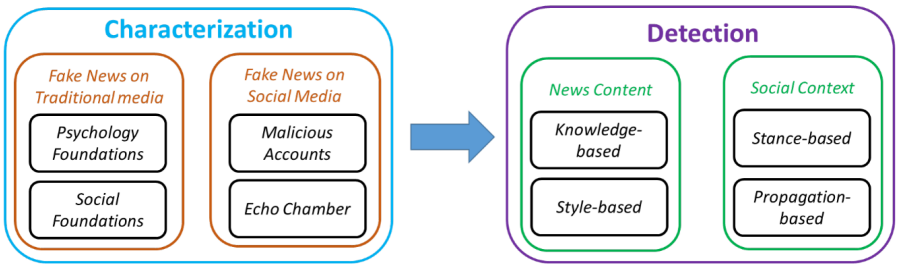


Figure 1.1: Fake news on social media: from characterization to detection.[9]

The part of the definition introducing the intent of misleading the reader automatically discard satire news media, that is why this work will focus on the first part, the fact that the piece of information is verifiable false or true. Indeed, even if satire news media does not have the intent to mislead the readers, not all of them have the ability of making criticism and not taking it to the first degree. On the other hand, in the case of political media, even if it clearly tries to influence the consumer, verifying the authenticity of there claims is usually harder as, in most of the cases, openly lies.

1.2 Feature Extraction

1.2.1 News Content Features

Now that fake news has been defined and the target has been set, it is needed to analyse what features can be used in order to classify fake news. Starting by looking at news content, it can be seen that it is made of four principal raw components:

- **Source**: Where does the news come from, who wrote it, is this source reliable or not.
- **Headline**: Short summary of the news content that try to attract the reader.
- **Body Text**: The actual text content of the news.
- **Image/Video**: Usualy, textual information is agremented with visual information such as images, videos or audio.

Features will be extracted from these four basic components, with the mains features being linguistic-based and visual-based. As explained before, fake news is used to influence the consumer, and in order to do that, they often use a specific language in order to attract the readers. On the other hand, non-fake news will mostly stick to a different language register, being more formal. This is linguistic-based features, to which can be added lexical features such as the total number of words, frequency of large words or unique words.

The second features that need to be taken into account are visual features. Indeed, modified images are often used to add more weight to the textual information. For example, the **Figure 1.2** is supposed to show the progress of deforestation, but the two images are actually from the same original one, and in addition the WWF logo makes it look like to be from a trusted source.

Figure 1.2: The two images provided to show deforestation between two dates are from the same image taken at the same time. [10]

1.2.2 Social Context Features

In the context of news sharing on social media, multiple aspect can be taken into account, such as user aspect, post aspect and group aspect. For instance, it is possible to analyse the behaviour of specific users and use their metadata in order to find if a user is at risk of trusting or sharing false information. For instance, this metadata can be its centre of interest, its number of followers, or anything that relates to it.

Post-based aspect is in a sense similar to users based: it can use post metadata in order to provide useful information, but in addition to metadata, the actual content can be used. It is also possible to extract features from the content using latent Dirichlet allocation (LDA)[11].

1.3 News Content Models

1.3.1 Knowledge-based models

Now that the different kinds of features available for the news have been defined, it is possible to start to explain what kinds of models can be built using these features. The first model that relates to the news content is based on knowledge: the goal of this model is to check the truthfulness of the news content and can be achieved in three different ways (or a mixture of them):

- **Expert-oriented**: relies on experts, such as journalists or scientists, to assess the news content.
- **Crowdsourcing-oriented**: relies on the wisdom of crowd that says that if a sufficiently large number of persons say that something is false or true then it should be.
- **Computational-oriented**: relies on automatic fact checking, that could be based on external resources such as DBpedia.

These methods all have pros and cons, hiring experts might be costly, and expert are limited in number and might not be able to treat all the news that is produced. In the case of crowdsourcing, it can easily be fooled if enough bad annotators break the system and automatic fact checking might not have the necessary accuracy.

1.3.2 Style-Based Model

As explained earlier, fake news usually tries to influence consumer behaviour, and thus generally use a specific style in order to play on the emotion. These methods are called deception-oriented stylometric methods.

The second method is called objectivity-oriented approaches and tries to capture the objectivity of the texts or headlines. These kind of style is mostly used by partisan articles or yellow journalism, that is, websites that rely on eye-catching headlines without reporting any useful information. An example of these kind of headline could be

> You will never believe what he did !!!!!!

This kind of headline plays on the curiosity of the reader that would click to read the news.

1.4 Social Context Models

The last features that have not been used yet are social media features. There are two approaches to use these features: stance-based and propagation-based.

Stance-based approaches use implicit or explicit representation. For instance, explicit representation might be positive or negative votes on social media. Implicit representation needs to be extracted from the post itself.

Propagation-based approaches use features related to sharing such as the number of retweet on twitter.

1.5 Related Works

1.5.1 Fake news detection

There are two main categories of state of the art that are interesting for this work: previous work on fake news detection and on general text classification. Works on fake news detection is almost inexistent and mainly focus in 2016 US presidential elections or does not use the same features. That is, when this work focus on automatic features extraction using machine learning and deep learning, other works make use of hand-crafted features[12, 13] such as psycholinguistic features[14] which are not the goal here.

Current research focus mostly on using social features and speaker information in order to improve the quality of classifications.

Ruchansky et al.[15] proposed a hybrid deep model for fake news detection making use of multiple kinds of feature such as temporal engagement between n users and m news articles over time and produce a label for fake news categorization but as well a score for suspicious users.

Tacchini et al.[16] proposed a method based on social network information such as likes and users in order to find hoax information.

Thorne et al.[17] proposed a stacked ensemble classifier in order to address a subproblem of fake news detection which is stance classification. It is the fact of finding if an article agree, disagree or simply discus a fact.

Granik and Mesyura[18] used Naïve-Bayes classifier in order to classify news from buzzfeed datasets.
In addition to texts and social features, Yang et al.[19] used visual features such as images with a convolutional neural network.
Wang et al.[20] also used visual features for classifying fake news but uses adversarial neural networks to do so.

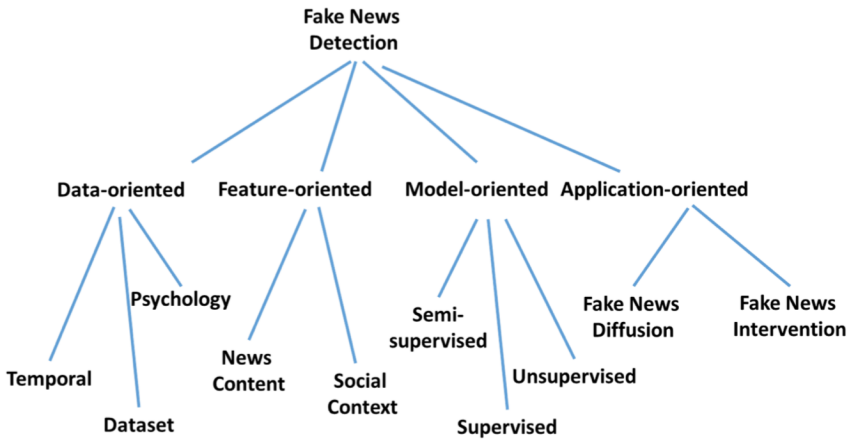


Figure 1.3: Different approaches to fake news detection.

1.5.2 State of the Art Text classification

When it comes to state of the art for text classification, it includes Long short-term memory (LSTM)[8], Attention Mechanism[21], IndRNN[22], Attention-Based Bidirection LSTM[1], Hierarchical Attention Networks for Text Classification[23], Adversarial Training Methods For Supervised Text Classification[24], Convolutional Neural Networks for Sentence Classification[25] and RMDL: Random Multimodel Deep Learning for Classification[26]. All of these models have comparable performances.

1.6 Conclusion

As it has been shown in **Section 1.2** and **Section 1.3** multiple approaches can be used in order to extract features and use them in models. This works focus on textual news content features. Indeed, other features related to social media are difficult to acquire. For example, users information is difficult to obtain on Facebook, as well as post information. In addition, the different datasets that have been presented at **Section 3.2** does not provide any other information than textual ones.
Looking at **Figure 1.3** it can be seen that the main focus will be made on unsupervised and supervised learning models using textual news content. It should be noted that machine learning models usually comes with a trade-off between precision and recall and thus that a model which is very good at detected fake news might have a high false positive rate as opposite to a model with a low false positive rate which might not be good at detecting them. This cause ethical questions such as automatic censorship that will not be discussed here.

Chapter 2

Related Work

2.1 Supervised Learning for Fake News Detection[12]

Reis et al. use machine learning techniques on buzzfeed article related to US election. The evaluated algorithm are k-Nearest Neighbors, Naïve-Bayes, Random Forests, SVM with RBF kernel and XGBoost.

In order to feed this network they used a lot of hand-crafted features such as

- Language Features: bag-of-words, POS tagging and others for a total of 31 differents features,
- Lexial Features: amount of uniques words and their frenquencies, pronouns, etc,
- Pyschological Features[14]: build using Linguistic Inquiry and Word Count which is a specific dictionary build by a text mining software,
- Semantic Features: Toxic score from Google's API,
- Engagement: Number of comments within several time interval.

Many other features where also used, based on the source and social metadata.

Their results is shown at **Figure 2.1**.

Table 1. Results obtained for different classifiers w.r.t AUC and F1 score.

Classifier	AUC	F1
KNN	0.80±0.009	0.75±0.008
NB	0.72±0.009	0.75±0.001
RF	**0.85±0.007**	**0.81±0.008**
SVM	0.79±0.030	0.76±0.019
XGB	**0.86±0.006**	**0.81±0.011**

RF and XGB performed best.

Figure 2.1: Results by Reis et al.

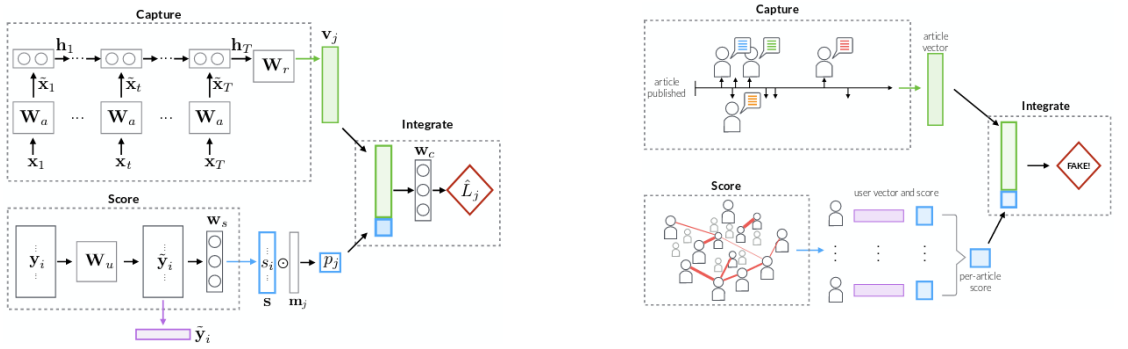
They also shows that XGBoost is good for selecting texts that need to be hand verfied. This model is limited by the fact they do use metadata that are not always available. Pérez-Rosas et al.[13] used almost the same set of features but used linear SVM as a model and worked on a different dataset.

2.2 CSI: A Hybrid Deep Model for Fake News Detection

Ruchansky et al.[15] used an hybrid network, merging news content features and metadata such as social engagement in a single network. To do so, they used an RNN for extracting temporal features of news content and a fully connected network in the case of social features. The results of the two network is then concatenated and use for final classification.

As textual features they used doc2vec[27].

Network's architecture is shown at **Figure 2.2**.



(a) The CSI model specification. The Capture module depicts the LSTM for a single article a_j, while the Score module operates over all users. The output of Score is then filtered to be relevant to a_j.

(b) Intuition behind CSI. Here, Capture receives the temporal series of engagements, and Score is fed an implicit user graph constructed from the engagements over all articles in the data.

Figure 2.2: CSI model

They did test their model on two datasets, one from Twitter and the other one from Weibo, which a Chinese equivalant of Twitter. Compared to simpler models, CSI performs better, with 6% improvement over simple GRU network (**Figure 2.3**).

	Twitter		Weibo	
	Accuracy	F-score	Accuracy	F-score
DT-Rank	0.624	0.636	0.732	0.726
DTC	0.711	0.702	0.831	0.831
SVM-TS	0.767	0.773	0.857	0.861
LSTM-1	0.814	0.808	0.896	0.913
GRU-2	0.835	0.830	0.910	0.914
CI	0.847	0.846	0.928	0.927
CI-t	0.854	0.848	0.939	0.940
CSI	**0.892**	**0.894**	**0.953**	**0.954**

Table 2: Comparison of detection accuracy on two datasets

Figure 2.3: Results by Ruchansky et al.

2.3 Some Like it Hoax: Automated Fake News Detection in Social Networks [16]

Here, Tacchini et al. focus on using social network features in order to improve the reliability of their detector. The dataset was collected using Facebook Graph API, collection pages from two mains categories: scientific news and conspiracy news. They used logistic regression and harmonic algorithm[28] to classify news in categories hoax and non hoax. Harminoc Algorithm is a method that allows to transfer information across users who liked some common posts.

For the training they used cross-validation, dividing the dataset into 80% for training and 20% for testing and performing 5-fold cross-validation, reaching 99% of accuracy in both cases.

In addition they used one-page out, using posts from a single page as test data or using half of the page as training and the other half as testing. This still lead to good results, harmonic algorithm outperforming logistic regression. Results are shown at **Figures 2.4** and **2.5**.

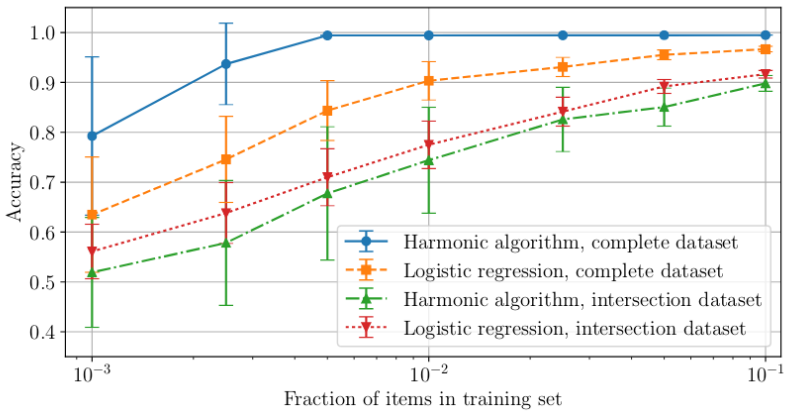


Figure 2.4: Results by tacchnini et al.

	One-page-out		Half-pages-out	
	Avg accuracy	Stdev	Avg accuracy	Stdev
Logistic regression	0.794	0.303	0.716	0.143
Harmonic BLC	0.991	0.023	0.993	0.002

Figure 2.5: Results by tacchnini et al.

Chapter 3

Data Exploration

3.1 Introduction

A good starting point for the analysis is to make some data exploration of the data set. The first thing to be done is statistical analysis such as counting the number of texts per class or counting the number of words per sentence. Then it is possible to try to get an insight of the data distribution by making dimensionality reduction and plotting data in 2D.

3.2 Datasets

3.2.1 Fake News Corpus

This works uses multiple corpus in order to train and test different models. The main corpus used for training is called Fake News Corpus[29]. This corpus has been automatically crawled using `opensources.co` labels. In other words, domains have been labelled with one or more labels in

- Fake News
- Satire
- Extreme Bias
- Conspiracy Theory
- Junk Science
- Hate News
- Clickbait
- Proceed With Caution
- Political
- Credible

These annotations have been provided by crowdsourcing, which means that they might not be exactly accurate, but are expected to be close to the reality. Because this works focus on fake news detection against reliable news, only the news labels as fake and credible have been used.

3.2.2 Liar, Liar Pants on Fire

The second dataset is **Liar, Liar Pants on Fire** dataset[30], which is a collection of twelve thousand small sentences collected from various sources and hand labelled. They are divided in six classes:

- pants-fire
- false
- barely-true
- half-true
- mostly-true
- true

This set will be used a second test set. Because in this case there are six classes against two in the other cases, a threshold should be used in order to fix which one will be considered as true or false in order to be compared with the other dataset.
It should be noted that this one differs from the two other datasets is it is composed only on short sentences, and thus it should not be expected to have very good results on this dataset for models trained on Fake News Corpus which is made of full texts. In addition, the texts from the latest dataset are more politically oriented than the ones from the first one.

3.3 Dataset statistics

3.3.1 Fake News Corpus

General Analysis

Because **Fake News Corpus** is the main dataset, the data exploration will start with this dataset. And the first thing is to count the number of items per class. Before starting the analysis, it is needed to clean up the dataset. As it is originally given in a large 30GB CSV file, the first step is to put everything in a database in order to be able to retrieve only wanted a piece of information. In order to do so, the file has been red line by line. It appears that some of the lines are badly formated, preventing them from being read correctly, in this case they are dropped without being put in the database. Also, each line that is a duplicate of a line already red is also dropped. The second step in cleaning the set consists of some more duplicate removal. Indeed, dropping same lines remove only exact duplicate. It appears that some news does have the same content, with slight variation in the title, or a different author. In order to remove the duplicate, each text is hashed using SHA256 and those hash a compared, removing duplicates and keeping only one.

Because the dataset has been cleaned, numbers provided by the dataset creators and number computed after cleaning will be provided. We found the values given at **Table 3.1**. It shows that the number of fake news is smaller by a small factor with respect to the number of reliable news, but given the total number of items it should not cause any problems. But it will still be taken into account later on. In addition to the numbers

Type	Provided	Computed
Fake News	928,083	770,287
Satire	146,080	855,23
Extreme Bias	1,300,444	771,407
Conspiracy Theory	905,981	495,402
Junk Science	144,939	79,342
Hate News	117,374	65,264
Clickbait	292,201	176,403
Proceed With Caution	319,830	104,657
Political	2,435,471	972,283
Credible	1,920,139	1,811,644

Table 3.1: Number of texts per categories

provided at **Table 3.1**, there are also two more categories that are in the dataset but for which no description is provided:

- Unknown: 231301
- Rumour: 376815

To have a better view of the distribution of categories, a histogram is provided at **Figure 3.1**.

In addition, the number of words per text and the average number of words per sentences have been computed for each text categories. **Figure 3.2** shows the boxplots for these values. It can be seen that there is no significative difference that might be used in order to make class prediction.

Before counting the number of words and sentences, the texts are preprocessed using gensim[31] and NLTK[32]. The first step consists of splitting text into an array of sentences on stop punctuation such as dots or questions mark, but not on commas. The second step consists of filtering words that are contained in these sentences, to do so, stop words (words such as 'a', 'an', 'the'), punctuation, words or size less or equal to tree, non-alphanumeric words, numeric values and tags (such as html tags) are removed. Finally, the number of words still present is used.

An interesting feature to look at is the distribution of news sources with respect to their categories. It shows that in some case some source a predominant. For instance, looking at **Figure 3.3** shows that most of the reliable news are from *nytimes.com* and in the same way, most of the fake news is coming from *beforeitsnews.com*. That has to be taken into account when training and testing models as the goal is not to distinguish between these two sources but between fake news and reliable news.

Another import feature to look at is the distribution of the number of words in the text. Indeed, at some point it will be needed to fix a constant length for the texts and using to small length would mean a lot of cutting and using too long size would mean too much padding. It is thus needed to investigate the length of the texts in order to choose the

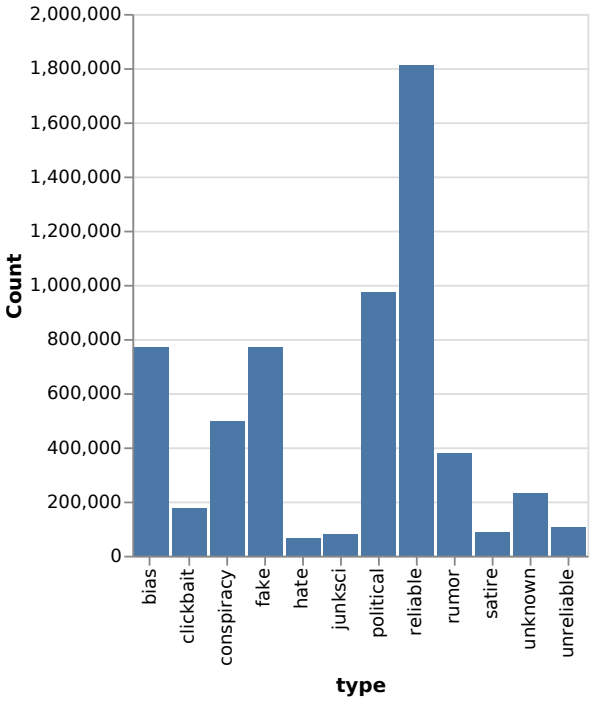
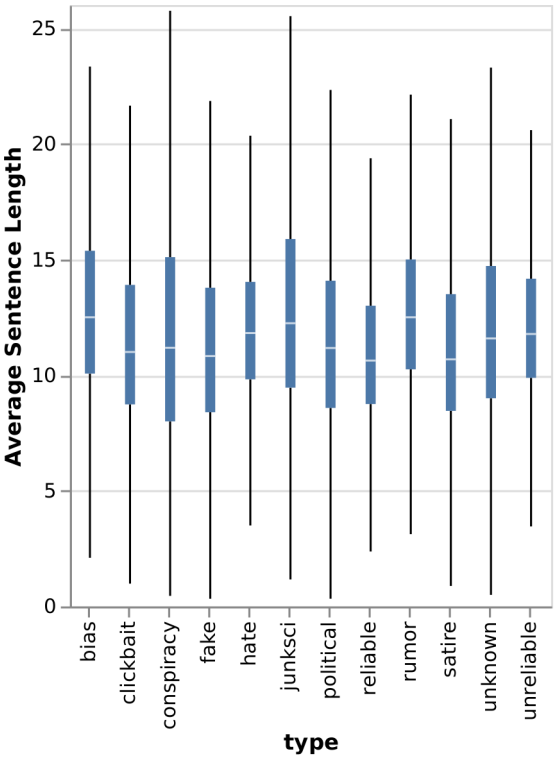
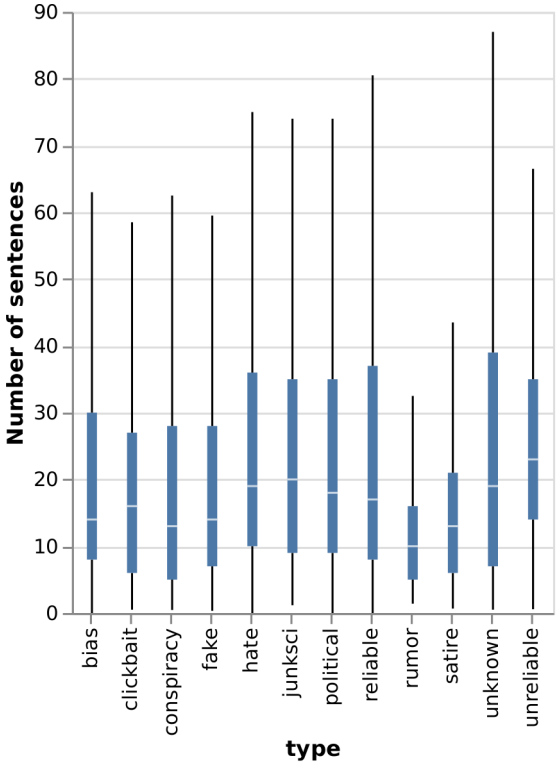


Figure 3.1: Histogram of text distribution along their categories on the computed numbers.

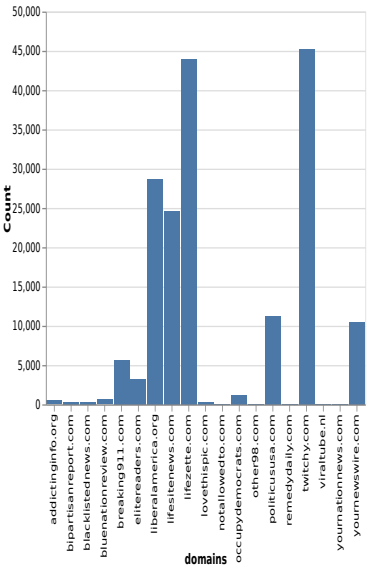


(a) Boxplot of average sentence length for each category.

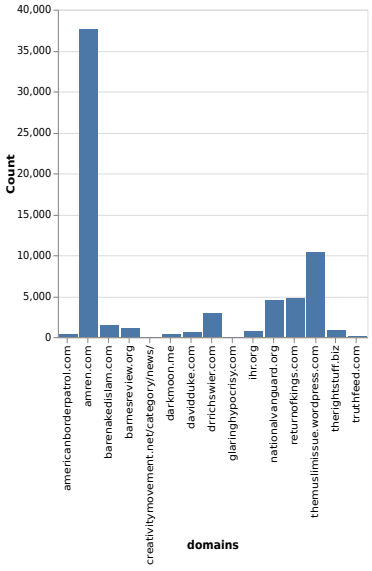


(b) Boxplot of number of sentences for each category.

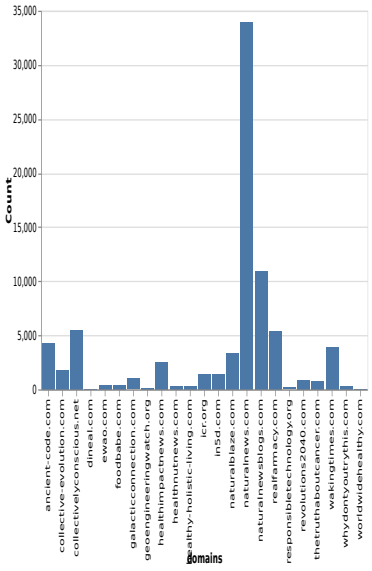
Figure 3.2: Summary statistics



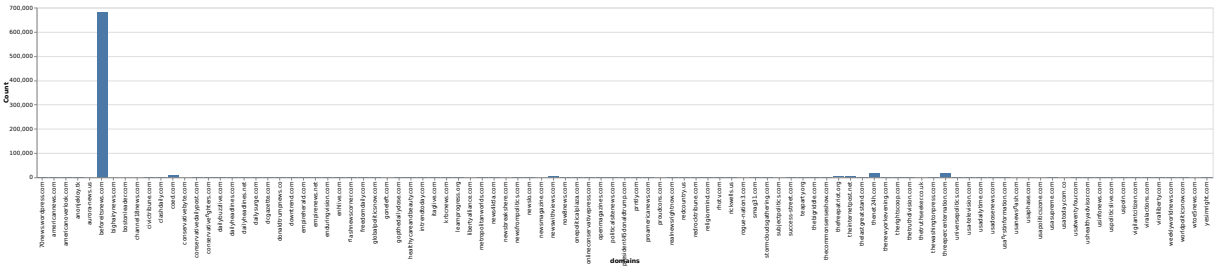
(a) Clickbaits



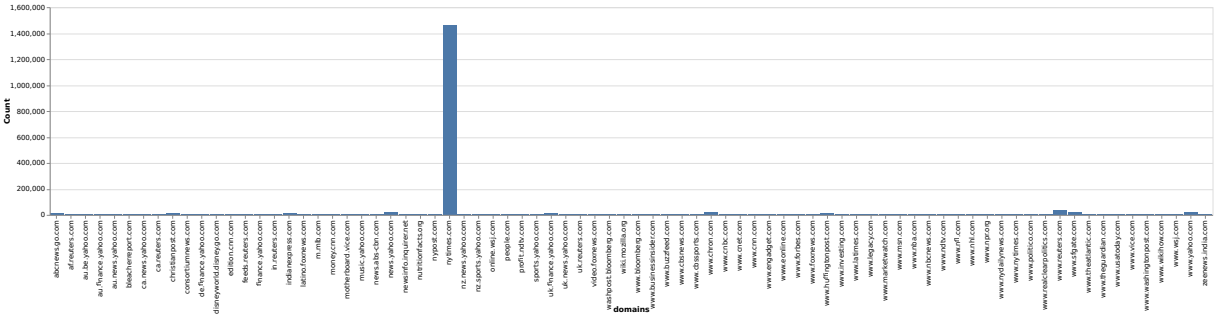
(b) Hate



(c) Junk sience



(d) Fake



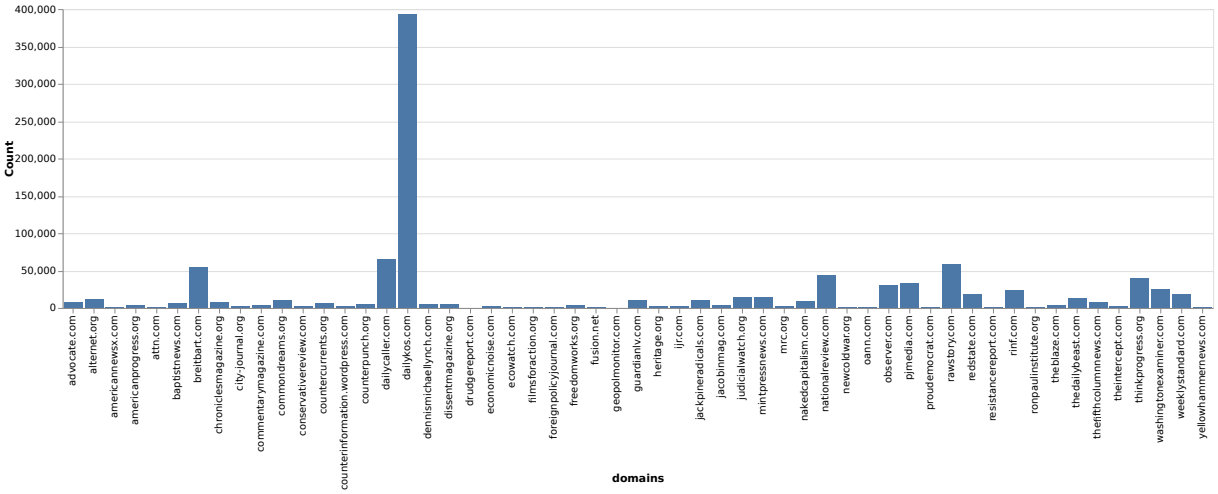
(e) Reliable

right one. It can be seen at **Figure 3.4** that reliable news has slightly more words than fake news, but the difference is minimal.

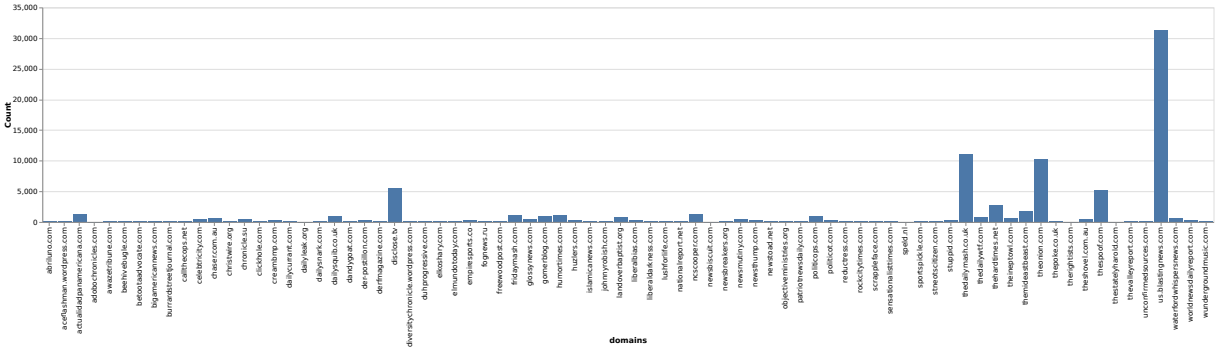
Fake News analysis

In this section, the analysis will focus on fake news and reliable news. Because of what shows **Figure 3.3**, that is, some categories are almost all from the same source, an analysis of what happens when dropping these sources. First, lets compare the amount of news while and while not taking into account major sources.

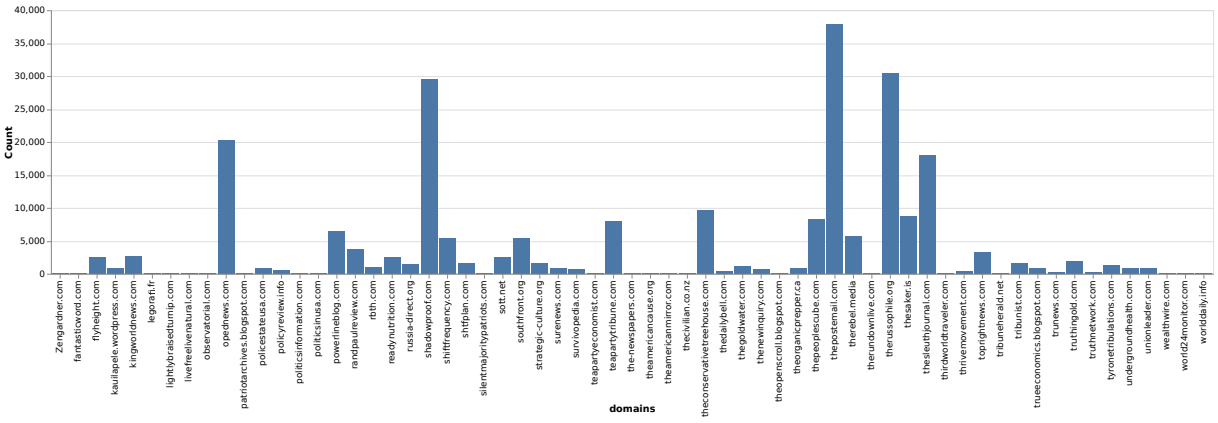
Comparing **Figure 3.5** and **Figure 3.6** shows that even by removing *nytimes.com* and *beforeitsnews.com* news from the dataset still leave enough texts to train the different models without the risk of learning something unwanted such as only separating these two sources. But one drawback is that the ratio between fake news and reliable news is



(f) Political



(g) Satire



(h) Unknown

Figure 3.3: Histogram of news origin for each category.

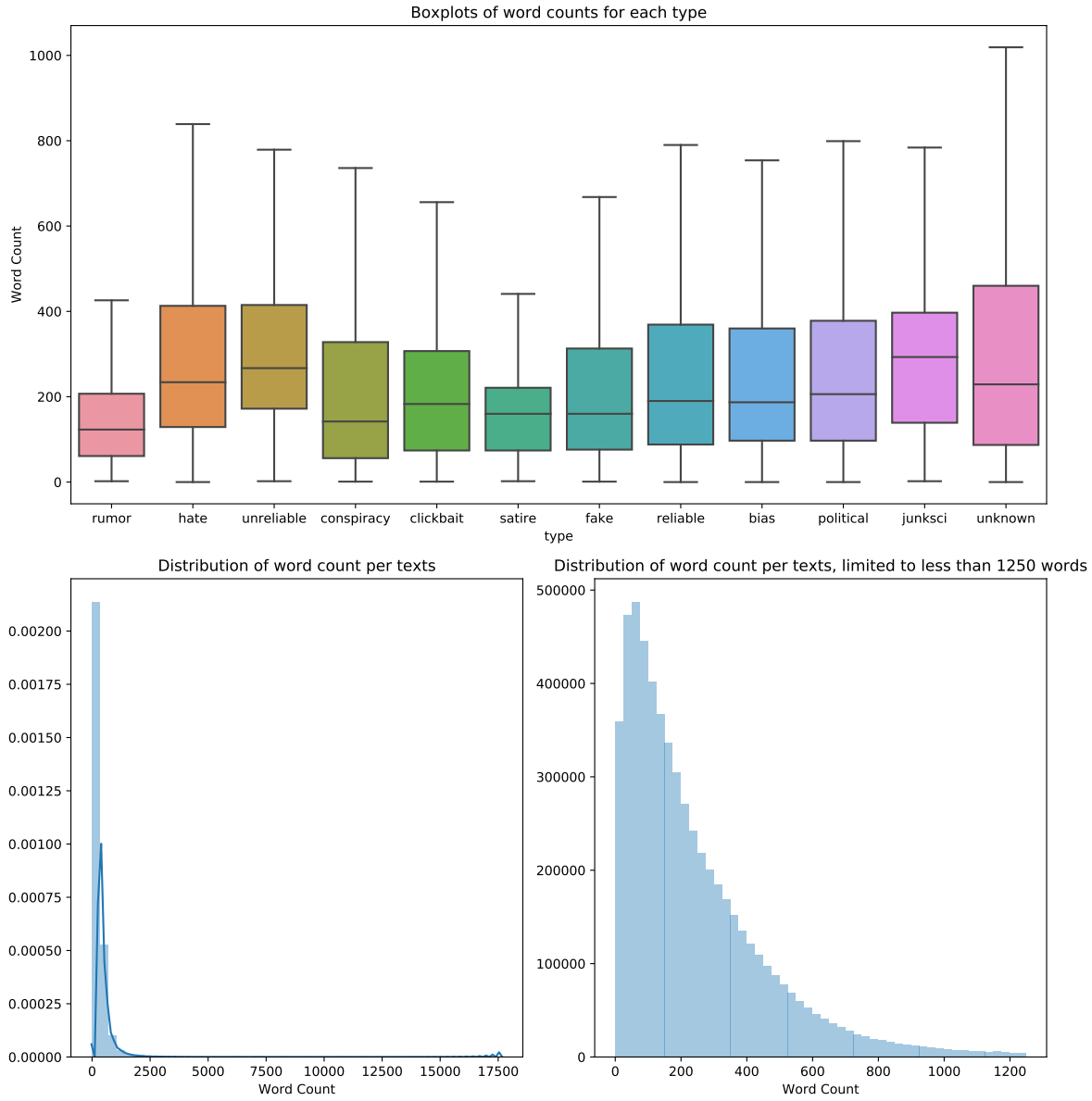


Figure 3.4: Distribution of the number of words per text

going from around one half to around one fourth.

3.3.2 Liar-Liar Corpus

As said in **Chapter 1**, this dataset is made of small text of one or two sentences at most. Which means that they are smaller than the ones in *fake news corpus*. The distribution of words length can be seen at **Figure 3.7**. In addition, this dataset is not unbalanced as the other corpus is, which means that precautions do not have to be taken.

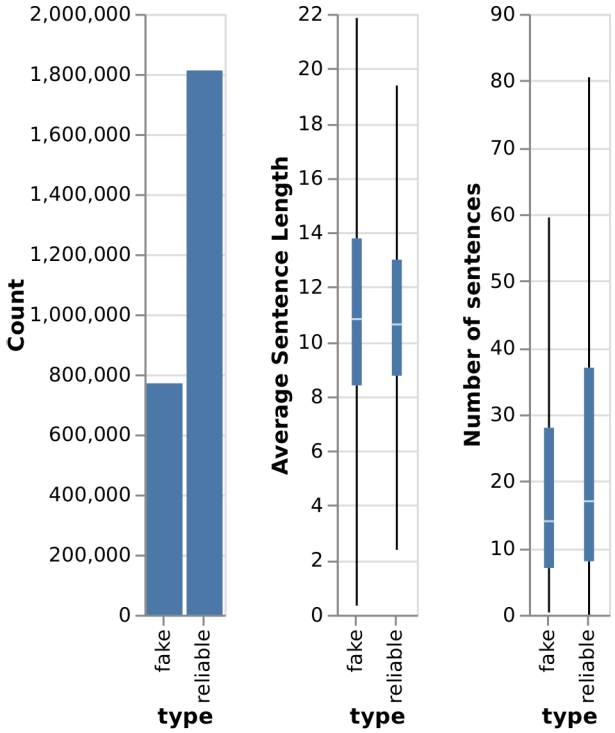


Figure 3.5: Summary statistics for not downsampled fake and reliable news.

3.4 Visualization With t-SNE

In order to visualize the data, it is needed to transform text in a numerical way and eventually reduce the dimension in order to allow it to be plotted on a 2D or 3D plot. Here TF-IDF (term frequency, inverse document frequency[6, 33]) is used. How it works will be details later on. This produces a sparse matrix with each document being represented as an array, each value of the array being a value for one term. That is, the more term in the corpus, the longer the array becomes. For example, a corpus of 10.000 text with 20.000 unique words would be represented as a 10000×20000 sparse matrix. As said before, plotting in 20000 dimension is not possible. In order to do so, the number of dimensions needs to be reduced. Here, principal component analysis before t-SNE[34] will be used together.

What is t-SNE? It stands for **t-distributed stochastic neighbour embedding**. The goal of this method is that two points to are closed in $\mathcal{R}^Q$ should be close in $\mathcal{R}^S, S \ll Q$. In order to do so, the algorithm starts by fitting a probability distribution on the input points. It starts by computing.

$$p_{i|j} = \frac{\frac{exp(-||x_i - x_j||^2)}{2\sigma_i^2}}{\sum_{k \neq i} \frac{exp(-||x_i - x_k||^2)}{2\sigma_i^2}}$$

Each of there $y_i, i \in 0 \dots N$ being a point of dimensions Q.
Then

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

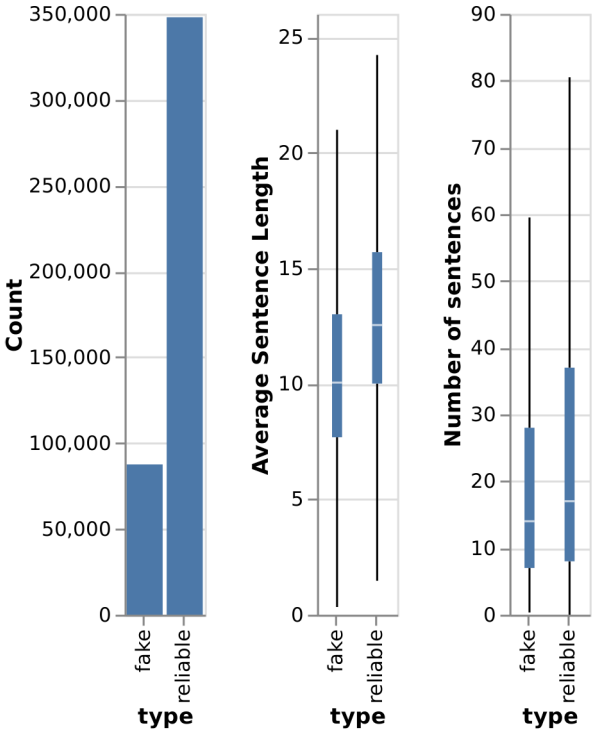


Figure 3.6: Summary statistics for downsampled dataset on fake and reliable news

is computed.
The probability distribution for the low density map is given by

$$q_{ij} = \frac{(1 + ||y_i - y_j||^2)^{-1}}{\sum_{k \neq l}(1 + ||y_k - x_l||^2)^{-1}}$$

And in order to have a low dimension probability distribution as close as possible to the high dimension one, it minimizes the KullbackLeibler divergence of the two distributions.

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

The minimization of KL divergence is achieved by gradient descent. The documentation of the module of scikit-learn[35] used for computing this value recommends applying PCA first in order to reduce the original dimension and speedup computation. The result of these computations can be seen at **Figure 3.8**. These firsts 500 PCA components explain around 47% of the total variance. The figure shows that there is no clear clustering of the classes. Increasing the number of PCA components to 1750 gives the results at **Figure 3.9** and does not show more clustering, even if it explains 75% of the variance. This shows that classifying the dots might not be easy, but it should be reminded that it is a dimensionality reduction and that there is a loss of information. Some of the original data dimensions can have a better separation of the classes.

It is not possible to use t-SNE on the *Fake News Corpus* because the algorithm is quadratic with respect to the number of samples. Which makes it impossible to compute for that corpus which is larger than the *liar-liar* corpus. But it is still possible to try to make some visualization using truncated singular value decomposition.
In this case, only with a 2D projection, we can already see some kind of separation between the two classes, thus we can think that the **Fake News Corpus** will be easier to deal with.

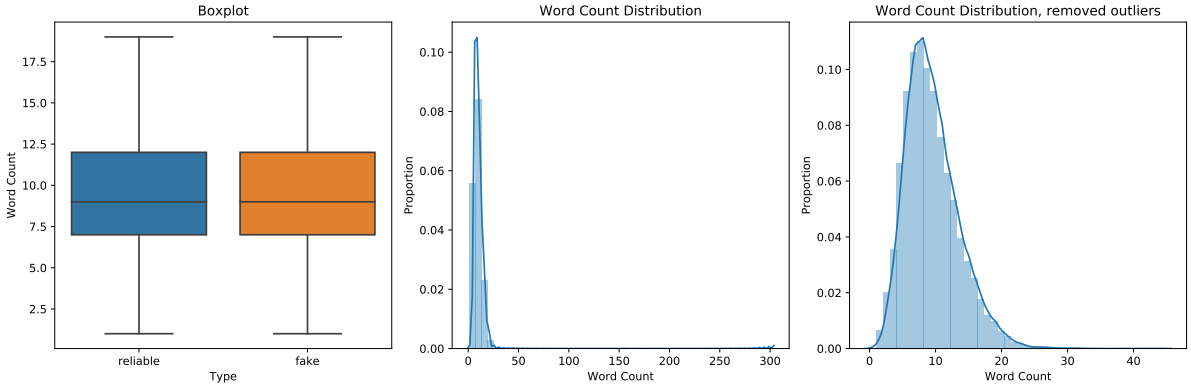


Figure 3.7: Number of words distributions for liar-liar datasets. On the first and the third plots, a few outliers with length greater than 50 have been removed in order to make the plots more readable.

3.5 Conclusion

Data exploration has shown that there is no real statistical differences between text metadata for fake and reliable news, and thus make it not interesting for using it for classifying new texts. In addition, dimensionality reduction does not show any sign of helpfulness for the classification.

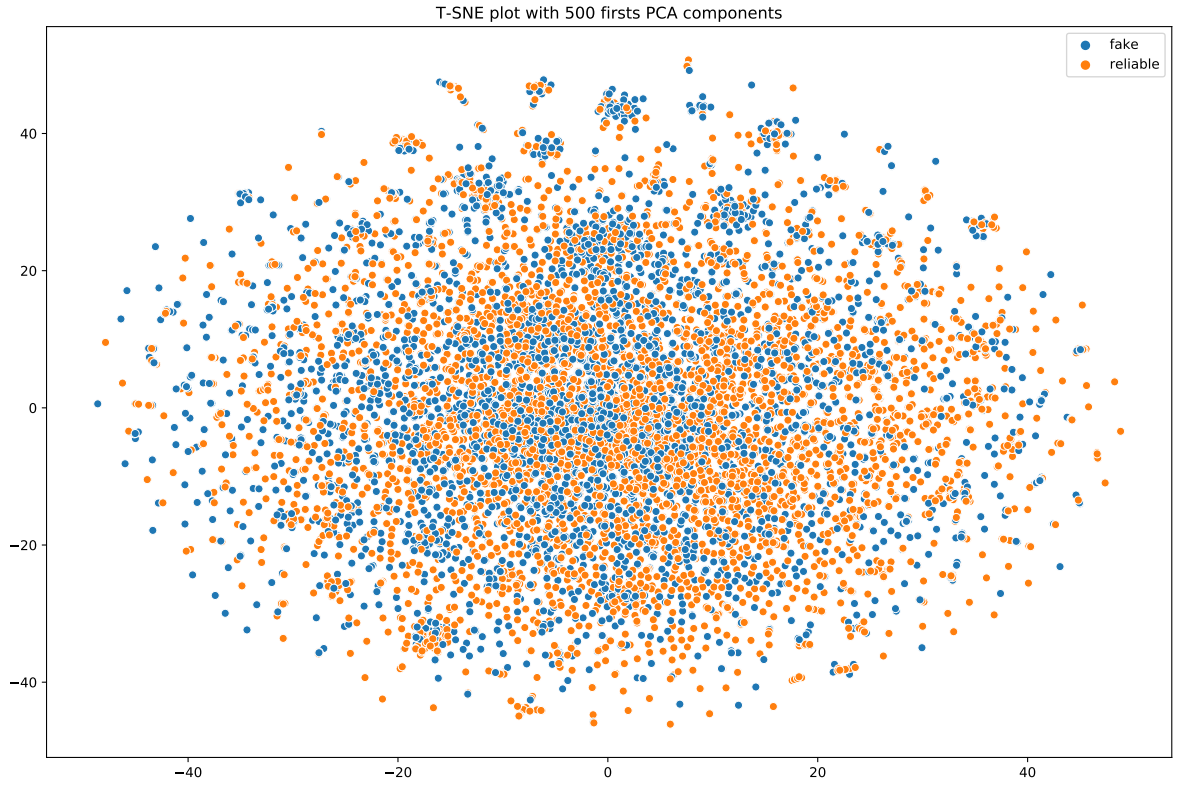


Figure 3.8: t-SNE plot for liar-liar dataset.

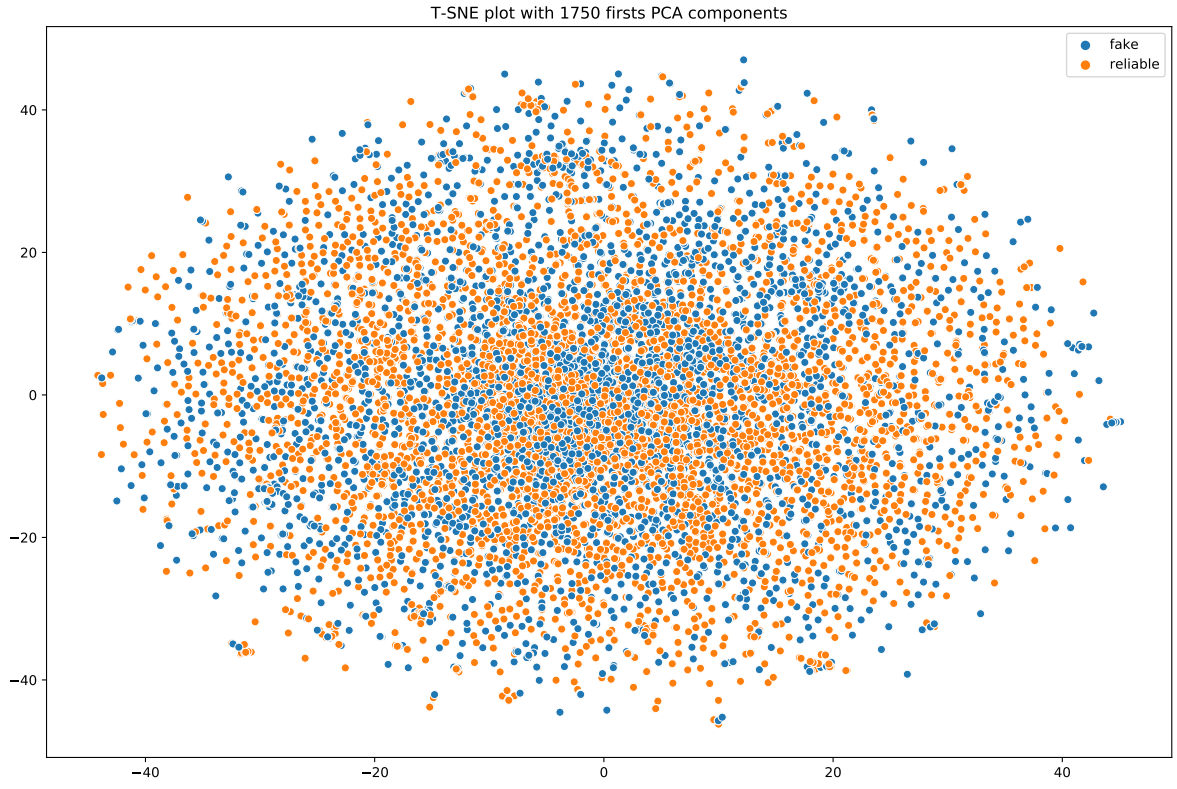


Figure 3.9: t-SNE plot for liar-liar dataset.

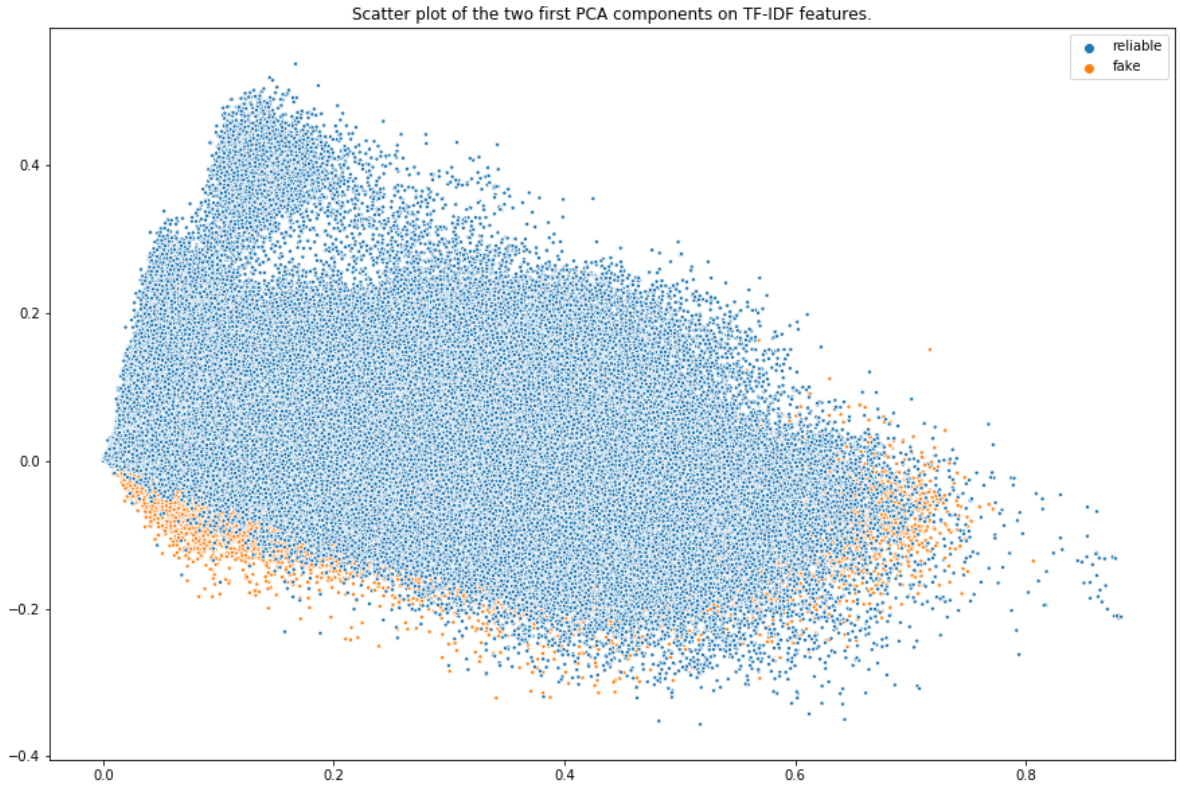


Figure 3.10: First two LSA components for fake news corpus

Chapter 4

Machine Learning techniques

4.1 Introduction

In this chapter, we will focus on the more traditional methods used in natural language processing such as Naïve-Bayes, decision trees, linear SVM and others. These will serve as a baseline for comparing the performances of the more two advanced models that will be analysed later on: LSTM and Attention Mechanism. The first thing to do when working with text is the do words and texts embedding, indeed, in order to use machine learning algorithms on texts, a mathematical representation of these texts is required.

4.2 Text to vectors

As explained before, text needs to be represented in a way that gives more meaningful information than a simple sequence of bits, which have additional drawbacks such that for a given word, the sequence of bits representing it depends on the coding. The first and simplest coding that comes to mind is a one-hot encoding: a matrix M of size number of texts $\times$ number of words where $M_{ij} = 1$ if the word j is present in the text i and 0 in the other case. But this is still not enough as each word is given the same weight, no matter how often it appears in the text.
In order to overcome this problem, term-frequency might be used, that is, rather than setting M_{ij} to 0 or 1 we set it to the amount of time it appears in the text.
It is possible to use even better text embedding. It is called term-frequency, inverse document frequency. The main idea is that a word that appears often in all the documents is not helpful in order to classify the documents. For example, if the task is to classify books of biology and physics, words atoms, cells or light are more useful than today or tomorrow.
In order to compute tf-idf, it is separated in two parts, the first one being term frequency and the second one inverse document frequency. We have that

$$tf_{ij} = \#(W_j | W_j \in D_i) \tag{4.1}$$

That it tf_{ij} is the number of times the word j appears in the document i. Secondly, we have that

$$idf_j = \log(\frac{\#D}{\#(D_i | W_j \in D_i)})$$

this is the log of the total number of documents, over the number of documents that contains the word j. Finally, the value tfidf value is computed by

$$tf - idf_{ij} = tf_{ij} * idf_j \tag{4.2}$$

This the text embedding methods that will be used in this section.

4.3 Methodology

All the methods presented will be tested in two different ways:

- On the liar-liar dataset
- On the fake corpus dataset, excluding the news from *beforeitsnews.com* and *nytimes.com*

To be more precise, in the first case, the models will be trained on a training set, tuned using validation set and finally tested using test set. In the second case, the same methodology will be used, the dataset has been split be choosing 60% of the text from each domain for training, and 20% for validation and testing. This way of splitting has been chosen because of the uneven representation of each domain in the dataset in order to ensure representation of all the domains in the tree subsets.

4.3.1 Evaluation Metrics

In order to evaluate each model, multiple evaluation metrics have been used. There are recall, precision and f1-score. It is needed to use multiple metrics because they don't all account for the same values. For instance, it is possible to have a model with a recall of 1 that behave extremely bad because it simply classifies all the inputs in the same single class. Remember That Precision Is Defined As

$$Precision = \frac{TP}{TP + FP} \tag{4.3}$$

Which means that we can have two different precision, depending on which classes is considered as being positive. This is the proportion of correctly classified positive elements over the number of elements classified as positive. It is equals to 1 when there is no false positive, but it does not mean that all the positive elements are correctly classified as it might be some false negative. The recall helps to solve this problem. It Is Defined As

$$recall = \frac{TP}{TP + FN} \tag{4.4}$$

The f1-score combines the recall and the precision. It is defined by

$$f1 - score = \frac{2 * precision * recall}{precision + recall} \tag{4.5}$$

It is also possible to look at the weighted average of all these values. For instance, it is possible to compute the global recall by averaging the recall for both classes by the respective class ratio.
Finally, raw output can be used by looking at the confusion matrix.
The first parameter to tune is the max number of features used by tf-idf. This is the maximum number of words that will be kept to create the text encoding. The words that are kept are the most frequent words.

4.4 Models

Four models have been used in order to classify texts represented as a TF-IDF matrix. These are Multinomial Naïve-Bayes, Linear SVM, Ridge Classifier and Decision Tree. I will start by a very brief recap of each model and how they work.

4.4.1 Naïve-Bayes[7]

The basic idea of Naïve-Bayes model is that all features are independent of each other. This is a particularly strong hypothesis in the case of text classification because it supposes that words are not related to each other. But it knows to work well given this hypothesis. Given an element of class y and vector of features $\mathbf{X} = (x_1, ..., x_n)$. The probability of the class given that vector is defined as

$$P(y|\mathbf{X}) = \frac{P(y) * P(\mathbf{X}|y)}{P(\mathbf{X})} \tag{4.6}$$

Thanks to the assumption of conditional independence, we have that

$$P(x_i|y, x_1, ..., x_{i-1}, x_{i+1}, ..., x_n) = P(x_i|y) \tag{4.7}$$

Using Bayes rules we have that

$$P(y|x_1, ..., x_n) = \frac{P(y)\prod_{i=1}^{n} P(x_i|y)}{P(x_1, ..., x_n)} \tag{4.8}$$

Because $P(x_1, ..., x_n)$ is constant, we have the classification rule

$$\hat{y} = \underset{y}{argmax} P(y) \prod_{i=1}^{n} P(x_i|y) \tag{4.9}$$

4.4.2 Linear SVM

Linear SVM is a method for large linear classification. Given pairs of features-label $(\mathbf{x_i}, y_i), y_i \in \{-1, 1\}$, it solves the following unconstrained optimization problem.

$$\underset{w}{min} \frac{1}{2}\mathbf{w^T w} + \mathbf{C}\sum_{i=1}^{l} \xi(\mathbf{w}; \mathbf{x_i}, y_u) \tag{4.10}$$

Where ξ is a loss function, in this case L2 loss function has been used, and $\mathbf{C} > 0$ a penalty parameter. Class of new examples are assigned by looking at the value of $\mathbf{w^T w}$. The class 1 is assigned if $\mathbf{w^T w} \geq 0$ and the class -1 if $\mathbf{w^T w} < 0$.

4.4.3 Decision Tree[36]

Decision tree works by recursively selecting features and splitting the dataset on those features. These features can either be nominal or continuous.
In order to find the best split, it uses gini impurity.

$$G = \sum_{i=1}^{C} p(i) * (1 - p(i)) \tag{4.11}$$

Figure 4.1: Tuning linearSVC parameters

Where $p(i)$ is the probability of class i in the current branch. The best split is chosen as the one that decreases the most the impurity. For instance, beginning from the root, the gini impurity is computed on the complete dataset, then the impurity of each branch is computed over all features, weighting it by the number of elements in each branch. The chosen feature is the one that has the highest impurity.

4.4.4 Ridge Classifier

Ridge classifier works the same way as ridge regression. It states the problem as a minimization of the sum of square errors with penalization. It can be expressed as in **Equation 4.12**.

$$\min_{w} ||Xw - y||_2^2 + \alpha ||w||_2^2 \tag{4.12}$$

The predicted class if positive if Xw is positive and negative otherwise.

4.5 Models on liar-liar dataset

4.5.1 Linear SVC

In the case of linear SVC there is one parameter to tune up, which is the penalty parameters for the error term. **Figure 4.1** shows the three main metrics with respect to the penalty parameter. This show that a parameter of around 0.1 is the best one.

4.5.2 Decision Tree

With decision trees, it is possible to reduce overfitting by pruning the tree. It is possible to do pre-pruning or post pruning. Pre-pruning means that a stopping criterion is used to stop tree growth earlier and post pruning cut the tree once it has been fully grown. It this case pre-pruning is done by limiting the maximum depth of the tree. **Figure 4.2**

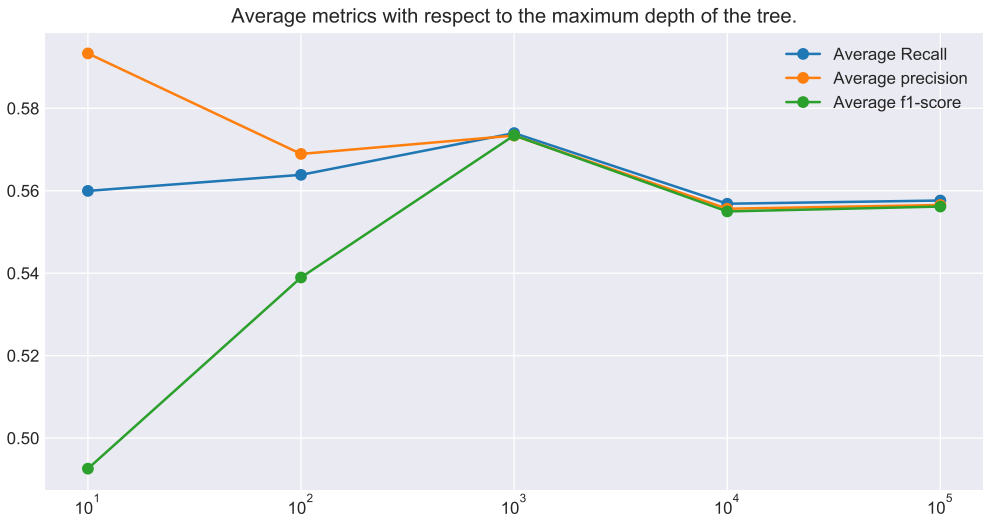


Figure 4.2: Tuning Decision Tree Parameters

shows metrics values for different depths. It seems that tree of depth 1000 are the best ones.

4.5.3 Ridge Classifier

With the ridge classifier model, it is also possible to tweak the penalty value of the optimization problem. At **Figure 4.3** we can see that the optimal parameter is around 10 or 20, depending of the metrics that we want to maximize. Later on, the value of 10 will be chosen as a compromise between precision and recall. It is the value that maximizes the f1-score.

4.5.4 Max Feature Number

Starting with the maximum number of features, the precision of each model can be analysed when limiting the maximum number of words. The results for each model can be seen at **Figure 4.4, 4.5** and **4.6**. Shows that depending on the metrics we want to optimize it is better to choose different parameters. For instance, in order to maximize F1-score, it is better to use a maximum number of features of 1000.
The results are slightly different if the goal is to optimize the precision because if the best value stays the same for Linear SVM and Ridge Classifier, the Naïve-Bayes work better when using the maximum number of features and it goes the same way for recall. Based on **Figure 4.4** we can say that when it comes to precision and recall, Naïve-Bayes is the one that performs the best.
Row results for max features selection are available at **Appendix A**. It goes differently when we focus on a single class. For example, the precision for fake detection is at its maximum for Linear SVM and Ridge Classifier when only 10 features are used. But at the same time, it is at its minium for reliable class. It shows that when trying to optimize the overall model and not only for a single class, it is better to look at the weighted average than at the value for a single class. But it is still important to look at the metrics for a single class because it indicates how it behaves for this class. For instance, in the case of

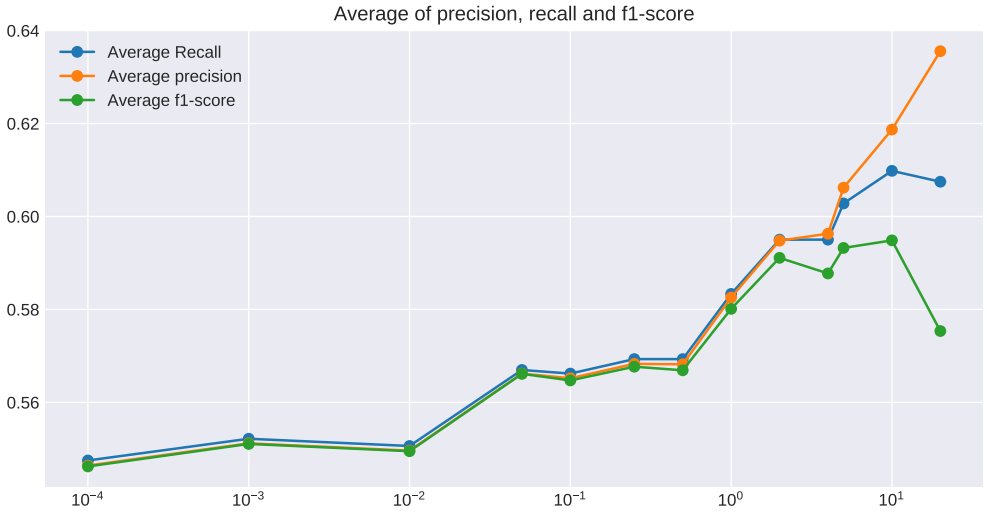


Figure 4.3: Average metrics for ridge classifiers with respect to the penalty parameter.

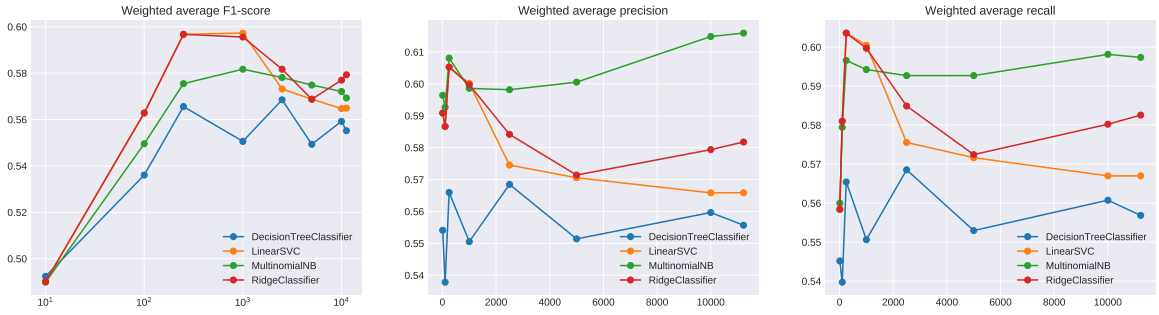


Figure 4.4: Weighted average of f1-score, precision and recall of each class.

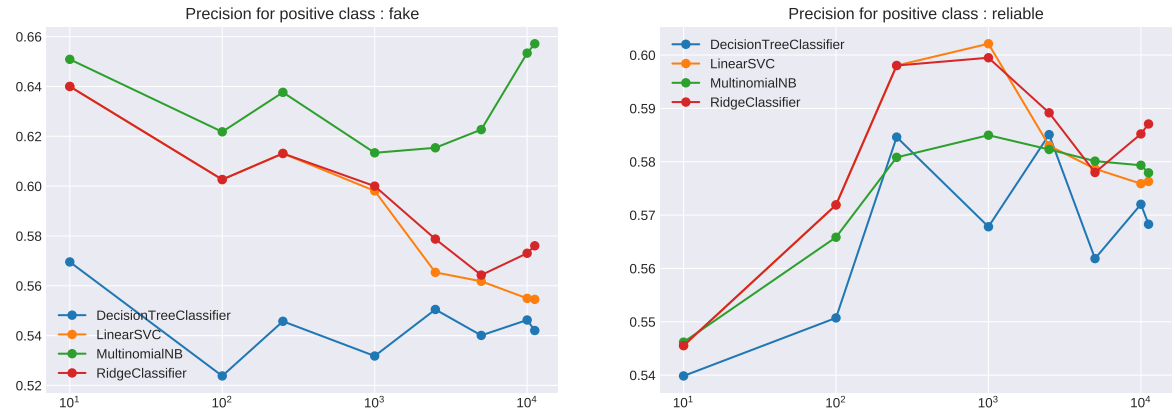


Figure 4.5: Precision of the model for each class, the x axes is log scale of the number of features

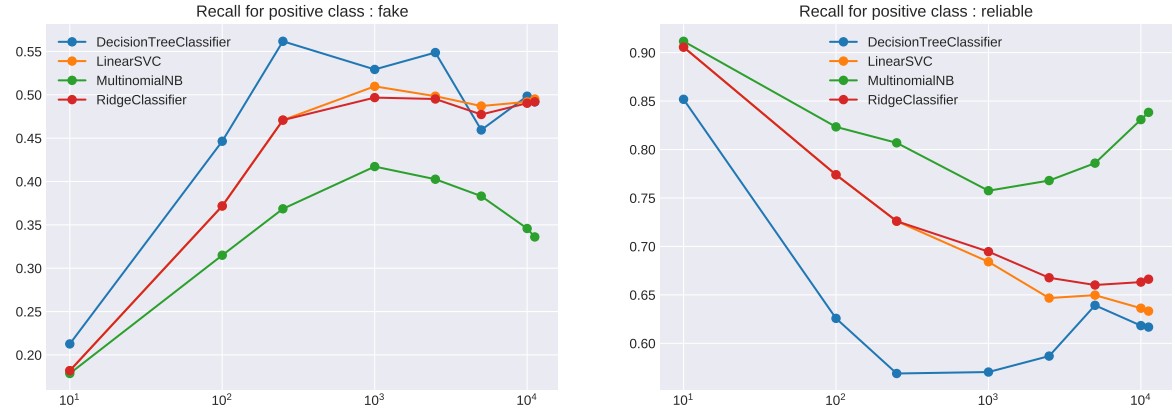


Figure 4.6: Precision of the model for each class, the x axes is log scale of the number of features

automatic fake news detection, it is important to minimize the number of reliable news misclassified in order to avoid what could be called censorship.

4.6 Models on fake corpus dataset

4.6.1 SMOTE: Synthetic Minority Over-sampling Technique[37]

As it has been shown in **Chapter 3**, the fake news corpus is unbalanced. Synthetic minority oversampling is a technique that allows generating fake samples from the minor class. It works by randomly choosing one or many nearest neighbours in the minority class. For instance, if the algorithm is set to use 5 nearest neighbours, for each sample it will choose one of its nearest neighbours, and generate a new sample on the segment joining the sample and its neighbour. **Algorithm 1** and **2** shows how it works. The first one computes the k-nearest neighbours and the second one computes a new element by randomly choosing one of these neighbours.

Data: k = Number of nearest neighbours
Data: T = number of minority class samples
for $i \leftarrow 1...T$ **do**
  Compute k-nearest neighbours of sample i;
  Populate(knn, i);
end

Algorithm 1: SMOTE

Data: knn = the k-nearest neighbour of sample i
Data: s = ith sample
nn = random_choice(knn);
newSample = s + rand(0, 1) * (nn - s);

Algorithm 2: Populate

4.6.2 Model selection without using SMOTE

Hyperparameters tuning

As for the models trained on the **liar-liar corpus**, hyper-parameters can be optimized the same way. The average metric for each model with respect to their parameters are shown at **Figure 4.7, 4.8** and **4.9**.

The optimal parameter for the ridge classifier is clearly 1. As well as for the decision tree trained on **liar-liar** dataset, the optimal maximum depth is of 1000. And finally, the optimal value for the penalty parameter of the svm is also 1.

By looking at **Figure 4.5, 4.6** and **4.4** we can find optimal parameters for the number of features used in TF-IDF. It shows that linear svm and ridge classifiers are the ones that perform the best, having an average precision of sligtly more than 94% for the linear svm and 94% for the ridge classifier. They achieve these performances from $50,000$ features and does not decrease. On the other hand, Naïve-Bayes reaches a pike at $100,000$ features and greatly decrease afterward.

Figure 4.4 shows why it is important to look at all the metrics, because Naïve-Bayes reaches a recall of 1 for the reliable class and close to 0 for the fake class, which means

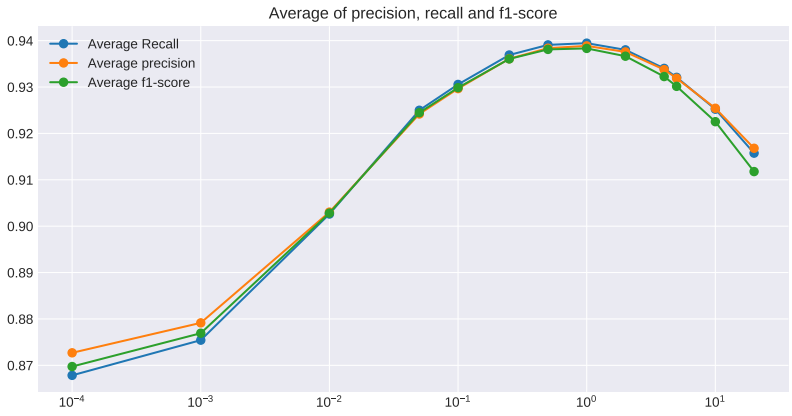


Figure 4.7: Metrics value with respect to the penalty parameter for ridge classifier

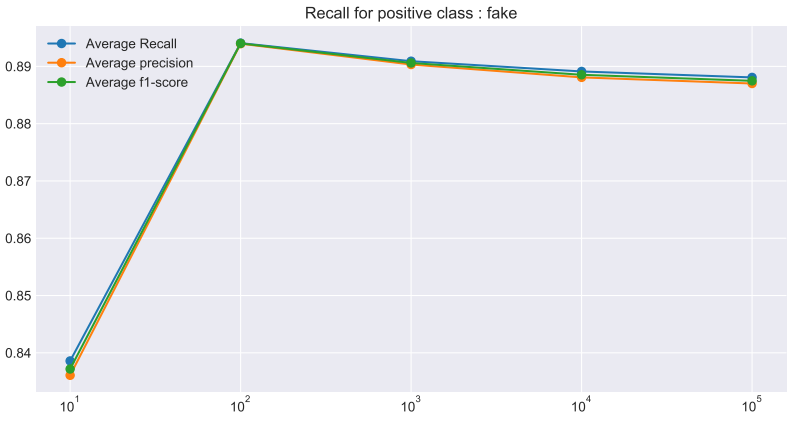


Figure 4.8: Optimal depth of decision tree.

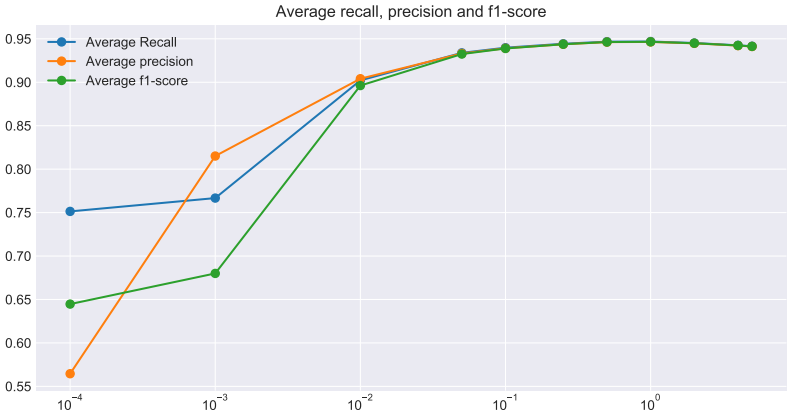


Figure 4.9: Optimal penalty parameters for linear svm

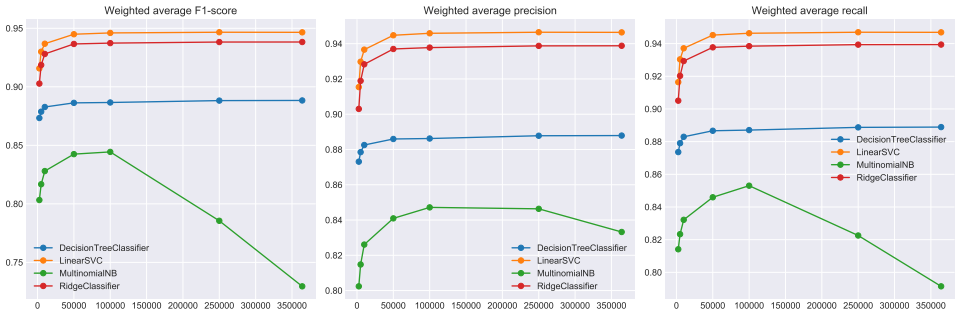


Figure 4.10: Average recall, precision and f1-score wti respect to the maximum number of features.

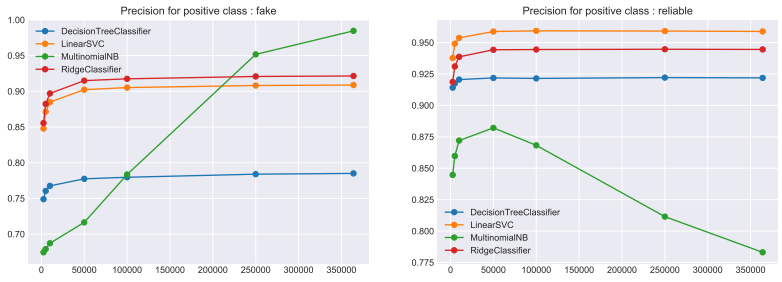


Figure 4.11: Precision for fake and reliable class for each model with respect to the maximum number of features

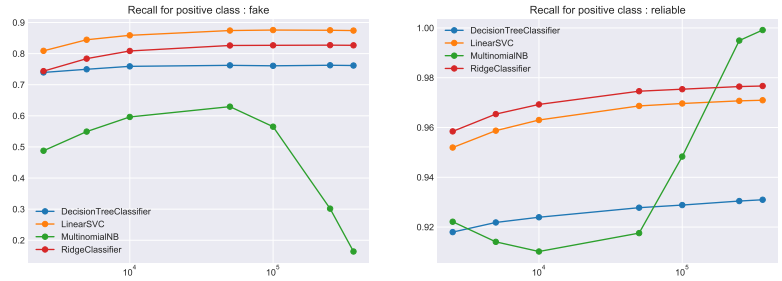


Figure 4.12: Recall for fake and reliable class for each model with respect to the maximum number of features

that almost all texts are classified as reliable. This can be verified by looking at **Figure 4.21**, only a small proportion of true fake is actually classified as it.

4.6.3 Model selection with SMOTE

The first thing that can be noticed at **Figure 4.14, 4.15, 4.16, 4.17, 4.18** and **4.19** is that the two models that worked the best without applying SMOTE method, linear SVM and ridge classifiers are still the ones that perform the best. By comparing **Figure 4.9** and **Figure 4.16** we can see that it works better when applying a smaller regularization parameter. It goes from 0.66% of accuracy to 0.86% of accuracy thus acting as a regularization. The same does not apply to ridge classifiers.

It also has a huge impact on how Naïve-Bayes behaves as it removes overfitting when using a larger number of features in TF-IDF, leading to a few percent of accuracy increase.

It conclusion for SMOTE method we can say that it does help models that do not have a regularization parameter or when the regularization parameter is low. Thus it does help prevent overfitting.

4.7 Results on testing set

4.7.1 Methodology

Now that all the models have been tuned, they need to be tested independently on testing set. Each dataset contains a testing set.

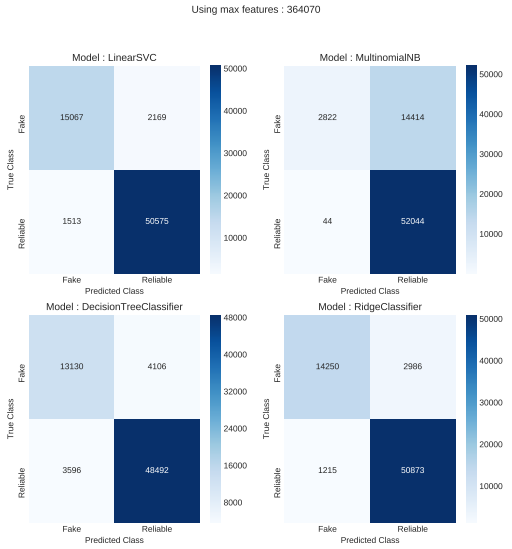


Figure 4.13: Confusion matrix for each model using 364, 070 features

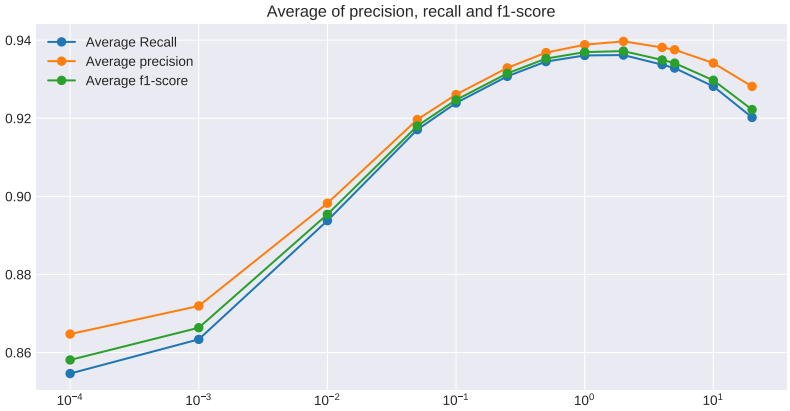


Figure 4.14: Metrics value with respect to the penalty parameter for ridge classifiers when using SMOTE.

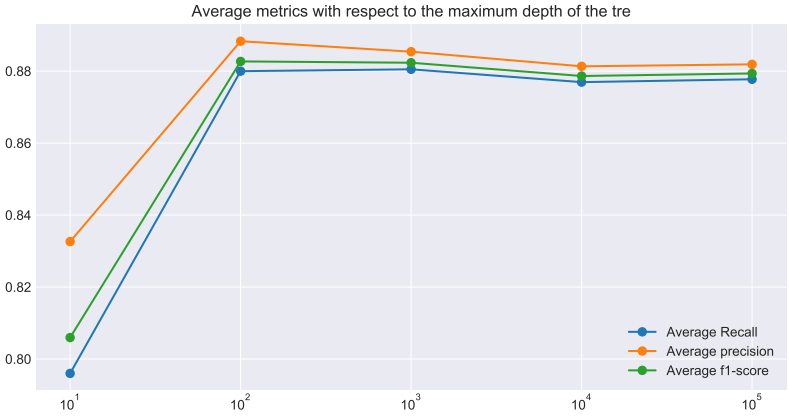


Figure 4.15: Optimal depth of decision tree when using SMOTE.

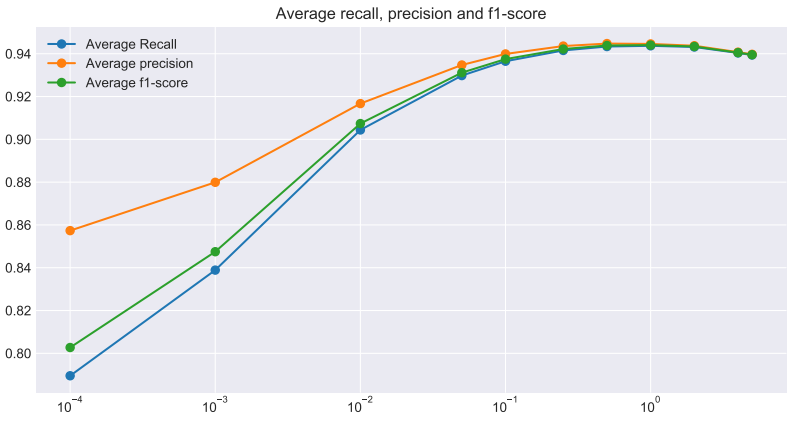


Figure 4.16: Optimal penalty parameters for linear svm when using SMOTE

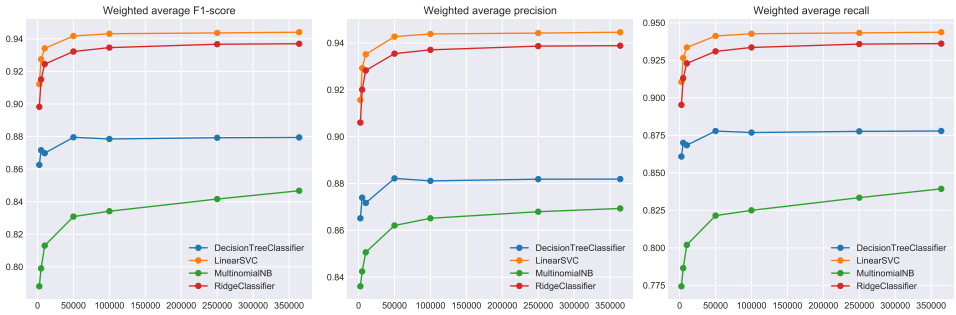


Figure 4.17: Average recall, precision and f1-score wti respect to the maximum number of features.

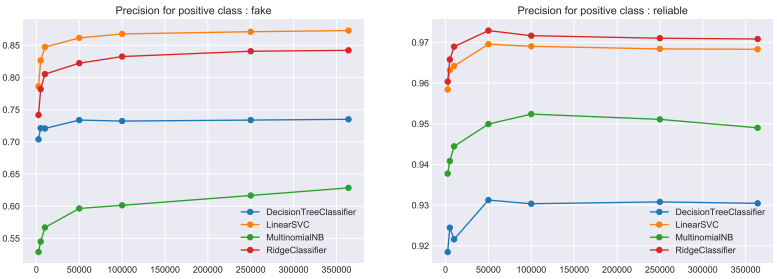


Figure 4.18: Precision for fake and reliable class for each model with respect to the maximum number of features

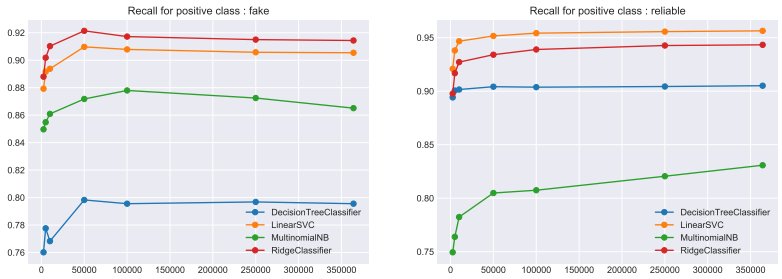


Figure 4.19: Recall for fake and reliable class for each model with respect to the maximum number of features

For the **liar-liar** dataset the following parameters will be used:

- Linear SVM with regularization parameters of 0.1 of a max TF-IDF features of 500,
- Ridge Classifier with $\alpha = 10$ and also max TF-IDF features of 500,
- Decision Tree with maximum depth of 1000 and the maximum number of features for TF-IDF,
- Naïve-Bayes will also use the maximum number of features for TF-IDF.

For the **Fake News Corpus**, the following setting will be used:

- Linear SVM with regularization parameters of 1,
- Ridge Classifier with $\alpha = 1$,
- Decision Tree with maximum depth of 100.

They will all be trained using $100,000$ features for TF-IDF. For the **Fake News Corpus** with SMOTE, the same parameters will be used, but the maximum number of features for TF-IDF will be used.
All the models will be trained on train and validation set and tested on test set.

4.7.2 Results

Liar-Liar Corpus

By looking at the row results, based on average accuracy Naïve-Bayes, Linear SVM and ridge classifiers perform very close, but when looking at the recall per class it shows that Naïve-Bayes is bad at detecting fake news and classifies most of the text as reliable, when Linear SVM and Ridge classifiers are more balanced. Finally, it is possible to look at the ROC curve at **Figure 4.20**. One more time, it shows that Naïve-Bayes, linear svm and ridge classifier have similar performance, but in this case it shows that NB has a little advantage, with a slightly larger AUC. There is only one point for the decision tree as it does not output probabilities for each class.

When it comes to the **Fake news corpus**, linear models are still the ones that perform the best, with linear svm reaching an accuracy of 94.7% and ridge classifiers 93.98%. Surprisingly, decision tree outperform Naïve-Bayes in this case, with an accuracy of 89.4%

	fake	reliable	accuracy	macro avg	weighted avg
f1-score	0.514399	0.679764	0.614049	0.597082	0.607588
precision	0.570485	0.638376	0.614049	0.604430	0.608744
recall	0.468354	0.726891	0.614049	0.597623	0.614049
support	1106.000000	1428.000000	0.614049	2534.000000	2534.000000

(a) Raw results for Linear SVM

	fake	reliable	accuracy	macro avg	weighted avg
f1-score	0.412107	0.698507	0.601421	0.555307	0.573504
precision	0.578431	0.608741	0.601421	0.593586	0.595512
recall	0.320072	0.819328	0.601421	0.569700	0.601421
support	1106.000000	1428.000000	0.601421	2534.000000	2534.000000

(b) Raw results for Naïve-Bayes

	fake	reliable	accuracy	macro avg	weighted avg
f1-score	0.496366	0.691279	0.617206	0.593822	0.606207
precision	0.582927	0.633606	0.617206	0.608266	0.611486
recall	0.432188	0.760504	0.617206	0.596346	0.617206
support	1106.000000	1428.000000	0.617206	2534.000000	2534.000000

(c) Raw results for Ridge Classifer.

	fake	reliable	accuracy	macro avg	weighted avg
f1-score	0.479354	0.591549	0.542226	0.535451	0.542580
precision	0.475936	0.594901	0.542226	0.535418	0.542977
recall	0.482821	0.588235	0.542226	0.535528	0.542226
support	1106.000000	1428.000000	0.542226	2534.000000	2534.000000

(d) Raw results for Decision Tree

Table 4.1: Raw results on **Liar-Liar Corpus**.

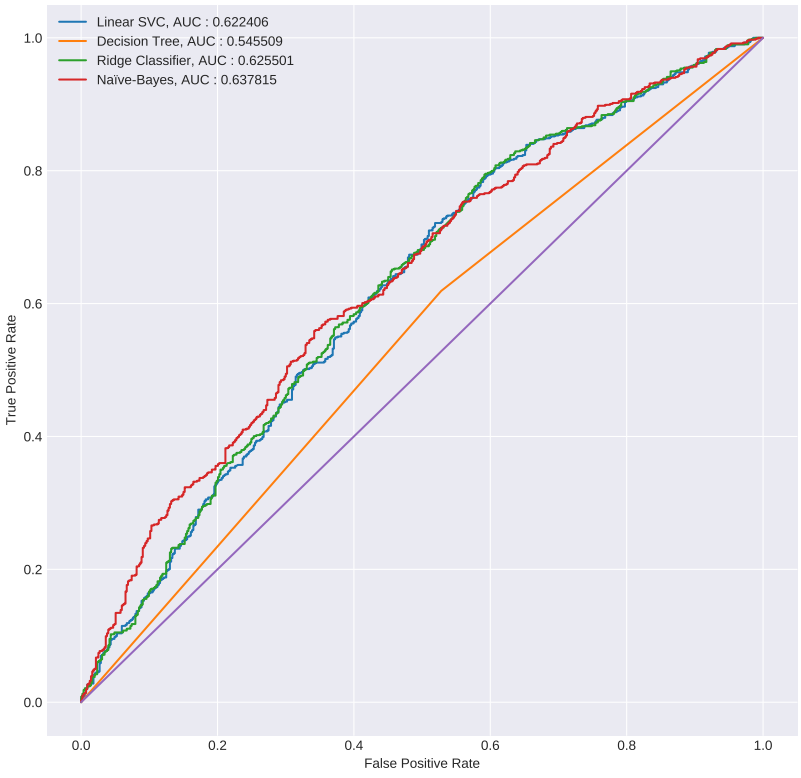


Figure 4.20: ROC curve for each model

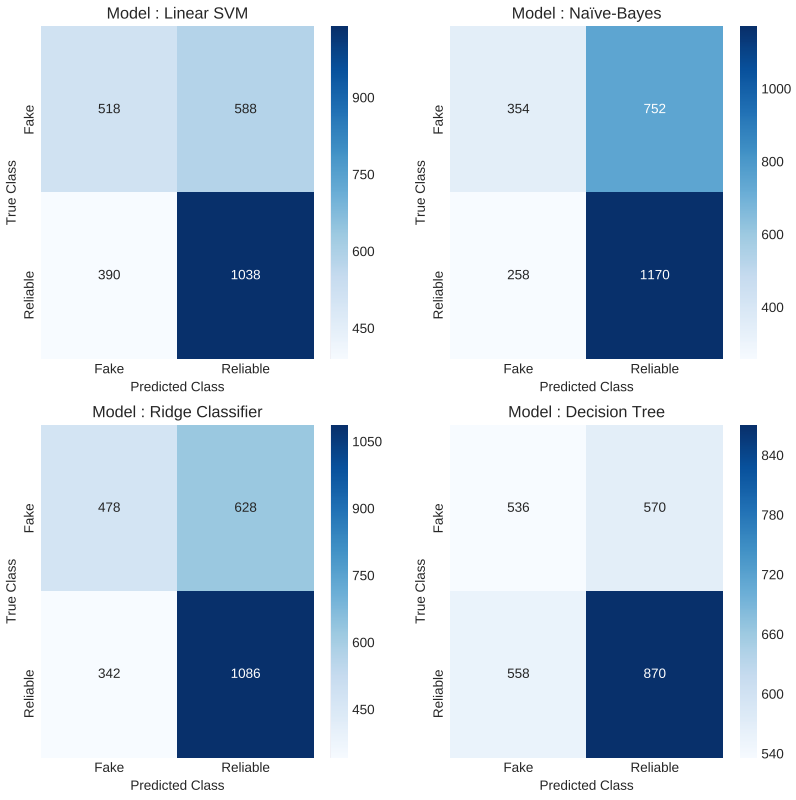


Figure 4.21: Confusion Matrix for each models

	fake	reliable	accuracy	macro avg	weighted avg
f1-score	0.894700	0.965364	0.947874	0.930032	0.947620
precision	0.907861	0.960783	0.947874	0.934322	0.947494
recall	0.881916	0.969989	0.947874	0.925952	0.947874
support	17496.000000	52181.000000	0.947874	69677.000000	69677.000000

(a) Raw results for Linear SVM on **Fake News Corpus**

	fake	reliable	accuracy	macro avg	weighted avg
f1-score	0.674458	0.905634	0.853682	0.790046	0.847585
precision	0.764127	0.875841	0.853682	0.819984	0.847790
recall	0.603624	0.937525	0.853682	0.770574	0.853682
support	17496.000000	52181.000000	0.853682	69677.000000	69677.000000

(b) Raw results for Naïve-Bayes on **Fake News Corpus**

	fake	reliable	accuracy	macro avg	weighted avg
f1-score	0.874220	0.960438	0.939808	0.917329	0.938788
precision	0.919674	0.945736	0.939808	0.932705	0.939192
recall	0.833048	0.975604	0.939808	0.904326	0.939808
support	17496.000000	52181.000000	0.939808	69677.000000	69677.000000

(c) Raw results for Ridge Classifier on **Fake News Corpus**

	fake	reliable	accuracy	macro avg	weighted avg
f1-score	0.791687	0.929799	0.894987	0.860743	0.895119
precision	0.788700	0.930987	0.894987	0.859844	0.895258
recall	0.794696	0.928614	0.894987	0.861655	0.894987
support	17496.000000	52181.000000	0.894987	69677.000000	69677.000000

(d) Raw results for Decision Tree on **Fake News Corpus**

Table 4.2: Results on **Fake News Corpus** without using SMOTE.

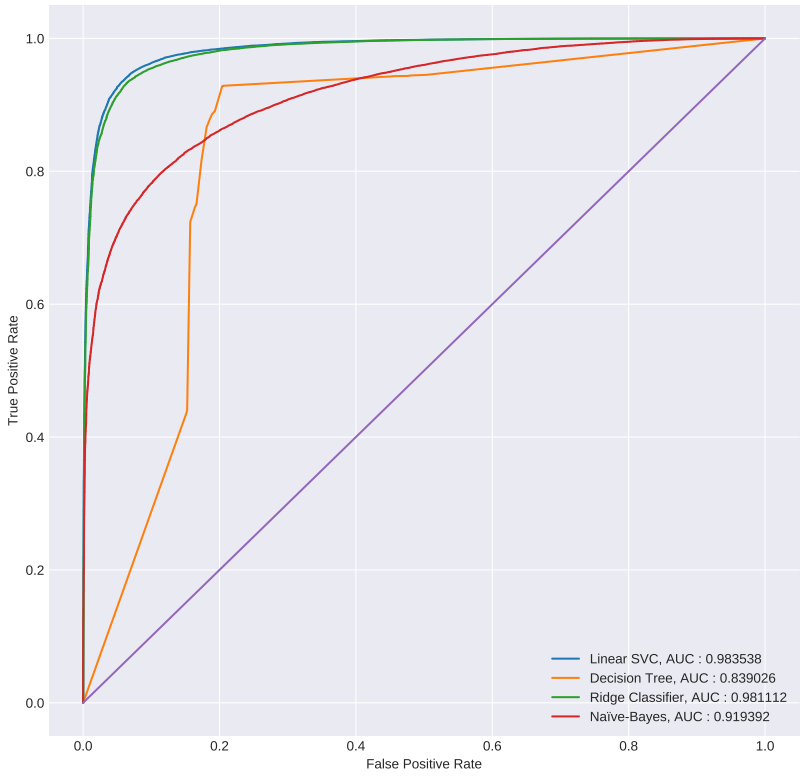


Figure 4.22: ROC curve for each model

when Naïve-Bayes only gets 85.3%.

In this case, the ROC curve (**Figure 4.22**) shows almost the same ranking of models, except for Decision Tree that is the last one, and Naïve-Bayes being juste above it. Confusion matrix (**Figure 4.23**) shows that Naïve-Bayes has a tendency of classifying fake news as being reliable. And the other hand, ridge classifier is the one that makes the least misclassification for reliable news, which is a good point.

Finally, there is the results of the models trained with SMOTE data augmentation. Using it shows little to no improvements. The only benefit is to balance a little bit the recall for Naïve-Bayes on fake and reliable news.

4.8 Conclusion

In this chapter we have analysed how traditional machine learning algorithms words on two different datasets, the second being imbalanced a data augmentation technique, called SMOTE, has been used in order to see if it improves the results.

We can conclude that in all the cases linear models are the ones that work the best, with a top accuracy of 61.7% on the **liar-liar corpus** using Ridge Classifier, and a top accuracy of 94.7% on the **Fake News Corpus** using linear svm. At the end, the result obtains on the second data set are really good, when those obtain on the first when are mitigated.

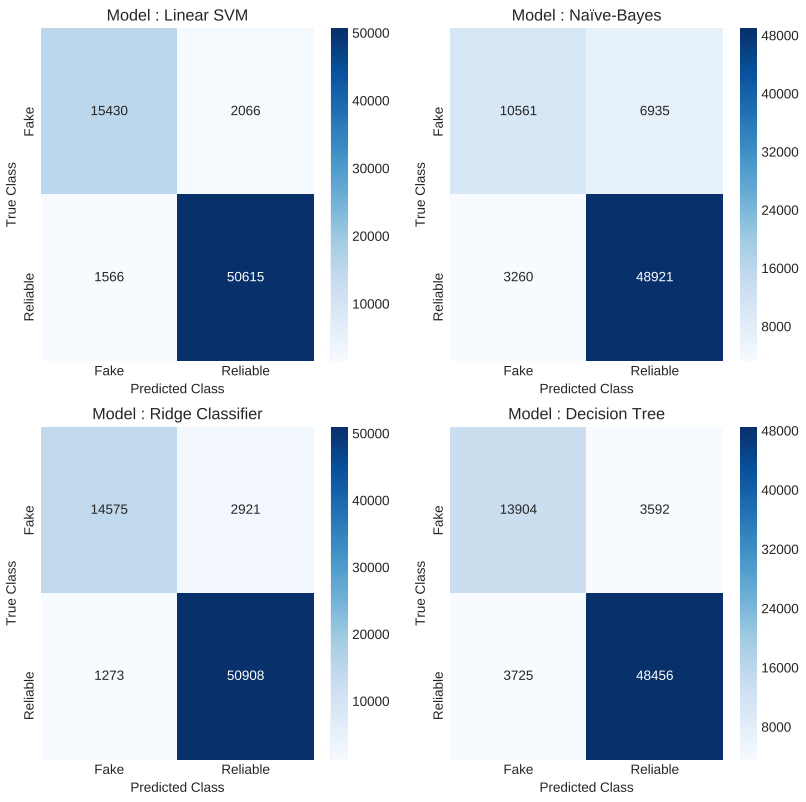


Figure 4.23: Confusion Matrix for Each models

	fake	reliable	accuracy	macro avg	weighted avg
f1-score	0.891373	0.962340	0.94407	0.926857	0.944520
precision	0.869960	0.970623	0.94407	0.920291	0.945346
recall	0.913866	0.954198	0.94407	0.934032	0.944070
support	17496.000000	52181.000000	0.94407	69677.000000	69677.000000

(a) Raw results of linear svm on **Fake News Corpus** when training using SMOTE

	fake	reliable	accuracy	macro avg	weighted avg
f1-score	0.714816	0.873538	0.824777	0.794177	0.833683
precision	0.604424	0.950521	0.824777	0.777472	0.863615
recall	0.874543	0.808091	0.824777	0.841317	0.824777
support	17496.000000	52181.000000	0.824777	69677.000000	69677.000000

(b) Raw results of Naïve-Bayes on **Fake News Corpus** when training using SMOTE

	fake	reliable	accuracy	macro avg	weighted avg
f1-score	0.877755	0.956129	0.935431	0.916942	0.936449
precision	0.836588	0.973317	0.935431	0.904953	0.938984
recall	0.923182	0.939537	0.935431	0.931360	0.935431
support	17496.000000	52181.000000	0.935431	69677.000000	69677.000000

(c) Raw results of Ridge Classifier on **Fake News Corpus** when training using SMOTE

	fake	reliable	accuracy	macro avg	weighted avg
f1-score	0.787226	0.921178	0.884969	0.854202	0.887542
precision	0.734992	0.946085	0.884969	0.840539	0.893079
recall	0.847451	0.897549	0.884969	0.872500	0.884969
support	17496.000000	52181.000000	0.884969	69677.000000	69677.000000

(d) Raw results of Decision tree on **Fake News Corpus** when training using SMOTE

Table 4.3: Results on **Fake News Corpus** when training with SMOTE.

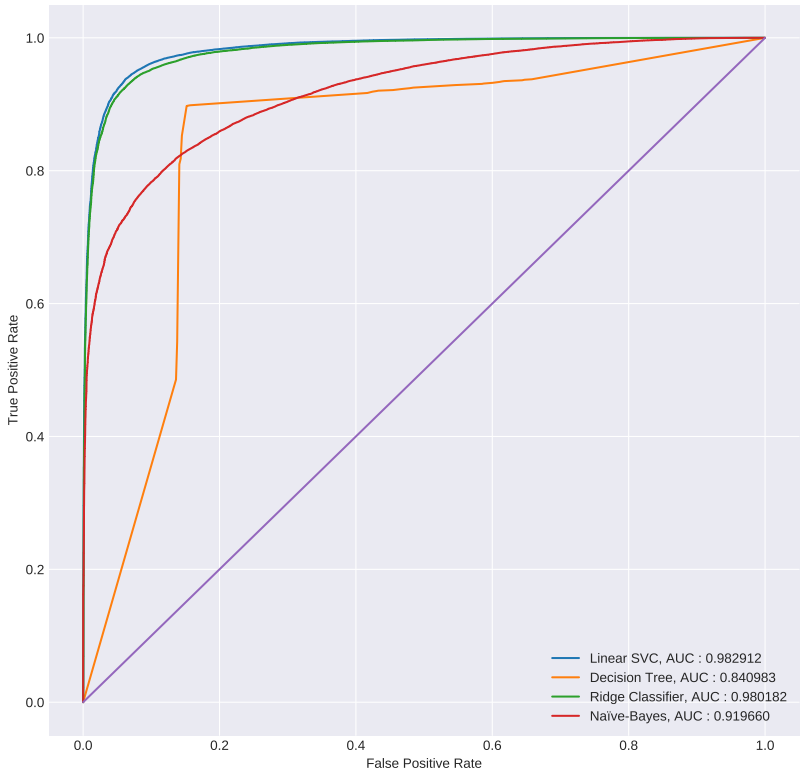


Figure 4.24: ROC curve for each model

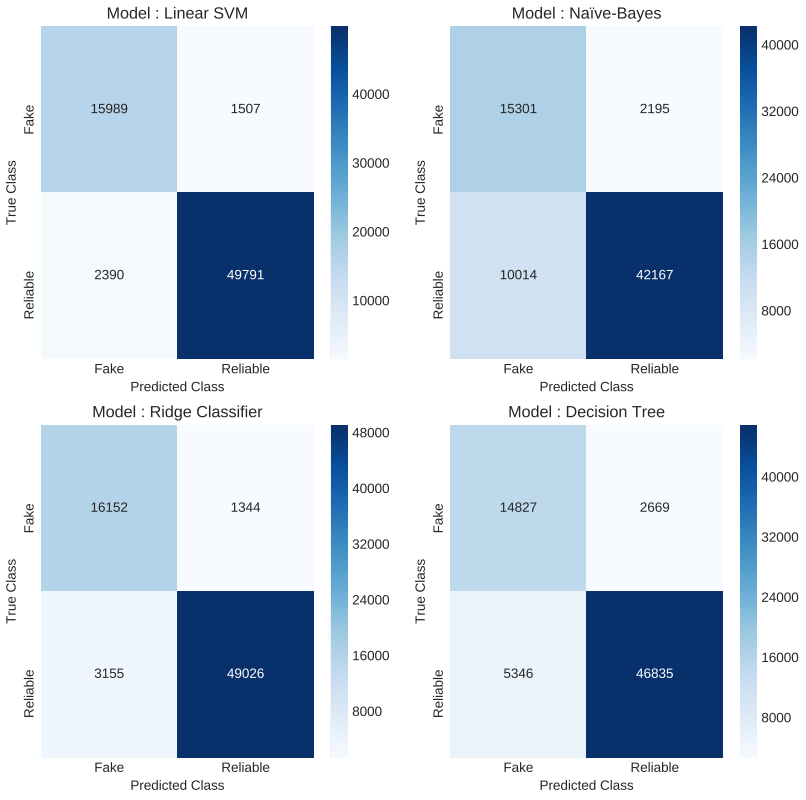


Figure 4.25: Confusion Matrix for each models

As explained earlier, it might be important to choose to model that makes the smaller misclassification rate on reliable news in order to avoid possible censorship and confusion matrix shows that in both case Ridge Classifiers is the ones that make the fewer errors in that case.
In addition, we have shown that Synthectic Minority Over Sampling Techniques acts as a regularizers, as it does improve performance when the penalization term in small on linear models.
In the next section, the focus will be put on trying to improve results on the **Liar-Liar corpus** as there is room for improvement and that the second dataset already as very good results. But models will still be trying on it for comparison.

Chapter 5

Attention Mechanism

5.1 Introduction

In this section, we will focus on the deep learning models, the first one being a bidirectional LSTM and the second one an attention layer is added to this LSTM. But it is need to use another text embedding in order to work with LSTM. Indeed, tf-idf create a sparse matrix with each row corresponding to a value for a given word. This means that the order of the words are lost. In order to solve this, word2vec[38] is used. It allows matching words to continuous vectors of a given size with interesting properties. Another method, which consists in making word embedding as tuning parameters will be used.

5.2 Text to Vectors

5.2.1 Word2Vec

Word2Vec comes in two fashions: continuous bag of words (CBOW) and skip gram. It is originally designed to predict a word given a context. For instance, given two previous words and the next two words, which word is the most likely to take place between them. But it appears that the hidden representation of these words works well as word embedding and has very interesting properties such that words with similar meaning have similar vector representation. It is also possible to perform arithmetic that captures information such as singular, plural or even capital and countries. For example, we have that $dog - dogs \approx cat - cats$ but also $Paris - France \approx Berlin - Germany$.
It is possible to visualize these relationships by using t-SNE for projecting high dimensions word vectors in 2D space. The results of various relationships can be seen at **Figure 5.1**.

How does it work?

As the original authors did not intend this kind of result, Xin Rong[39] did a good job explaining how it works. Let V be the size of the vocabulary and that there is only one word in the CBOW model, it give **Figure 5.2** models. Each word is encoded as a one-hot vector of size V. That means that it is a sparse vector full of zeros except for the position assigned to that word which is one. The hidden layer is computed as

$$\mathbf{h} = \mathbf{W}^{\mathbf{T}}\mathbf{x} \tag{5.1}$$

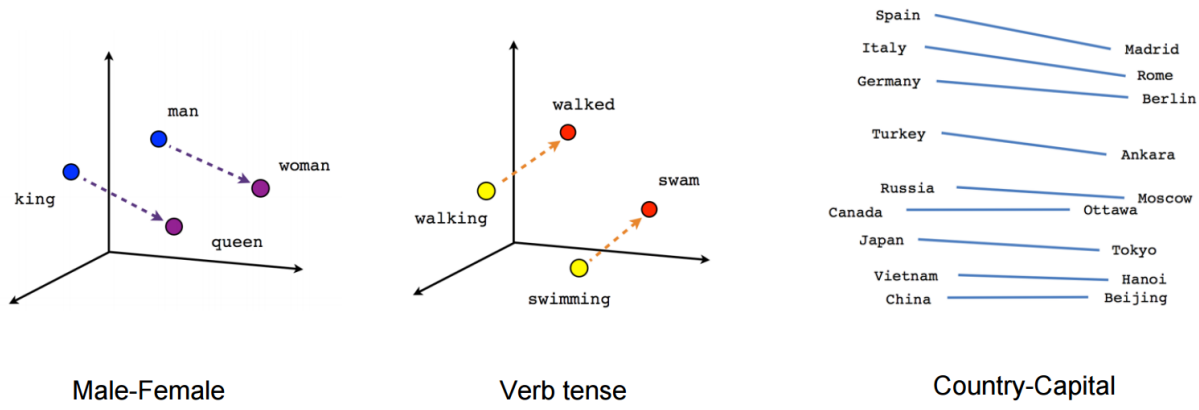


Figure 5.1: Relationships between different words with t-SNE dimensionality reduction.

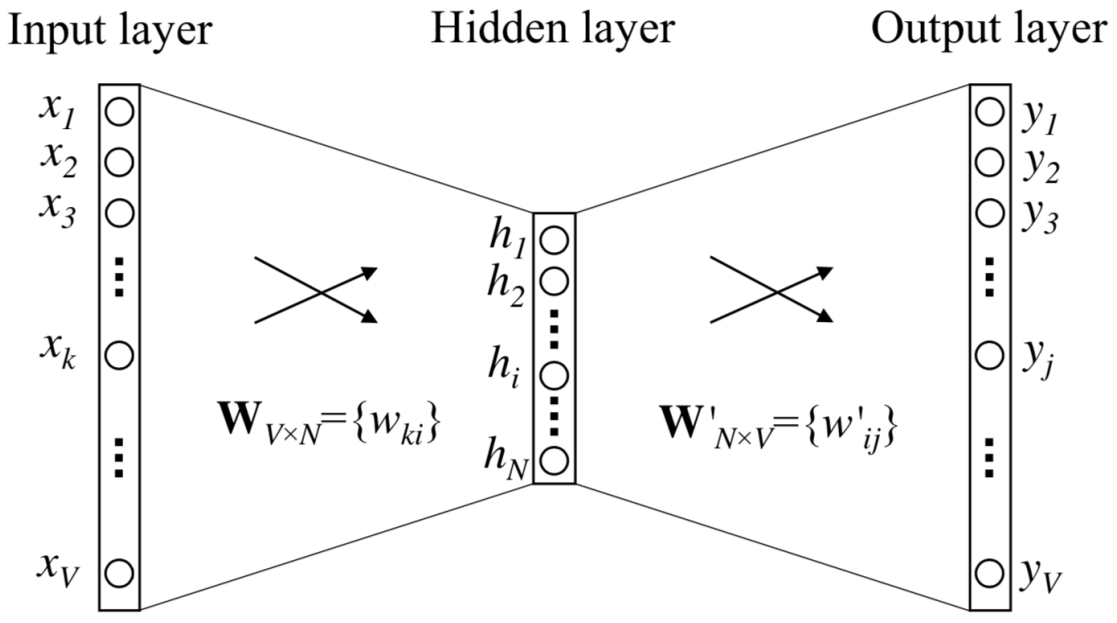


Figure 5.2: A simple CBOW model with only one word in the context

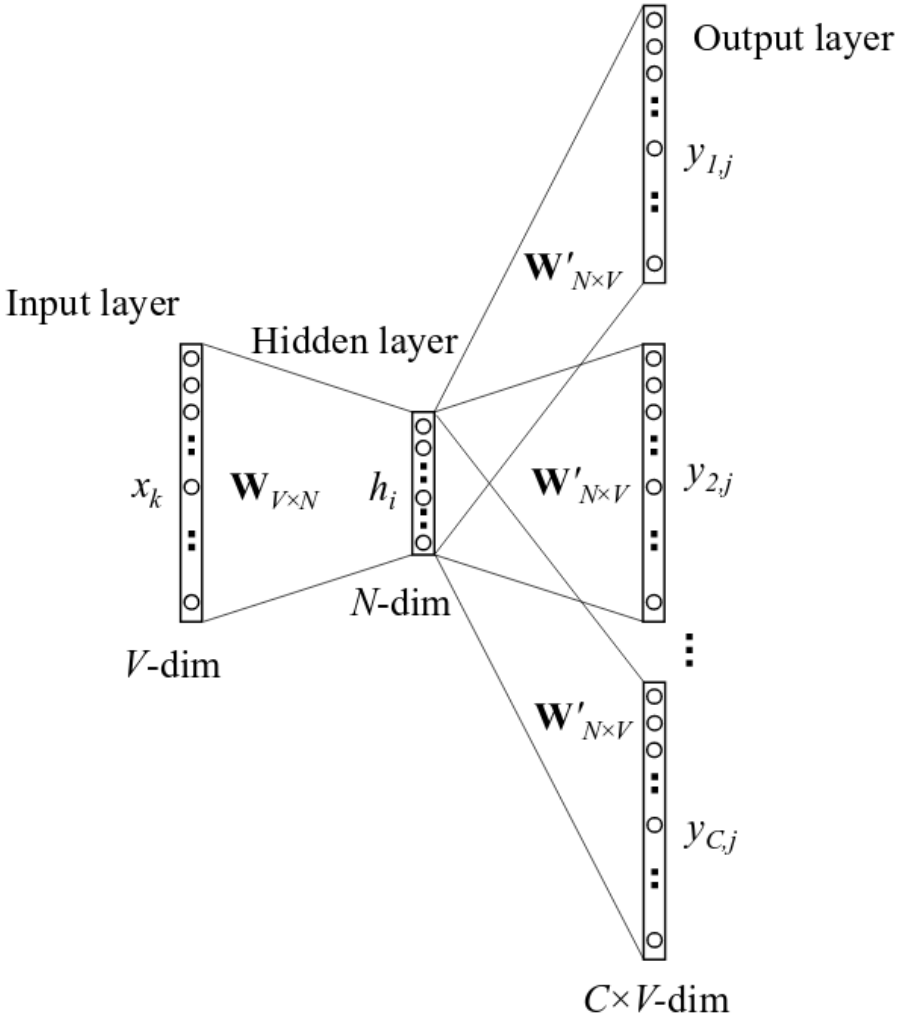


Figure 5.3: Skip-gram model with multiple outputs.

Where $\mathbf{W}^{\mathbf{V}\times\mathbf{N}}$ is the weight matrix to optimize over. The output layer values are computed as

$$\mathbf{Y} = \mathbf{W}'^{\mathbf{T}}\mathbf{h} \tag{5.2}$$

As before $\mathbf{W}'^{\mathbf{N}\times\mathbf{V}}$ is also a weight matrix to optimize. The loss can be computed as softmax cross entropy.
It is also possible to make the opposite: predicting the context given a single input word. This is the skip-gram model. In this case the loss becomes **Equation 5.3**.

$$E = -\sum_{c=1}^{C} u_{j_c^*} + C \cdot \sum_{j'=1}^{V} \exp(u_{j'}) \tag{5.3}$$

j_c^* is the index of the cth output context word and $u_{j_c^*}$ is the score of the jth word in the vocabulary for the cth context word. Finally, the embedding that is used are the value of the hidden layers produced for a given word.

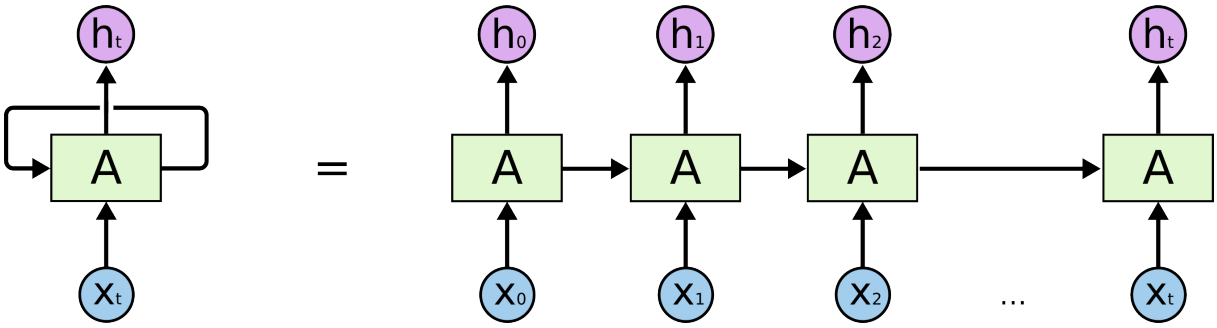


Figure 5.4: Unrolled RNN (Understanding LSTM Networks, `https://colah.github.io/posts/2015-08-Understanding-LSTMs/`)

5.3 LSTM

LSTM or Long Short Term Memory[8] is a kind of recurrent neural network that fits well to temporal or sequential input such as texts. A RNN is a type of neural network where the hidden state is fed in a loop with the sequential inputs. There are usually shown as unrolled version of it (**Figure 5.4**). Each of the X_i being one value in the sequence.
In this case, X_i values are word vectors. There are two possibilities, either use pre-trained vector with word2vec or make X_i inputs a parameter to learn in the same way as it works for the Word2Vec algorithm, having a one-hot encoding of the word and a matrix of weights to tune. Each method will be used.
Recurrent Neural Networks do not works very well with long-term dependencies, that is why LSTM have been introduced. It is made of an input gate, an output gate and a forget gate that are combined in **Equation 5.4**.

$$f_t = \sigma_g(\mathbf{W}_f x_t + \mathbf{U}_f h_{t-1} + b_f) \tag{5.4}$$
$$i_t = \sigma_g(\mathbf{W}_i x_t + \mathbf{U}_i h_{t-1} + b_i) \tag{5.5}$$
$$o_t = \sigma_g(\mathbf{W}_o x_t + \mathbf{U}_o h_{t-1} + b_o) \tag{5.6}$$
$$c_t = f_t \circ c_{t-1} + i_t \circ \sigma_c(\mathbf{W}_c x_t + \mathbf{U}_c h_{t-1} + b_c) \tag{5.7}$$
$$h_t = o_t \circ \sigma_h(c_t) \tag{5.8}$$

Figure 5.5 shows how it works. A bidirectional LSTM works the same way, but the input is fed in the two directions, from the start to the end and from the end to the start.

5.4 Attention Mechanism

Attention mechanism[1, 21] adds an extra layer between LSTM outputs and the final output of the network. It merges word-level features into sentence features using a weight vector.

Outputs sequence of the LSTM is summed element-wise in order to merge them. We have that $h_i = [\overrightarrow{h_i} + \overleftarrow{h_i}]$, $\overrightarrow{h_i}$ and $\overleftarrow{h_i}$ begin the outputs i of sequence in each direction as show at **Figure 5.6**.
Lets H be a matrix of the concatenation of all the h_i,

$$H = [h_1, h_2, ..., h_T] \tag{5.9}$$

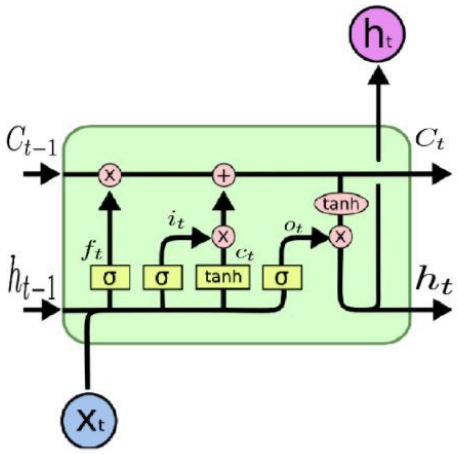


Figure 5.5: LSTM gates,
`https://hackernoon.com/understanding-architecture-of-lstm-cell-from-scratch-with-code-8da40f0b71f4)`

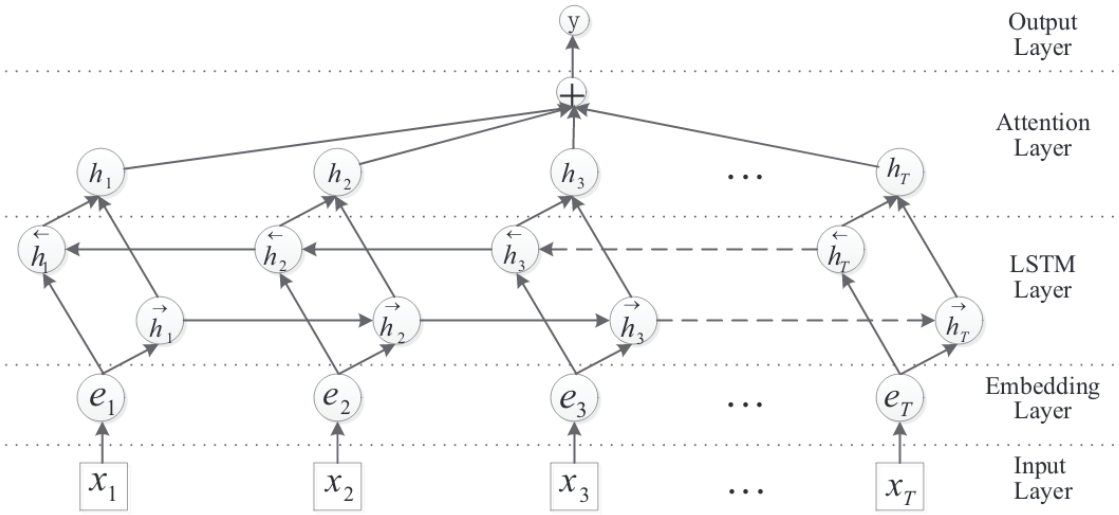


Figure 5.6: Bidirectional LSTM Model With Attention

Where T is the sequence length. Then we define

$$M = \tanh(H) \tag{5.10}$$

$$\alpha = softmax(w^T M) \tag{5.11}$$

$$r = H\alpha^T \tag{5.12}$$

Finally, we compute $h^* = \tanh(r)$. For the classification, it uses a softmax classifier as $\hat{p}(y|S) = softmax(W^S h^* + b)$. Originally the loss function is the negative log likelihood, but as in this case it is a binary classification I used binary cross entropy.

5.5 Results

5.5.1 Methodology

In order to train the models and perform hyper parameters optimization grid search have been used when it was possible (on the liar-liar dataset) and knwoleadge acquired there have been used in order to tune parameters for the networks on the **Fake News Corpus**. In addition, in order to find the best parameters among all tested with gird search, for each metric, the training epochs having the highest validation value for those metrics have been chosen.
All the models have been trained using adam optimizer and initialized using a normal distribution for the weights.
As SMOTE cannot be used on the **Fake News Corpus** dues to the size of the corpus, in order to rebalance the dataset the minority class have been over sampled by feeding multiple times the same input by looping through them.

5.5.2 Liar-Liar dataset results

As explained earlier, both models have been trained using different embedding: the first one being pre-trained word2vec vectors of size 300 and the second one being a tunable parameter with different embedding size.

LSTM

When it comes to LSTM trained on liar-liar dataset, it simply does not works. It classifies almost all the texts as being from the same class. Although, it reaches a good score on the training data, it does not manage to generalize correctly. **Figure 5.7** shows the recall, precision and f1-score and loss for training and testing set of the best models for the LSTM using word2vec. We can see that even if the training score increase, the testing values oscillate.

When training the models with word embedding as tunable parameters, the results slightly improve, with an average precision between 55% and 60%. This can be seen at **Figure 5.8**.
The training was stopped after 200 iterations because the validation score was not improving anymore.

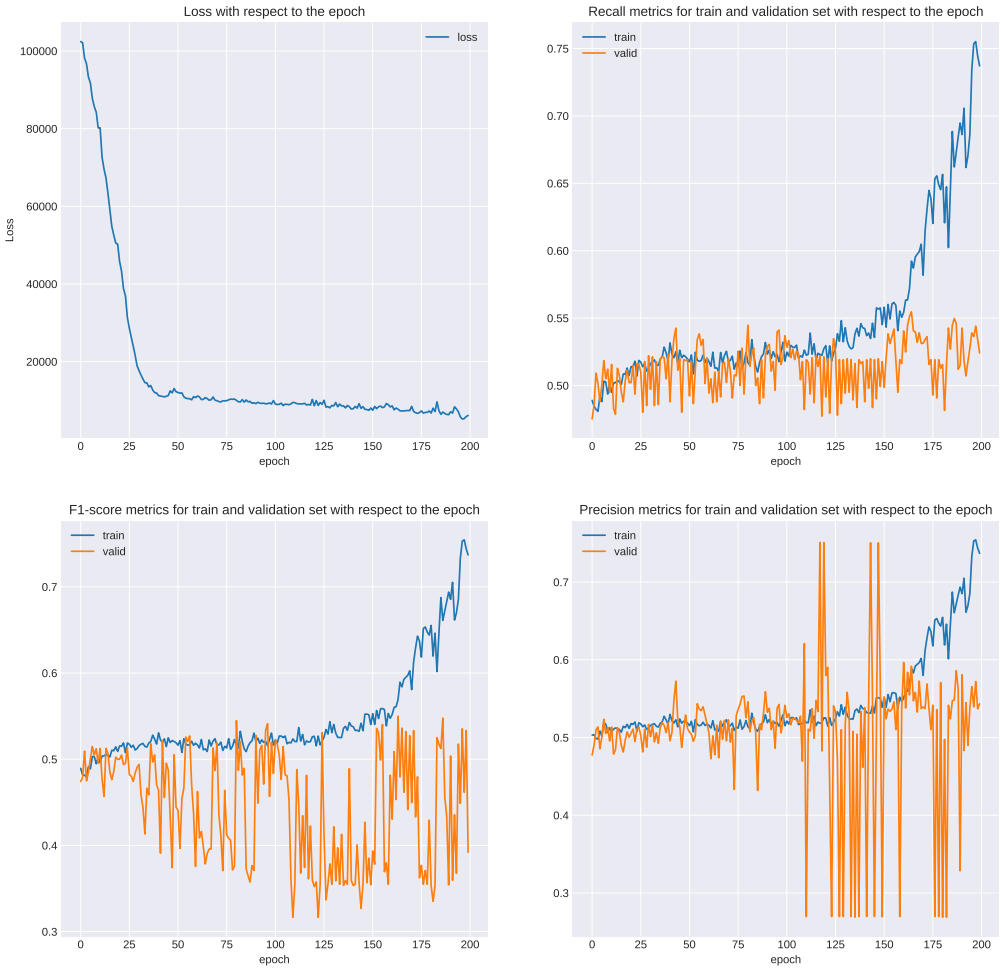


Figure 5.7: Best LSTM With word2vec

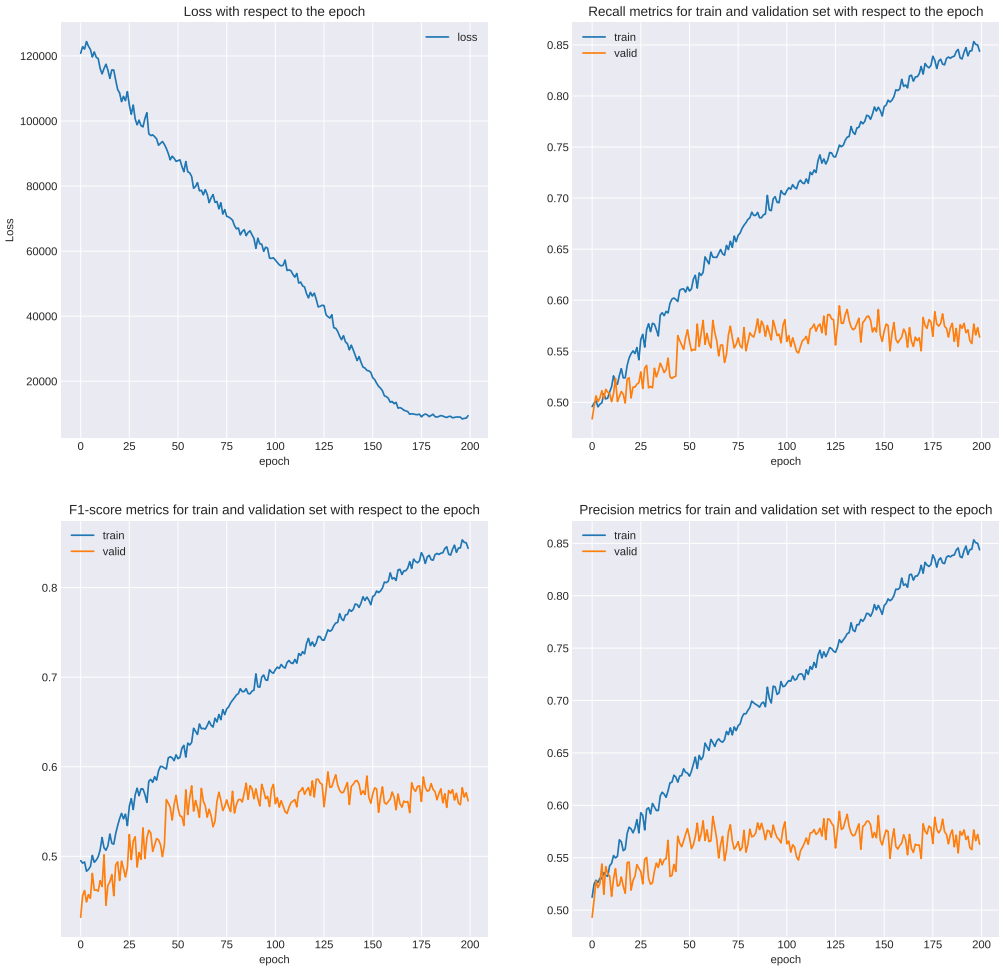


Figure 5.8: Best LSTM with word embedding as tunable parameters.

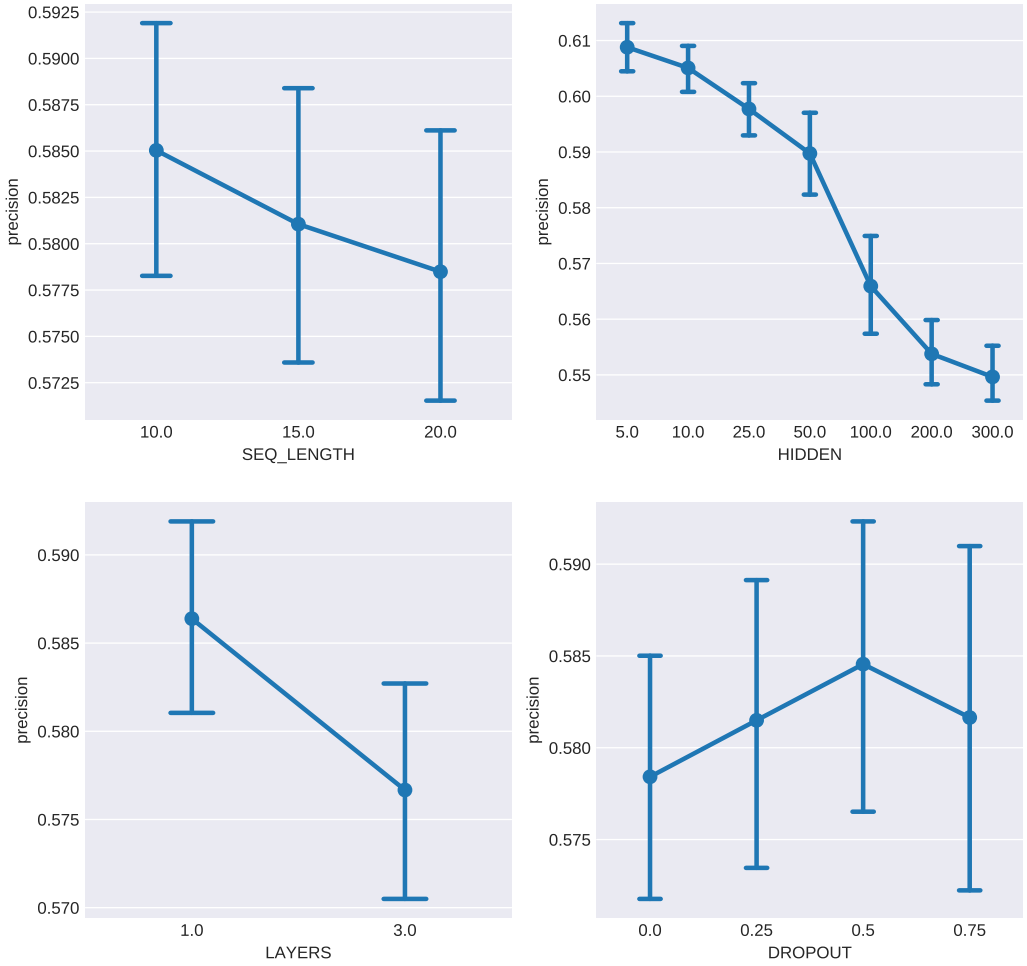


Figure 5.9: Confidence Interval of Precision for Each Parameter Value

5.5.3 Attention Mechanism

We could think that adding an extra layer to such a bad model would not make any improvement and be useless, but adding an attention layer does improve a little the results. When using word2vec embedding, the best epoch reaches up to 62.9595% of average accuracy, which is better than the simple LSTM. Because there are a lot of models with different parameters, it is interesting to look at the distribution of the results for the best epochs by fixing parameters one by one. **Figure 5.9** shows the 95% confidence interval for precision for a fixed parameter.

It shows that it is better to use fewer hidden units in the model, and only a single layer. The sequence length has a very small impact on the precision. Actually, the best model uses a sequence length of 20.

The precision of the different models range from 53% to 63% (**Figure 5.10**.) The training plot of the model that reaches the maximum precision can be seen at **figure 5.11**. It

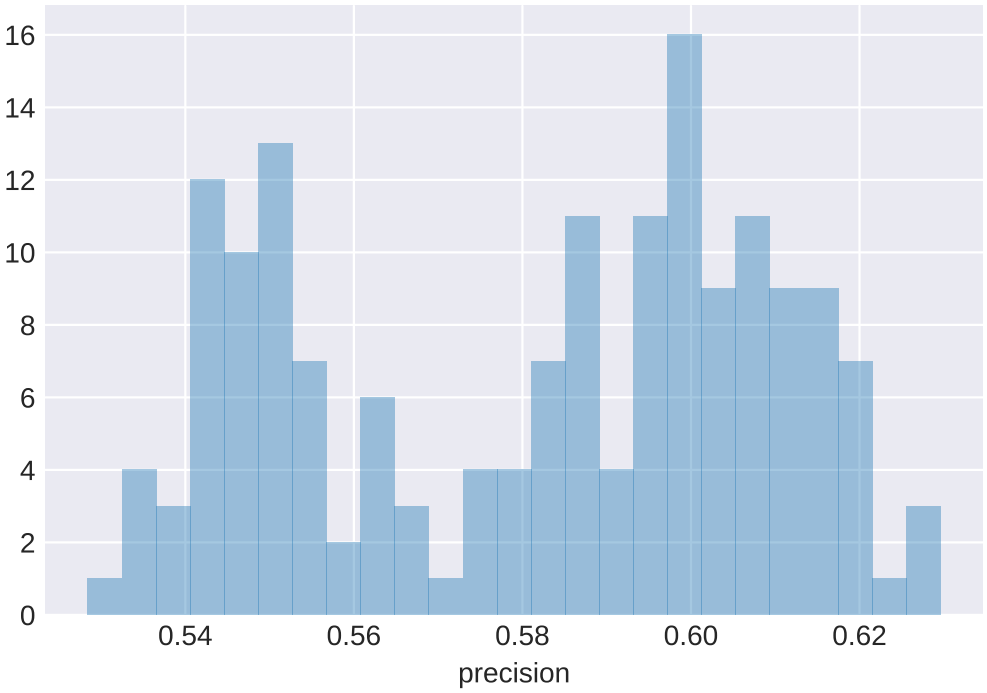


Figure 5.10: Distribution of the precision of best epochs for all the models trained with word2vec embedding.

shows that after the 25th iteration, the validation values start to decrease, which is a sign of overfitting.

Finally, there is the models where the embedding is a tunable parameter. The **Figure 5.12** shows that in this case, the longer the sequence the better, and that as before using few hidden units perform better. In this case, variation has a wider range than when using word2vec. There are a few models that have top precision higher than 75%, but looking at the training plot (**Appendix B, Figure B.1**) shows that the model does not perform well. Because in particular case precision in not a good indicator of how well a model perform, f1-score will be used instead, as it is a balance between precision and recall.

And the best f1-score obtained is 0.55384, which is quite smaller than the 0.63 for the model using word2vec. The training plot is at **Figure 5.13**. We can see that there is still room for improvement, the next step is to see what happens when training on more epochs.

Training on 1000 epochs rather than 200 does not improve validation score, but it does for training (**Figure 5.14**).

5.5.4 Result Analysis

The previous section shows a few things

- LSTMs do not work well,

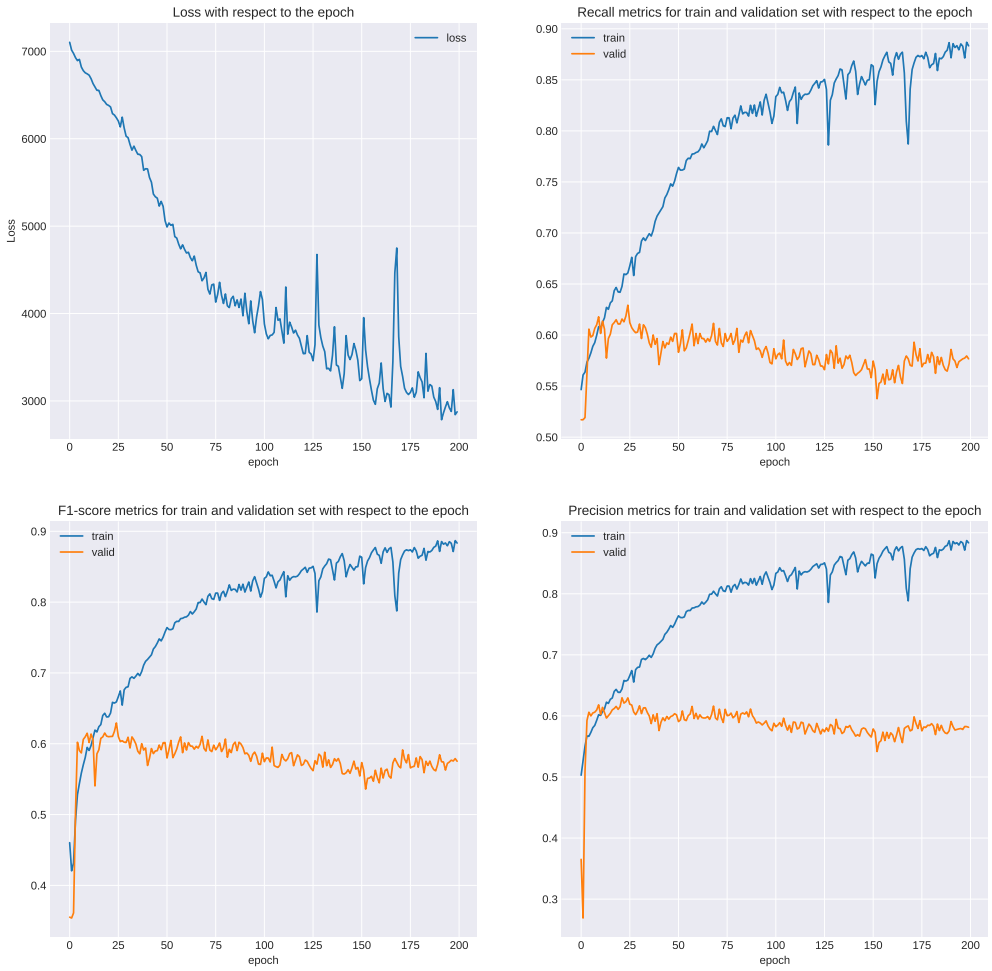


Figure 5.11: Training and validation of the model with top precision trained with word2vec embedding.

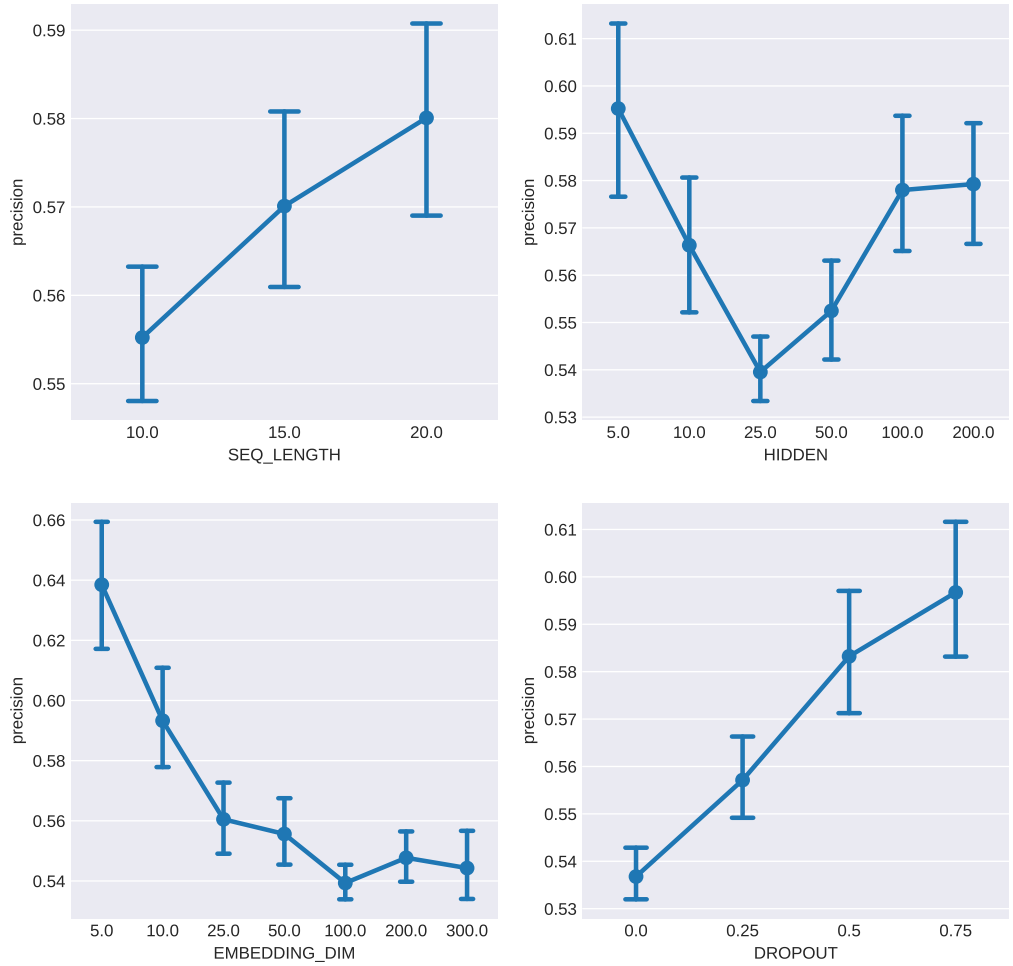


Figure 5.12: Training and Validation of the Model With top Precision

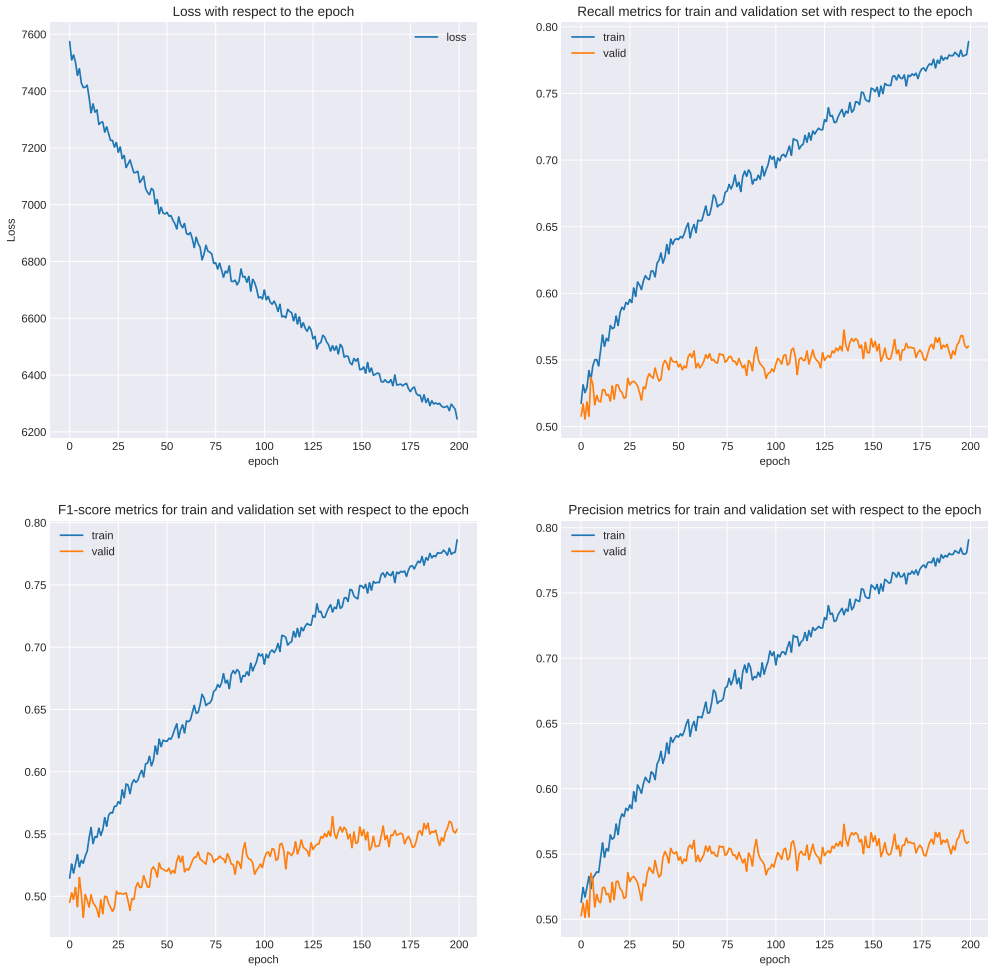


Figure 5.13: Training and validation of the model with top f1-score

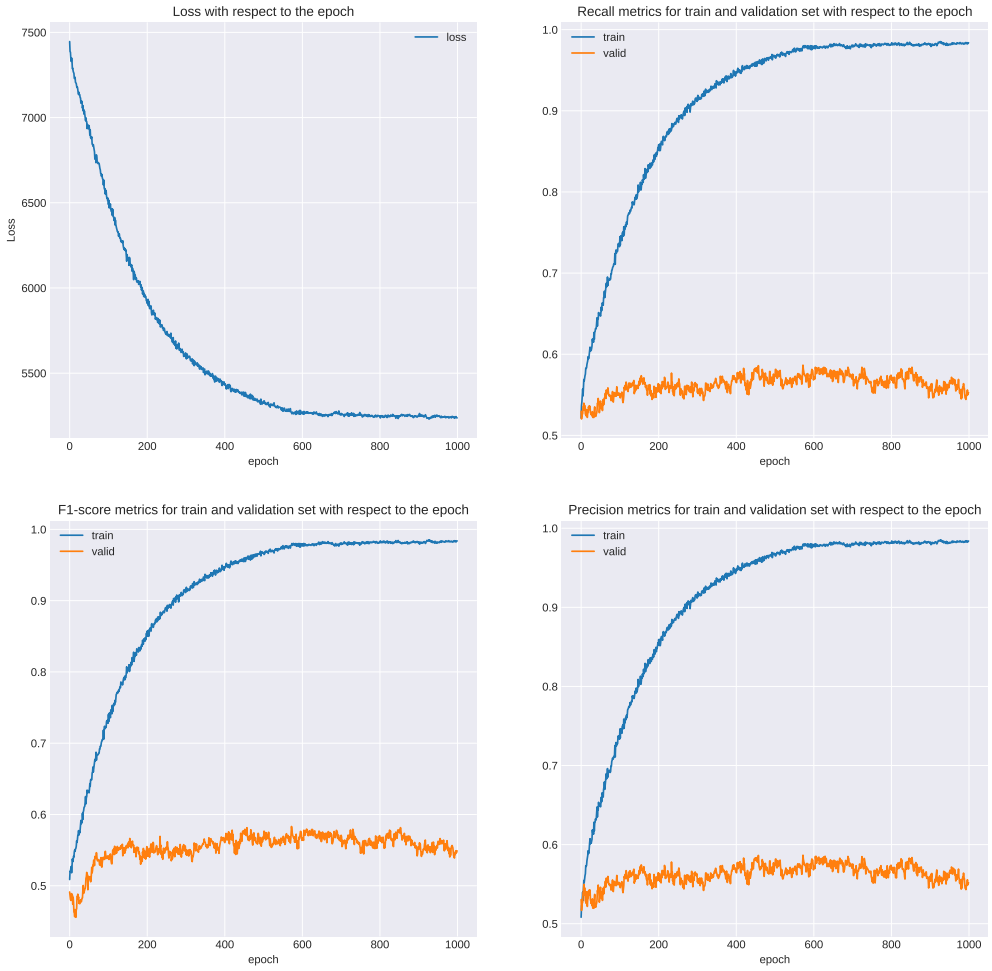


Figure 5.14: Training on 1000 epochs rather than 200

- Adding attention layer improve LSTM results,
- Using word2vec rather than training the embedding gives better results.

It also shows that despite reaching a very good precision, recall and f1-score on the training set it does not perform well on the validation set. This is a sign of overfitting. In order to avoid this, multiple methods have been applied without showing any improvement.
The following methods have been applied:

- Dropout[40],
- batch-normlaziation[41],
- reducing network capacity (fewer hidden layers, lower embedding dimensions, less training parameters with word2vec),
- Early stopping of training.

The highest gain was from using word2vec embedding. This significantly reduces the amount of training parameters, secondly dropout also helped a little.

5.5.5 Testing

The same way as in **Chapter 4**, the models will be trained on the parameters that produced the best results on the training set, and trained on training and validation set, and tested on testing set.
The parameters used for training are given at **Table 5.1**. The results for all four models

model	embedding size	Sequence Length	num hiddens	dropout	Early Stop
LSTM	300	10	50	0.75	126
LSTM + word2vec	300	10	50	0.0	160
Attention	10	20	10	0.75	400
Attention + word2vec	300	20	5	0.75	25

Table 5.1: Parameters used for training

are given at **Table 5.2**. It shows that the model that works the best is attention network using word2vec embedding, with an accuracy of 61%, which is equivalent to ridge classifiers and linear svm. The three other models do not perform well, all having a average precision around 55%, which is close to being a random classifier.

5.6 Attention Mechanism on fake news corpus

5.6.1 Model Selection

The two models using word2vec embedding have shown to work better than their counterparts, this why only these two methods will be tested on **Fake News Corpus** for comparison as very good results have already been obtained.
It shows out that in this case LSTMs works better than Attention Mechanism, but as in previous section does not reach machine learning results.

	fake	reliable	accuracy	macro avg	weighted avg
f1-score	0.440574	0.649551	0.569061	0.545062	0.558340
precision	0.508274	0.599526	0.569061	0.553900	0.559698
recall	0.388788	0.708683	0.569061	0.548736	0.569061
support	1106.000000	1428.000000	0.569061	2534.000000	2534.000000

(a) Simple LSTM

	fake	reliable	accuracy	macro avg	weighted avg
f1-score	0.481724	0.623040	0.563536	0.552382	0.561361
precision	0.500000	0.606906	0.563536	0.553453	0.560245
recall	0.464738	0.640056	0.563536	0.552397	0.563536
support	1106.000000	1428.000000	0.563536	2534.000000	2534.000000

(b) LSTM + word2vec

	fake	reliable	accuracy	macro avg	weighted avg
f1-score	0.486636	0.615597	0.560379	0.551116	0.559310
precision	0.496241	0.606803	0.560379	0.551522	0.558546
recall	0.477396	0.624650	0.560379	0.551023	0.560379
support	1106.000000	1428.000000	0.560379	2534.000000	2534.000000

(c) Attention network

	fake	reliable	accuracy	macro avg	weighted avg
f1-score	0.511397	0.676721	0.610892	0.594059	0.604563
precision	0.565789	0.636252	0.610892	0.601021	0.605497
recall	0.466546	0.722689	0.610892	0.594618	0.610892
support	1106.000000	1428.000000	0.610892	2534.000000	2534.000000

(d) Attention Network + word2vec

Table 5.2: Results for the differents models trained with parameters given at **Table 5.1**.

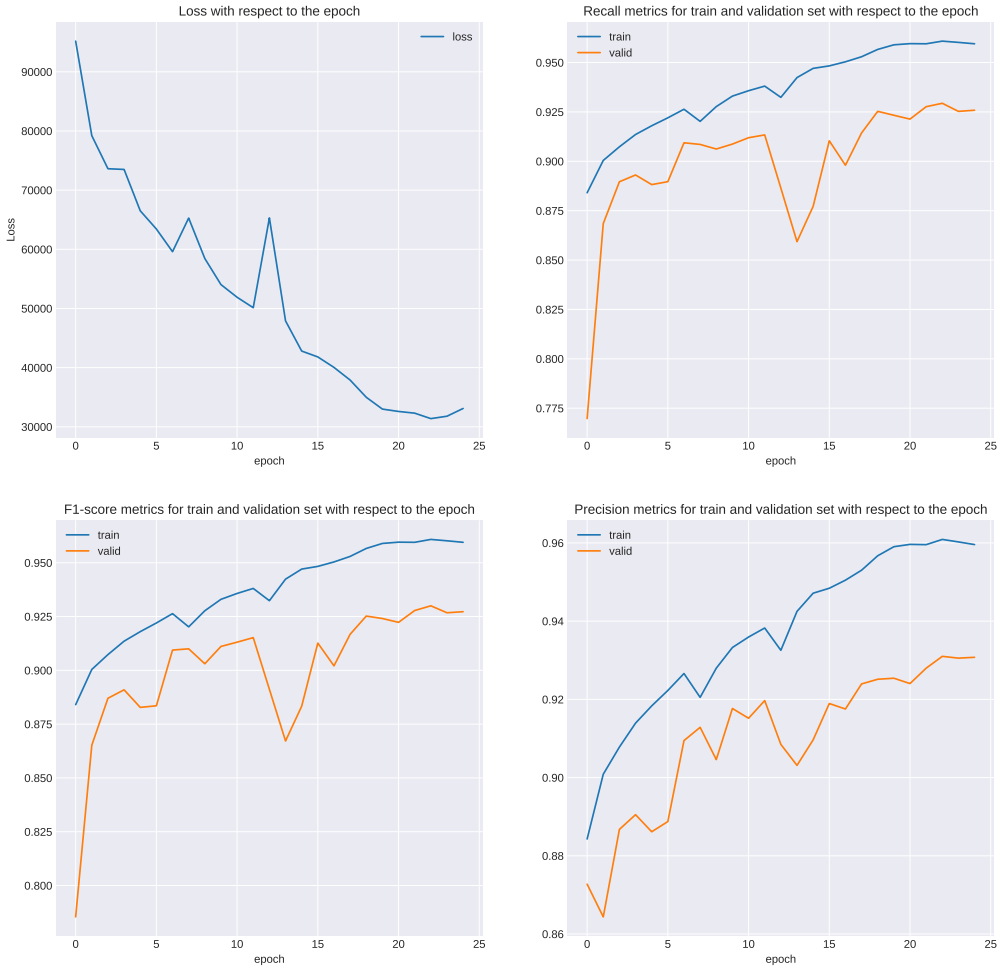


Figure 5.15: Training plots of the best LSTM using word2vec embedding.

The best LSTM obtained use sequence of 200 words and 200 hidden layers, with an average precision of 0.929376 on the validation set. The training plots of this particular model are shown at **Figure 5.15**.

In the case of attention mechanism, training on the **Fake News Corpus** has shown to be harder than on the **Liar-Liar Corpus** as using too large learning rate would lead to oscillating loss and too small learning rate lead to halting the loss decrease. This can be seen at **Appendix B.2**.
The same parameters as for the LSTM will be used for training the Attention Network. Its training plot is available at **Figure 5.16**. The final results are given at **Table 5.3**. It shows that for the same parameters LSTM works better than Attention Network on this particular dataset. It shows that LSTM place below Linear SVM and Ridge Classifier and above Decision Tree and Naïve-Bayes in terms of accuracy. It is likely to be possible to reach results as well as LSTM or even better for the Attention Network, but due to

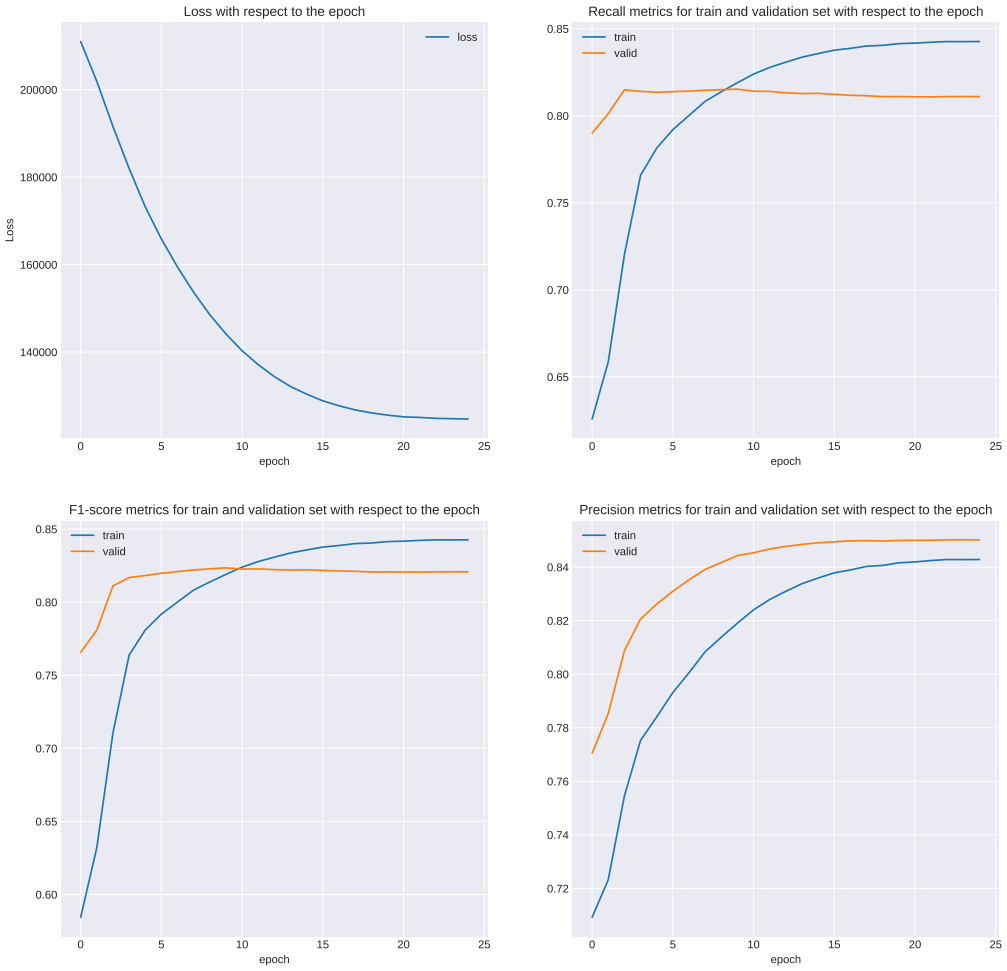


Figure 5.16: Training plots of the best attention network using word2vec embedding.

	fake	reliable	accuracy	macro avg	weighted avg
f1-score	0.856568	0.947577	0.923217	0.902073	0.924724
precision	0.806655	0.969503	0.923217	0.888079	0.928611
recall	0.913066	0.926621	0.923217	0.919843	0.923217
support	17496.000000	52181.000000	0.923217	69677.000000	69677.000000

(a) LSTM + word2vec results on **Fake News Corpus**

	reliable	fake	accuracy	macro avg	weighted avg
f1-score	0.850296	0.687493	0.797566	0.768894	0.809416
precision	0.952876	0.561344	0.797566	0.757110	0.854562
recall	0.767655	0.886774	0.797566	0.827215	0.797566
support	52181.000000	17496.000000	0.797566	69677.000000	69677.000000

(b) Attention Network + word2vec on **Fake News Corpus**

Table 5.3: Final result on testing set for LSTM and attention network using word2vec.

technical and time constraints I was not able to experiment further. For instance, using longer sequence length and more hidden units with a smaller learning rate might have overcome this problem.

5.7 Conclusion

In this chapter I have investigated how state-of-the-art deep learning models work on fake news detection, and it shows that for the particular case of fake news detection it does not outperform traditional machine learning methods. I have also made some addition to the original model that improves the performances by a few percent by replacing the tunable word embedding by constant one using word2vec. It shows out that it helps reduce overfitting and increase result on the testing set.
A hypothesis to explain why these two deep learning methods do not works as well as machine learning methods is the fact that in this case text are required to be the same size. Which means that some of them require some padding and the other are srunk. In the second case, information is lost.
In addition, it shows that **Liar-Liar Corpus** is hard to work on, with 60% precision, when **Fake News Corpus** still have good results.

Chapter 6

Conclusion

6.1 Result analysis

Some hypotheses can be made on why same models works very well on one dataset and does not work well on the other one. The first thing we can think of is that the original hypothesis on different styles of writing between fake and reliable news is only verified in one dataset, the **Fake News Corpus**, and it is the most logical one, as these texts are coming from online newspapers (or pretending to be), and thus capitalize on advertisements for making money. The second dataset, **Liar-Liar Corpus** is described by its authors as a collection a short sentence coming from various contexts such as political debate, interviews, TV ads and so on, thus it induces a lot of variety in writing style. For instance, it contains a transcription of vocal messages, which have in essence a different style from written one.
The data exploration chapter had already given an insight about this fact, as 2D data projection of the **Liar-Liar Corpus** shows no clear sign of separation, when **Fake News Corpus** shows one at the first look.

6.2 Future works

Basing fake news detection only on supervised models on text have shown not to be enough in all the cases. In order to solve this problem, most of the research focus on additional information such as author information. I think the most successful approach would be automatic fact checking model, that is, compelling the model with some kind of knowledge base, the purpose of the model would then be to extract information for the text and verify the information in the database. The problem with this approach would be that the knowledge base would need to be constantly and manually update to stay up to date.

Bibliography

[1] Peng Zhou, Wei Shi, Jun Tian, Zhenyu Qi, Bingchen Li, Hongwei Hao, and Bo Xu. Attention-based bidirectional long short-term memory networks for relation classification. In *Proceedings of the 54th Annual Meeting of the Association for Computational Linguistics (Volume 2: Short Papers)*, pages 207–212, Berlin, Germany, August 2016. Association for Computational Linguistics.

[2] Hunt Allcott and Matthew Gentzkow. Social media and fake news in the 2016 election. In *Journal of Economic Perspective*, volume 31, 2017.

[3] Jeffrey Gottfried and Elisa Shearer. *News Use Across Social Medial Platforms 2016*. Pew Research Center, 2016.

[4] Craig Silverman and Lawrence Alexander. How teens in the balkans are duping trump supporters with fake news. *Buzzfeed News*, 3, 2016.

[5] Rong-En Fan, Kai-Wei Chang, Cho-Jui Hsieh, Xiang-Rui Wang, and Chih-Jen Lin. Liblinear: A library for large linear classification. *J. Mach. Learn. Res.*, 9:1871–1874, 2008.

[6] Stephen Robertson. Understanding inverse document frequency: On theoretical arguments for idf, 2004.

[7] Harry Zhang. The Optimality of Naive Bayes. page 6.

[8] Sepp Hochreiter and Jürgen Schmidhuber. Long short-term memory. *Neural Computation*, 9:1735–1780, 1997.

[9] Kai Shu, Amy Sliva, Suhang Wang, Jiliang Tang, and Huan Liu. Fake news detection on social media: A data mining perspective. *ACM SIGKDD Explorations Newsletter*, 19(1):22–36, 2017.

[10] WWF. Wwf 10yearschallenge, 2019.

[11] David M Blei, Andrew Y Ng, and Michael I Jordan. Latent dirichlet allocation. *Journal of machine Learning research*, 3(Jan):993–1022, 2003.

[12] Julio CS Reis, André Correia, Fabrício Murai, Adriano Veloso, Fabrício Benevenuto, and Erik Cambria. Supervised learning for fake news detection. *IEEE Intelligent Systems*, 34(2):76–81, 2019.

[13] Vernica Prez-Rosas, Bennett Kleinberg, Alexandra Lefevre, and Rada Mihalcea. Automatic detection of fake news.

[14] James W Pennebaker, Martha E Francis, and Roger J Booth. Linguistic inquiry and word count: Liwc 2001. *Mahway: Lawrence Erlbaum Associates*, 71(2001):2001, 2001.

[15] Natali Ruchansky, Sungyong Seo, and Yan Liu. Csi: A hybrid deep model for fake news detection. In *Proceedings of the 2017 ACM on Conference on Information and Knowledge Management*, pages 797–806. ACM, 2017.

[16] Eugenio Tacchini, Gabriele Ballarin, Marco L. Della Vedova, Stefano Moret, and Luca de Alfaro. Some like it hoax: Automated fake news detection in social networks.

[17] James Thorne, Mingjie Chen, Giorgos Myrianthous, Jiashu Pu, Xiaoxuan Wang, and Andreas Vlachos. Fake news stance detection using stacked ensemble of classifiers. In *Proceedings of the 2017 EMNLP Workshop: Natural Language Processing meets Journalism*, pages 80–83, 2017.

[18] Mykhailo Granik and Volodymyr Mesyura. Fake news detection using naive bayes classifier. In *2017 IEEE First Ukraine Conference on Electrical and Computer Engineering (UKRCON)*, pages 900–903. IEEE, 2017.

[19] Yang Yang, Lei Zheng, Jiawei Zhang, Qingcai Cui, Zhoujun Li, and Philip S. Yu. Ti-cnn: Convolutional neural networks for fake news detection.

[20] Yaqing Wang, Fenglong Ma, Zhiwei Jin, Ye Yuan, Guangxu Xun, Kishlay Jha, Lu Su, and Jing Gao. Eann: Event adversarial neural networks for multi-modal fake news detection. In *Proceedings of the 24th acm sigkdd international conference on knowledge discovery & data mining*, pages 849–857. ACM, 2018.

[21] Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N. Gomez, Lukasz Kaiser, and Illia Polosukhin. Attention is all you need. In *NIPS*, 2017.

[22] Shuai Li, Wanqing Li, Chris Cook, Ce Zhu, and Yanbo Gao. Independently recurrent neural network (indrnn): Building a longer and deeper rnn.

[23] Zichao Yang, Diyi Yang, Chris Dyer, Xiaodong He, Alex Smola, and Eduard Hovy. Hierarchical Attention Networks for Document Classification. In *Proceedings of the 2016 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies*, pages 1480–1489, San Diego, California, 2016. Association for Computational Linguistics.

[24] Takeru Miyato, Andrew M. Dai, and Ian Goodfellow. Adversarial Training Methods for Semi-Supervised Text Classification. *arXiv:1605.07725 [cs, stat]*, May 2016. arXiv: 1605.07725.

[25] Yoon Kim. Convolutional Neural Networks for Sentence Classification. *arXiv:1408.5882 [cs]*, August 2014. arXiv: 1408.5882.

[26] Kamran Kowsari, Mojtaba Heidarysafa, Donald E. Brown, Kiana Jafari Meimandi, and Laura E. Barnes. RMDL: Random Multimodel Deep Learning for Classification. *Proceedings of the 2nd International Conference on Information System and Data Mining - ICISDM '18*, pages 19–28, 2018. arXiv: 1805.01890.

[27] Quoc V. Le and Tomas Mikolov. Distributed representations of sentences and documents.

[28] David R. Karger, Sewoong Oh, and Devavrat Shah. Iterative learning for reliable crowdsourcing systems. In J. Shawe-Taylor, R. S. Zemel, P. L. Bartlett, F. Pereira, and K. Q. Weinberger, editors, *Advances in Neural Information Processing Systems 24*, pages 1953–1961. Curran Associates, Inc., 2011.

[29] Maciej Szpakowski. Fake news corpus. `https://github.com/several27/FakeNewsCorpus`. Accessed: 2018-10.

[30] William Yang Wang. "liar, liar pants on fire": A new benchmark dataset for fake news detection.

[31] Radim Řehůřek and Petr Sojka. Software Framework for Topic Modelling with Large Corpora. In *Proceedings of the LREC 2010 Workshop on New Challenges for NLP Frameworks*, pages 45–50, Valletta, Malta, May 2010. ELRA. `http://is.muni.cz/publication/884893/en`.

[32] Steven Bird, Ewan Klein, and Edward Loper. *Natural Language Processing with Python*. O'Reilly Media, 2009.

[33] Karen Spärck Jones. A statistical interpretation of term specificity and its application in retrieval. *Journal of documentation*, 2004.

[34] Laurens van der Maaten and Geoffrey Hinton. Visualizing data using t-sne. *Journal of Machine Learning Research*, (9):2579–2605, 2008.

[35] F. Pedregosa, G. Varoquaux, A. Gramfort, V. Michel, B. Thirion, O. Grisel, M. Blondel, P. Prettenhofer, R. Weiss, V. Dubourg, J. Vanderplas, A. Passos, D. Cournapeau, M. Brucher, M. Perrot, and E. Duchesnay. Scikit-learn: Machine learning in Python. *Journal of Machine Learning Research*, 12:2825–2830, 2011.

[36] Lior Rokach and Oded Maimon. *Data Mining With Decision Trees: Theory and Applications*. World Scientific Publishing Co., Inc., River Edge, NJ, USA, 2nd edition, 2014.

[37] N. V. Chawla, K. W. Bowyer, L. O. Hall, and W. P. Kegelmeyer. Smote: Synthetic minority over-sampling technique.

[38] Tomas Mikolov, Kai Chen, Greg Corrado, and Jeffrey Dean. Efficient estimation of word representations in vector space.

[39] Xin Rong. word2vec parameter learning explained.

[40] Nitish Srivastava, Geoffrey Hinton, Alex Krizhevsky, Ilya Sutskever, and Ruslan Salakhutdinov. Dropout: a simple way to prevent neural networks from overfitting. *The journal of machine learning research*, 15(1):1929–1958, 2014.

[41] Sergey Ioffe and Christian Szegedy. Batch normalization: Accelerating deep network training by reducing internal covariate shift.

Appendix A

A.1 TF-IDF max features row results on liar-liar corpus

A.1.1 Weighted Average Metrics

	model	max_features	recall	precision	f1
0	LinearSVC	10	0.558411	0.590855	0.490108
1	MultinomialNB	10	0.559969	0.596418	0.489844
2	DecisionTreeClassifier	10	0.545171	0.554105	0.492387
3	RidgeClassifier	10	0.558411	0.590855	0.490108
4	LinearSVC	100	0.580997	0.586645	0.562808
5	MultinomialNB	100	0.579439	0.592686	0.549533
6	DecisionTreeClassifier	100	0.539720	0.537812	0.536041
7	RidgeClassifier	100	0.580997	0.586645	0.562808
8	LinearSVC	250	0.603583	0.605262	0.596719
9	MultinomialNB	250	0.596573	0.608079	0.575478
10	DecisionTreeClassifier	250	0.565421	0.565966	0.565582
11	RidgeClassifier	250	0.603583	0.605262	0.596719
12	LinearSVC	1000	0.600467	0.600183	0.597274
13	MultinomialNB	1000	0.594237	0.598593	0.581692
14	DecisionTreeClassifier	1000	0.550623	0.550539	0.550578
15	RidgeClassifier	1000	0.599688	0.599731	0.595559
16	LinearSVC	2500	0.575545	0.574544	0.573173
17	MultinomialNB	2500	0.592679	0.598169	0.578113
18	DecisionTreeClassifier	2500	0.568536	0.568482	0.568508
19	RidgeClassifier	2500	0.584891	0.584169	0.581693
20	LinearSVC	5000	0.571651	0.570574	0.568765
21	MultinomialNB	5000	0.592679	0.600539	0.574854
22	DecisionTreeClassifier	5000	0.552960	0.551400	0.549321
23	RidgeClassifier	5000	0.572430	0.571418	0.568760
24	LinearSVC	10000	0.566978	0.565837	0.564715
25	MultinomialNB	10000	0.598131	0.614854	0.572109
26	DecisionTreeClassifier	10000	0.560748	0.559664	0.559212
27	RidgeClassifier	10000	0.580218	0.579376	0.576984
28	LinearSVC	11222	0.566978	0.565860	0.564915
29	MultinomialNB	11222	0.597352	0.615925	0.569280

30	DecisionTreeClassifier	11222	0.556854	0.555689	0.555171
31	RidgeClassifier	11222	0.582555	0.581782	0.579279

A.1.2 Per Class Metrics

	model	max_features	type	recall	precision	f1
0	LinearSVC	10	fake	0.181818	0.640000	0.283186
1	LinearSVC	10	reliable	0.905689	0.545537	0.680923
2	MultinomialNB	10	fake	0.178571	0.650888	0.280255
3	MultinomialNB	10	reliable	0.911677	0.546188	0.683118
4	DecisionTreeClassifier	10	fake	0.212662	0.569565	0.309693
5	DecisionTreeClassifier	10	reliable	0.851796	0.539848	0.660859
6	RidgeClassifier	10	fake	0.181818	0.640000	0.283186
7	RidgeClassifier	10	reliable	0.905689	0.545537	0.680923
8	LinearSVC	100	fake	0.371753	0.602632	0.459839
9	LinearSVC	100	reliable	0.773952	0.571903	0.657761
10	MultinomialNB	100	fake	0.314935	0.621795	0.418103
11	MultinomialNB	100	reliable	0.823353	0.565844	0.670732
12	DecisionTreeClassifier	100	fake	0.446429	0.523810	0.482033
13	DecisionTreeClassifier	100	reliable	0.625749	0.550725	0.585844
14	RidgeClassifier	100	fake	0.371753	0.602632	0.459839
15	RidgeClassifier	100	reliable	0.773952	0.571903	0.657761
16	LinearSVC	250	fake	0.470779	0.613108	0.532599
17	LinearSVC	250	reliable	0.726048	0.598027	0.655849
18	MultinomialNB	250	fake	0.368506	0.637640	0.467078
19	MultinomialNB	250	reliable	0.806886	0.580819	0.675439
20	DecisionTreeClassifier	250	fake	0.561688	0.545741	0.553600
21	DecisionTreeClassifier	250	reliable	0.568862	0.584615	0.576631
22	RidgeClassifier	250	fake	0.470779	0.613108	0.532599
23	RidgeClassifier	250	reliable	0.726048	0.598027	0.655849
24	LinearSVC	1000	fake	0.509740	0.598095	0.550394
25	LinearSVC	1000	reliable	0.684132	0.602108	0.640505
26	MultinomialNB	1000	fake	0.417208	0.613365	0.496618
27	MultinomialNB	1000	reliable	0.757485	0.584971	0.660144
28	DecisionTreeClassifier	1000	fake	0.529221	0.531811	0.530513
29	DecisionTreeClassifier	1000	reliable	0.570359	0.567809	0.569081
30	RidgeClassifier	1000	fake	0.496753	0.600000	0.543517
31	RidgeClassifier	1000	reliable	0.694611	0.599483	0.643551
32	LinearSVC	2500	fake	0.498377	0.565378	0.529767
33	LinearSVC	2500	reliable	0.646707	0.582996	0.613201
34	MultinomialNB	2500	fake	0.402597	0.615385	0.486752
35	MultinomialNB	2500	reliable	0.767964	0.582293	0.662363
36	DecisionTreeClassifier	2500	fake	0.548701	0.550489	0.549593
37	DecisionTreeClassifier	2500	reliable	0.586826	0.585075	0.585949
38	RidgeClassifier	2500	fake	0.495130	0.578748	0.533683
39	RidgeClassifier	2500	reliable	0.667665	0.589168	0.625965
40	LinearSVC	5000	fake	0.487013	0.561798	0.521739
41	LinearSVC	5000	reliable	0.649701	0.578667	0.612130
42	MultinomialNB	5000	fake	0.383117	0.622691	0.474372
43	MultinomialNB	5000	reliable	0.785928	0.580110	0.667514
44	DecisionTreeClassifier	5000	fake	0.459416	0.540076	0.496491

45	DecisionTreeClassifier	5000	reliable	0.639222	0.561842	0.598039
46	RidgeClassifier	5000	fake	0.477273	0.564299	0.517150
47	RidgeClassifier	5000	reliable	0.660180	0.577982	0.616352
48	LinearSVC	10000	fake	0.491883	0.554945	0.521515
49	LinearSVC	10000	reliable	0.636228	0.575881	0.604552
50	MultinomialNB	10000	fake	0.345779	0.653374	0.452229
51	MultinomialNB	10000	reliable	0.830838	0.579332	0.682657
52	DecisionTreeClassifier	10000	fake	0.498377	0.546263	0.521222
53	DecisionTreeClassifier	10000	reliable	0.618263	0.572022	0.594245
54	RidgeClassifier	10000	fake	0.490260	0.573055	0.528434
55	RidgeClassifier	10000	reliable	0.663174	0.585205	0.621754
56	LinearSVC	11222	fake	0.495130	0.554545	0.523156
57	LinearSVC	11222	reliable	0.633234	0.576294	0.603424
58	MultinomialNB	11222	fake	0.336039	0.657143	0.444683
59	MultinomialNB	11222	reliable	0.838323	0.577915	0.684178
60	DecisionTreeClassifier	11222	fake	0.491883	0.542039	0.515745
61	DecisionTreeClassifier	11222	reliable	0.616766	0.568276	0.591529
62	RidgeClassifier	11222	fake	0.491883	0.576046	0.530648
63	RidgeClassifier	11222	reliable	0.666168	0.587071	0.624123

A.2 TF-IDF max features row results for fake news corpus without SMOTE

	model	recall	precision	max_features	f1
0	LinearSVC	0.744278	0.741135	10000	0.742519
1	MultinomialNB	0.638690	0.666131	10000	0.647986
2	DecisionTreeClassifier	0.649834	0.664873	10000	0.655847
3	RidgeClassifier	0.730762	0.739152	10000	0.734125
4	LinearSVC	0.757846	0.758336	50000	0.758086
5	MultinomialNB	0.658424	0.680365	50000	0.666197
6	DecisionTreeClassifier	0.668170	0.685830	50000	0.674751
7	RidgeClassifier	0.742044	0.756513	50000	0.746957
8	LinearSVC	0.757589	0.757112	100000	0.757346
9	MultinomialNB	0.657282	0.681421	100000	0.665562
10	DecisionTreeClassifier	0.669952	0.685943	100000	0.676058
11	RidgeClassifier	0.742437	0.754869	100000	0.746859
12	LinearSVC	0.757935	0.756228	250000	0.757021
13	MultinomialNB	0.660640	0.683517	250000	0.668590
14	DecisionTreeClassifier	0.668623	0.685926	250000	0.675103
15	RidgeClassifier	0.744877	0.754769	250000	0.748600
16	LinearSVC	0.753187	0.750750	500000	0.751847
17	MultinomialNB	0.672800	0.687760	500000	0.678584
18	DecisionTreeClassifier	0.673318	0.690695	500000	0.679764
19	RidgeClassifier	0.742800	0.751770	500000	0.746270
20	LinearSVC	0.753156	0.750642	1000000	0.751769
21	MultinomialNB	0.673599	0.688095	1000000	0.679241
22	DecisionTreeClassifier	0.673084	0.688515	1000000	0.679001
23	RidgeClassifier	0.742725	0.751611	1000000	0.746170

Appendix B

B.1 Training plot for attention mechanism

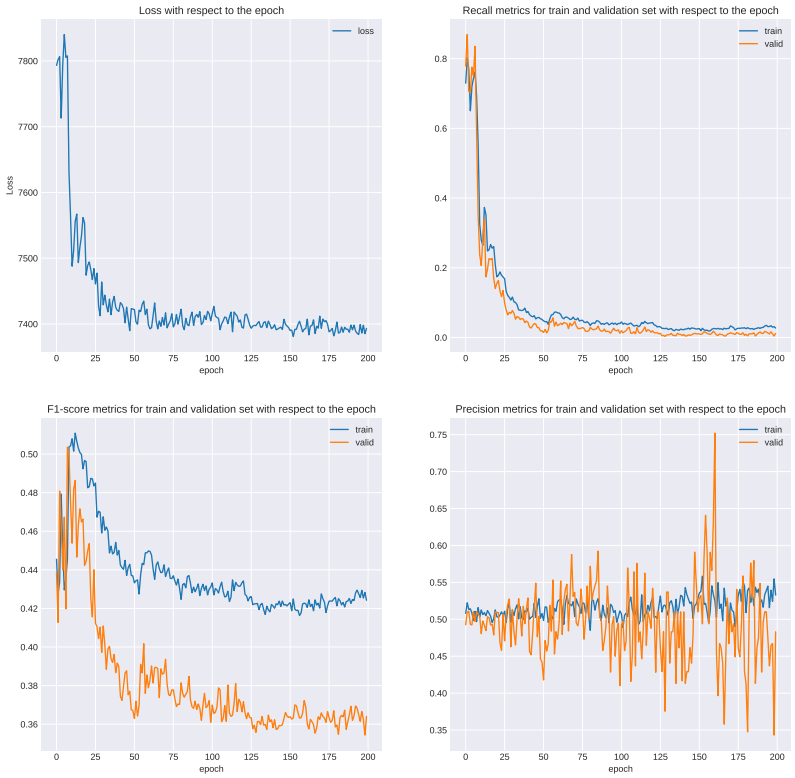


Figure B.1: There is a spike for the precision, but that does not means that the model performs well.

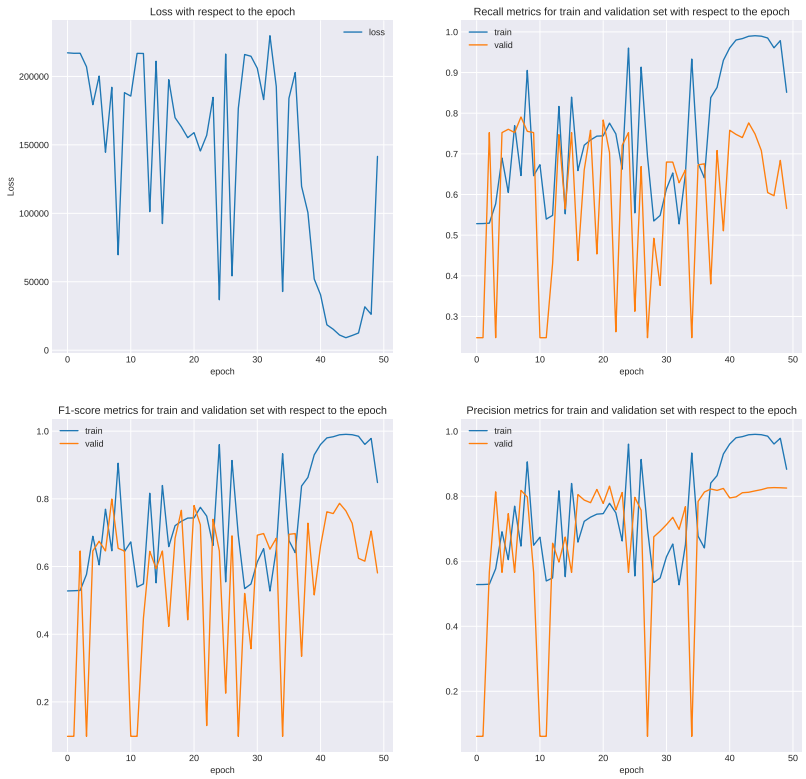


Figure B.2: Oscillation of the Loss