CONTENTS

Chapter No.	Topic Name	Page No.
1	INTRODUCTION	1 - 6
2	RELATED WORK	7 - 9
3	DATA EXPLORATION	10 - 15
4	MACHINE LEARNING TECHNIQUES	16 - 28
5	ATTENTION MECHANISM	28 – 45
6	IMPLEMENTATION AND OUTPUTS	46 - 48
7	CONCLUSION	49

1. INTRODUCTION

1.1 What is 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 people's behaviour. It has been shown that propagation of fake news has had a non-negligible influence of 2016 US presidential elections. A few facts on fake news in the United States:

- 62% of US citizens get their news for social medias
- Fake news had more share on Facebook than mainstream news.
- Fake news has also been used in order to influence the referendum in the United Kingdom for the "Brexit".

In this project we experiment the possibility to detect fake news based only on textual information by applying traditional machine learning techniques as well as bidirectional-LSTM and Attention Mechanism 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.*

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.

1.1.2 Fake News Characterization:

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.

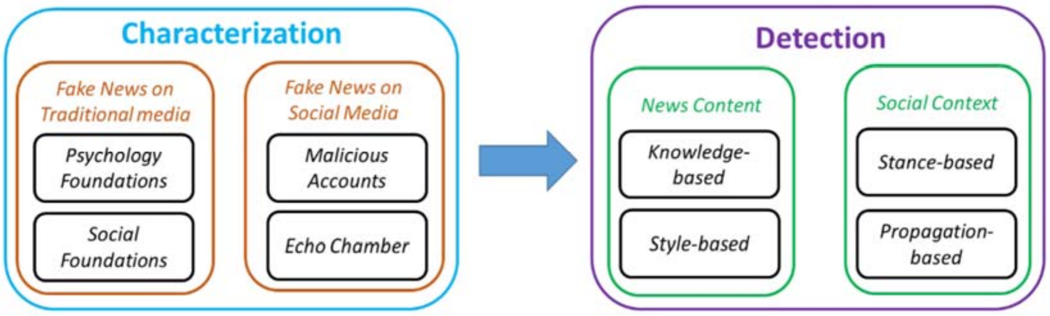


Figure 1.1: Fake news on social media: from characterization to detection

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:** Usually, textual information is agreement 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.

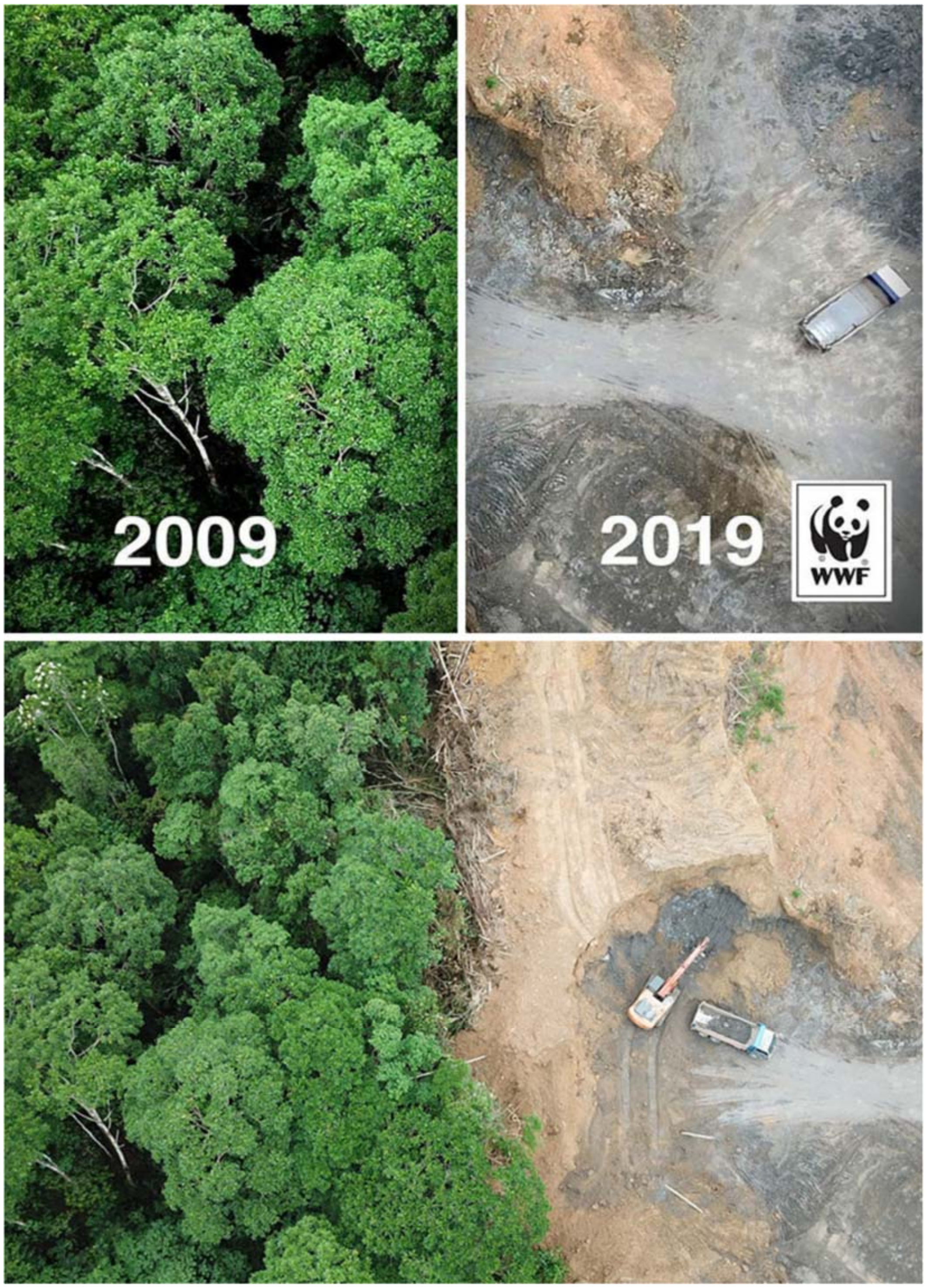


Figure 1.2: The two images provided to show deforestation between two dates are from the same image taken at the same time.

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).

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. This 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 retweets on twitter.

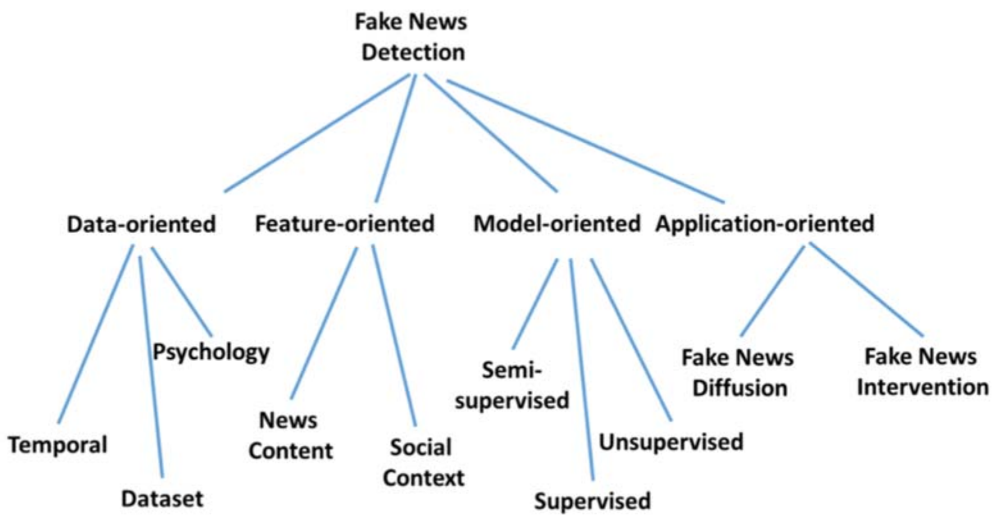


Figure 1.3: Different approaches to fake news detection.

1.5 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 focusses on textual news content features. Indeed, other features related to social media are difficult to acquire. For example, user's information is difficult to obtain on Facebook, as well as post information.

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.

2. RELATED WORK

2.1 Supervised Learning for Fake News Detection

Reis et al. use machine learning techniques on BuzzFeed article related to US election. The evaluated algorithm are k-Nearest Neighbours, Naive-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 different features,
- Lexical Features: number of unique words and their frequencies, pronouns, etc,
- Psychological Features: 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 were also used, based on the source and social metadata.

Their results are shown at Figure 2.1.

They also show that XGBoost is good for selecting texts that need to be hand-verified, this means that the texts classified as reliable are indeed reliable, and thus reducing the number of texts the be checked manually. This model is limited by the fact they do use metadata that is not always available.

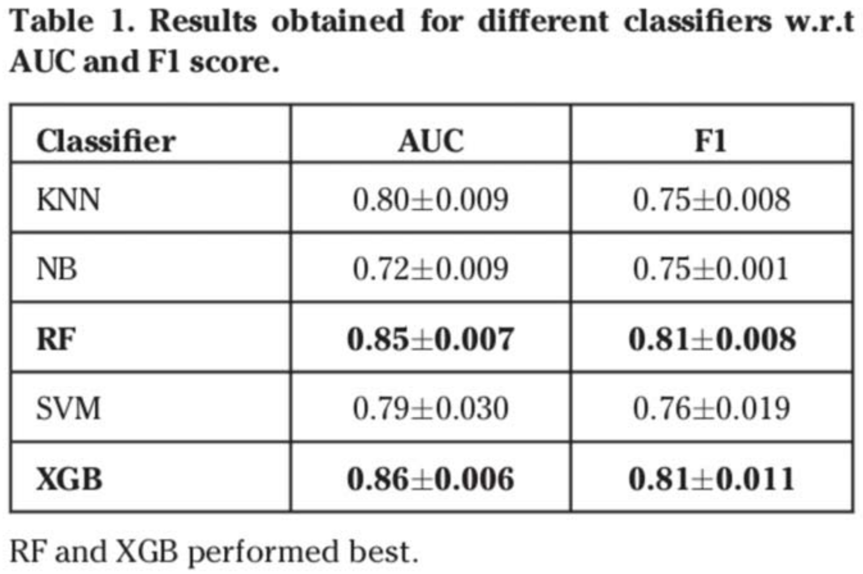


Table 1. Results obtained for different classifiers w.r.t AUC and F1 score.

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.

2.2 Some Like it Hoax: Automated Fake News Detection in Social Networks

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 main categories: scientific news and conspiracy news. They used logistic regression and harmonic algorithm to classify news in categories hoax and non-hoax. Harmonic Algorithm is a method that allows transferring 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 leads to good results, harmonic algorithm outperforming logistic regression. Results are shown at Figures 2.2 and 2.3.

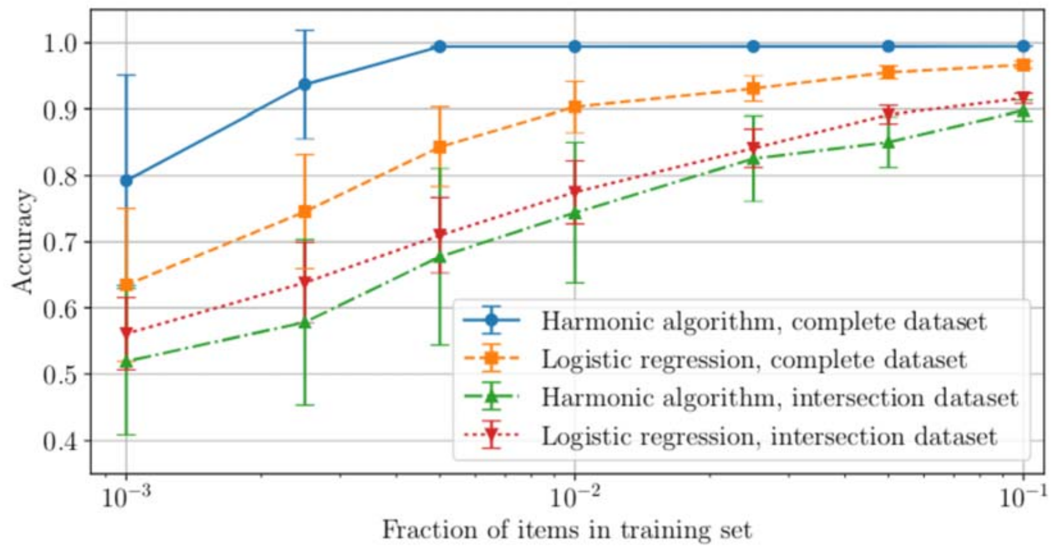
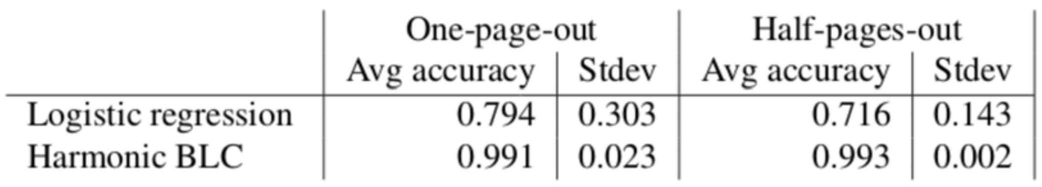


Figure 2.2: Results by Tacchnini et al.



	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 2.3: Results by tacchnini et al.

2.3 Convolutional Neural Networks for Fake News Detection

Yang et al. used a CNN with images contained in article in order to make the classification. They used Kaggle fake news dataset, in addition they scrapped real news from trusted source such as New York Times and Washington Post.

Their network is made of two branches: one text branch and one image branch (Figure 2.4). The textual branch is then divided of two subbranch: textual explicit: derived information from text such as length of the news and the text latent subbranch, which is the embedding of the text, limited to 1000 words.

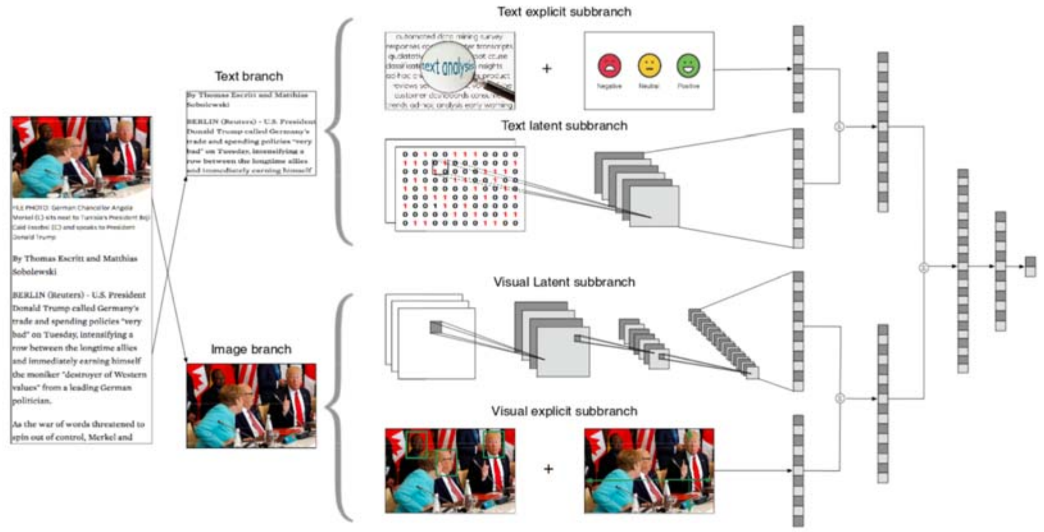


Figure 2.4: TI-CNN

2.4 Conclusion

We have seen in the previous sections that most of the related works focus on improving the prediction quality by adding additional features. The fact is that these features are not always available, for instance some article may not contain images. There is also the fact that using social media information is problematic because it is easy to create a new account on these media and fool the detection system. That's why I chose to focus on the article body only and see if it is possible to accurately detect fake news.

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. 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 focusses 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, 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 formatted, 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.

Type	Provided	Computed
Fake News	9,28,083	7,70,287
Satire	1,46,080	85,523
Extreme Bias	13,00,444	7,71,407
Conspiracy Theory	9,05,981	4,95,402
Junk Science	1,44,939	79,342
Hate News	1,17,374	65,264
Clickbait	2,92,201	1,76,403
Proceed with Caution	3,19,830	1,04,657
Political	24,35,471	9,72,283
Credible	19,20,139	18,11,644

Table 3.1: Number of texts per categories

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

- Unknown: 2,31,301
- Rumour: 3,76,815

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 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.

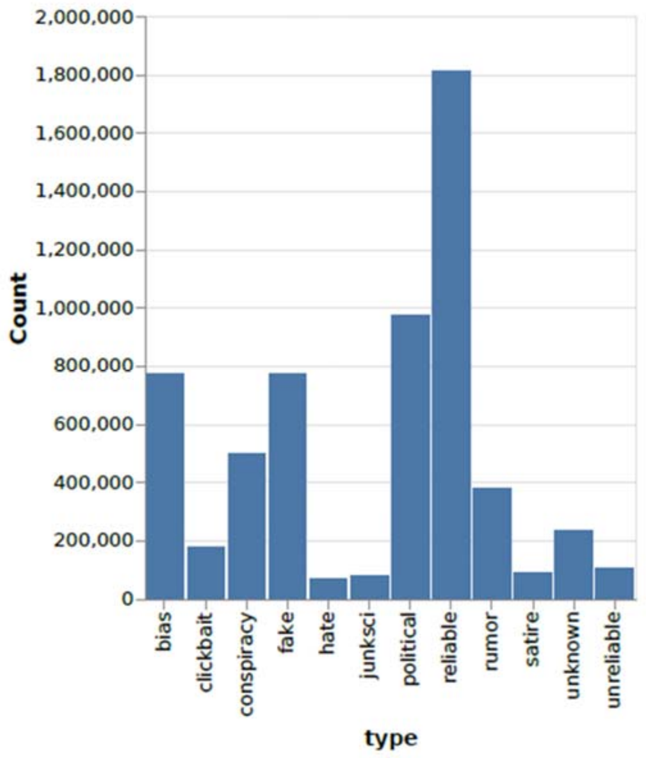


Figure 3.1: Histogram of text distribution along their categories on the computed numbers.

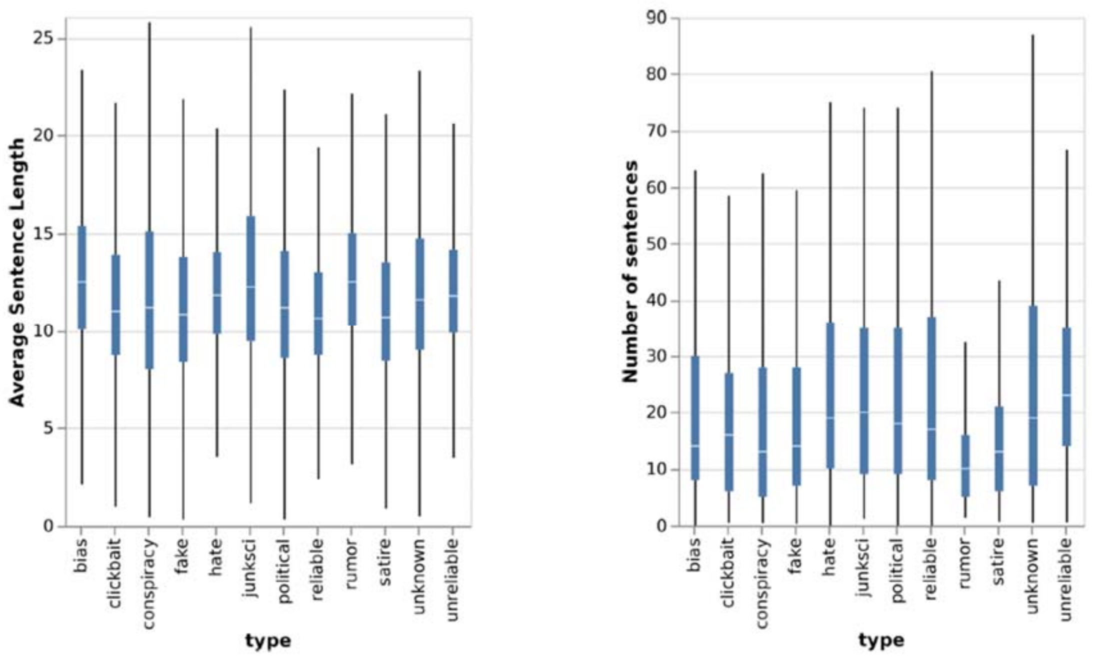
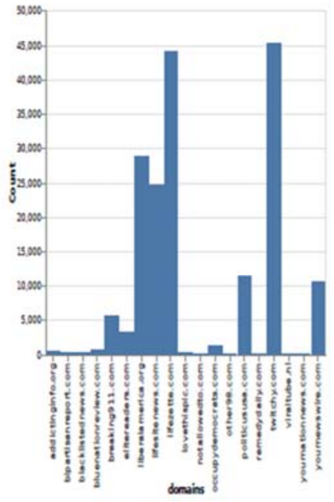
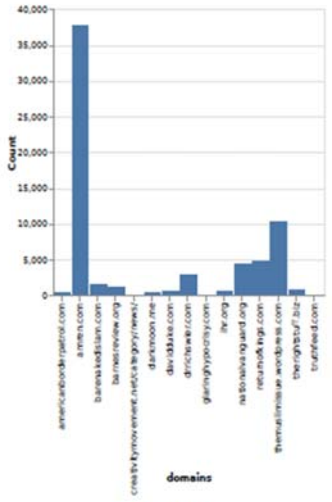


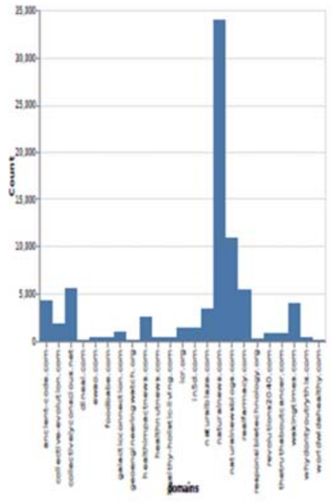
Figure 3.2: Summary statistics



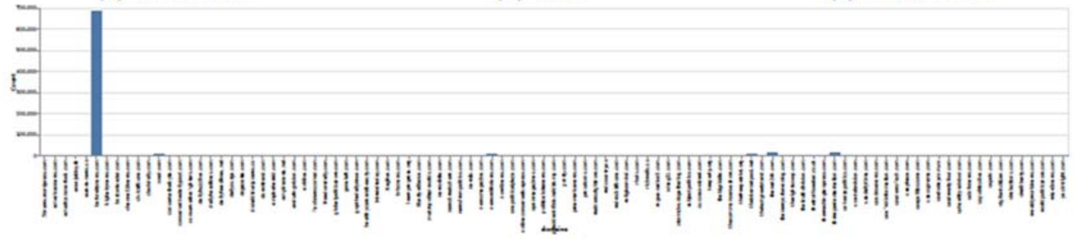
(a) Clickbaits



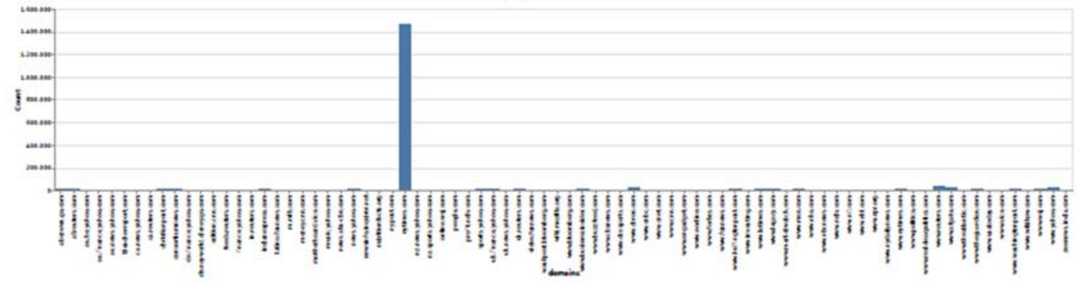
(b) Hate



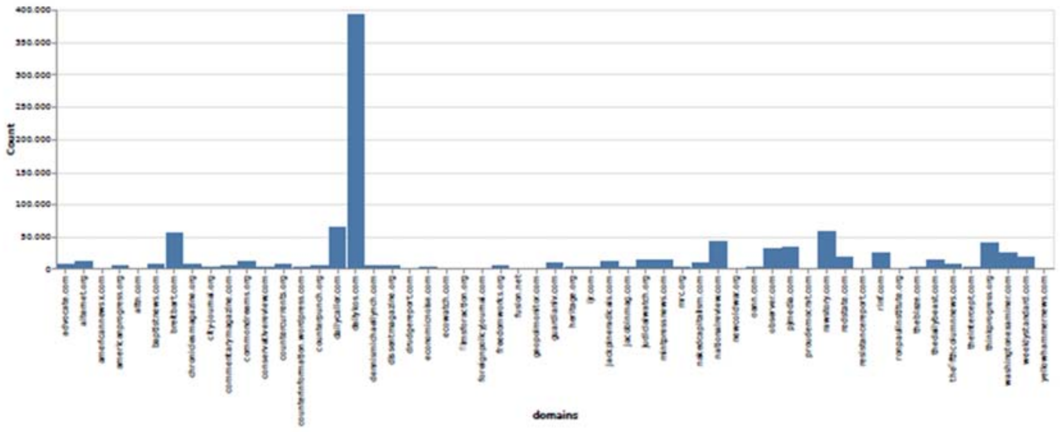
(c) Junk sience



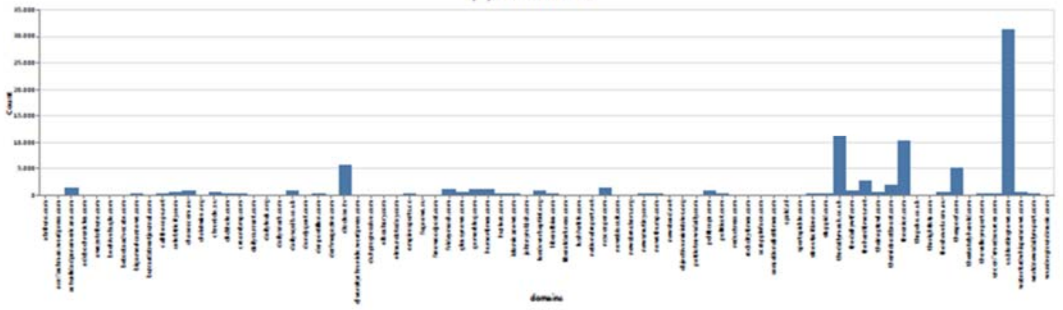
(d) Fake



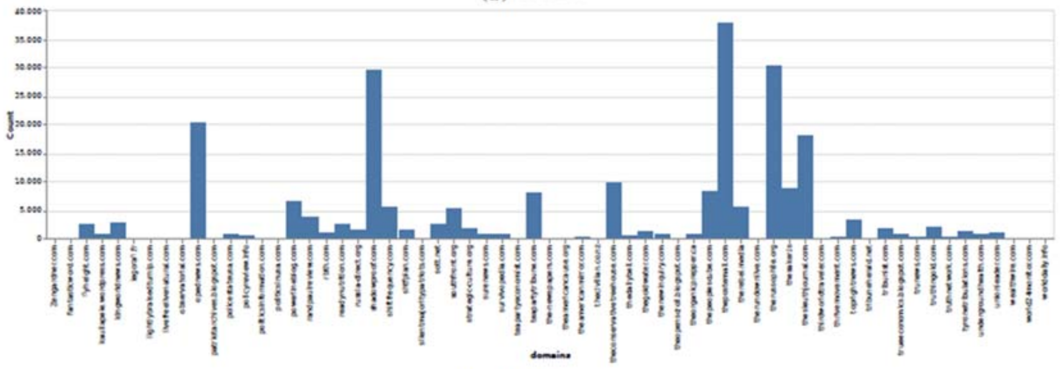
(e) Reliable



(f) Political



(g) Satire



(h) Unknown

Figure 3.3: Histogram of news origin for each category.

3.5 Conclusion

Data exploration has shown that there are 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.

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 Naive-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 × 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

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

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 *ny-times.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

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

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 Naive-Bayes, Linear SVM, Ridge Classifier and Decision Tree. we will start by a very brief recap of each model and how they work.

4.4.1 Naive-Bayes:

The basic idea of Naive-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 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 (x_i; y_i),y_i ∈ {-1, 1}, it solves the following unconstrained optimization problem.

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

Where ε_i is a loss function, in this case L2 loss function has been used, and C > 0 a penalty parameter. Class of new examples are assigned by looking at the value of $w^T w$.

The class 1 is assigned if $w^T w \geq 0$ and the class -1 if $w^T w < 0$.

4.4.3 Decision Tree:

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.

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

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.

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

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 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.

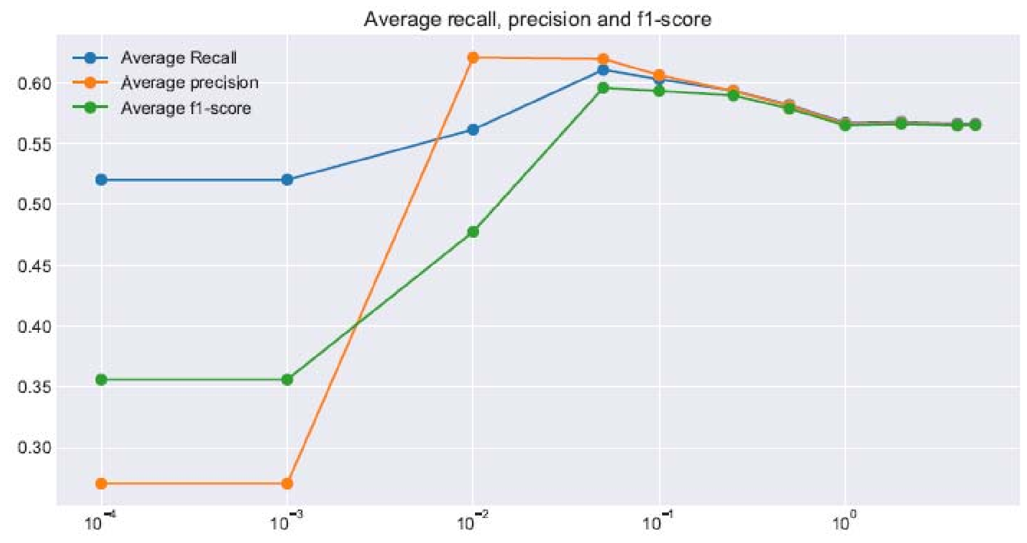


Figure 4.1: Tuning linearSVC parameters

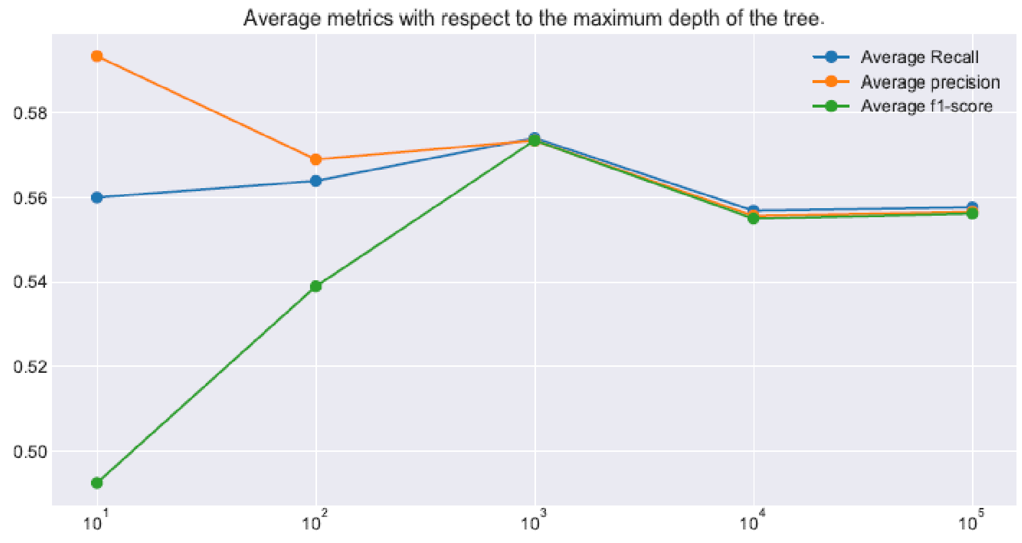


Figure 4.2: Tuning Decision Tree Parameters

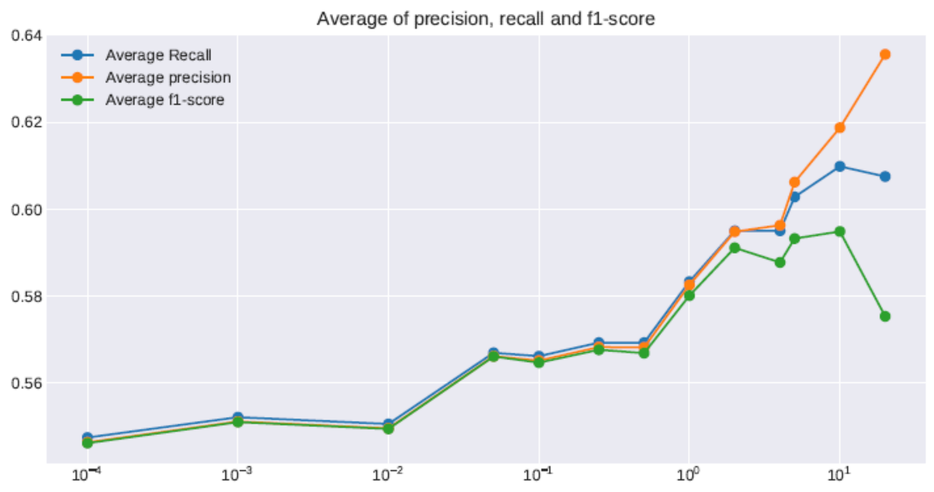


Figure 4.3: Average metrics for ridge classifiers with respect to the penalty parameter.

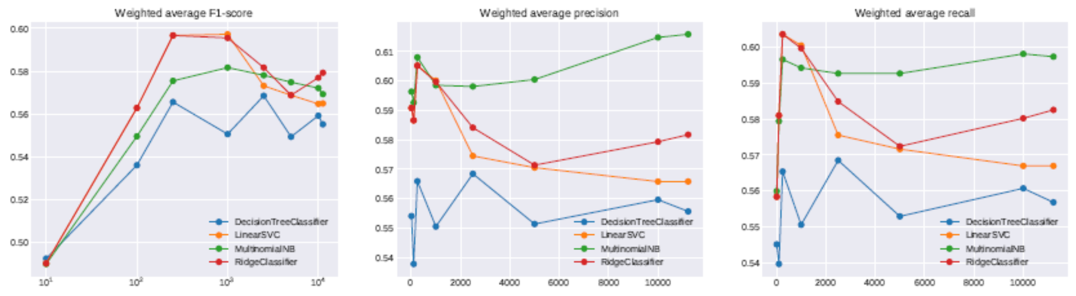


Figure 4.4: Weighted average of f1-score, precision and recall of each class.

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

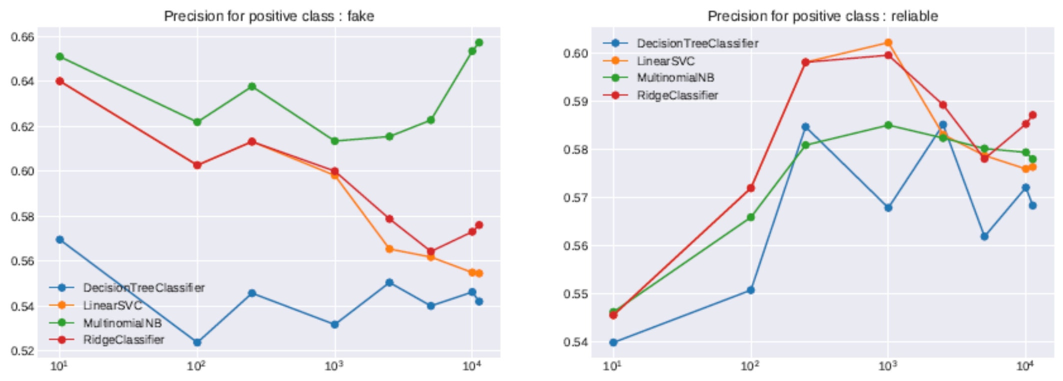


Figure 4.5: Precision of the model for each class, the x axes is log scale of the number of features

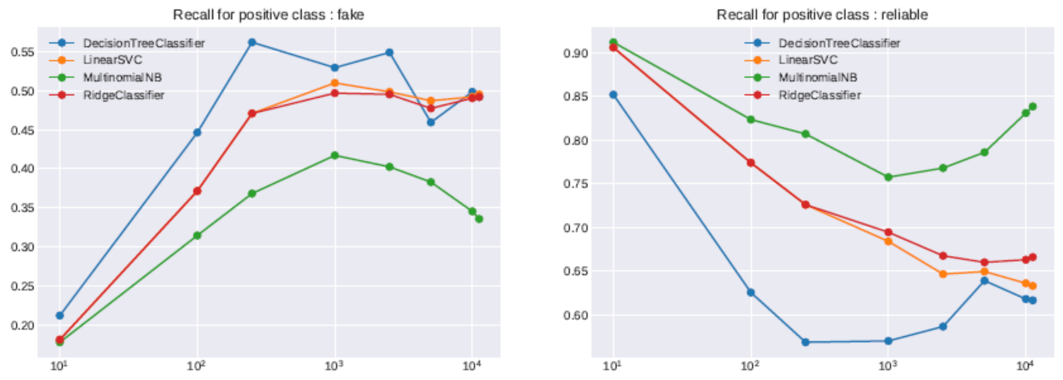


Figure 4.6: Precision of the model for each class, the x axes is log scale of the number of features

4.6 Models on fake corpus dataset

4.6.1 SMOTE: Synthetic Minority Over-sampling Technique:

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 4.4, 4.5 and 4.6 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 slightly 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, Naive-Bayes reaches a pike at 1,00,000 features and greatly decrease afterward.

Figure 4.4 shows why it is important to look at all the metrics, because Naive-Bayes reaches a recall of 1 for the reliable class and close to 0 for the fake class, which means that almost all texts are classified as reliable.

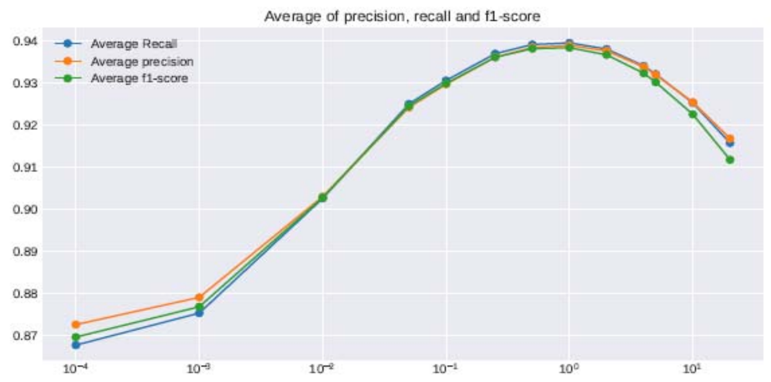


Figure 4.7: Metrics value with respect to the penalty parameter for ridge classifier

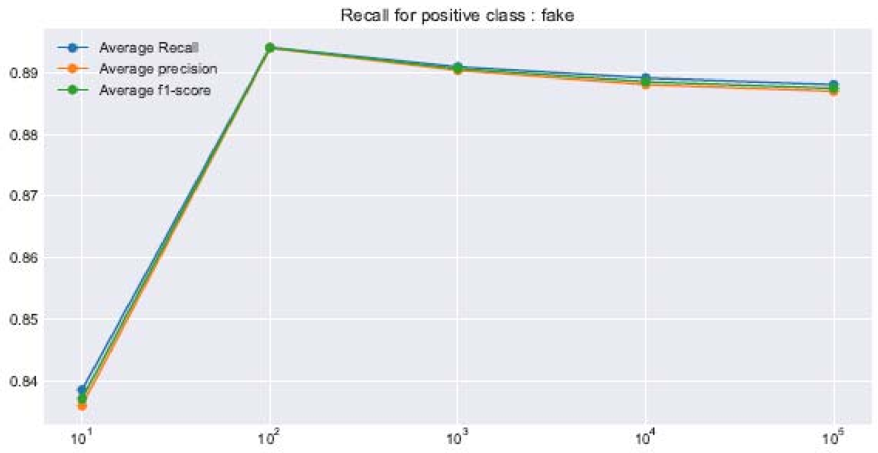


Figure 4.8: Optimal depth of decision tree.

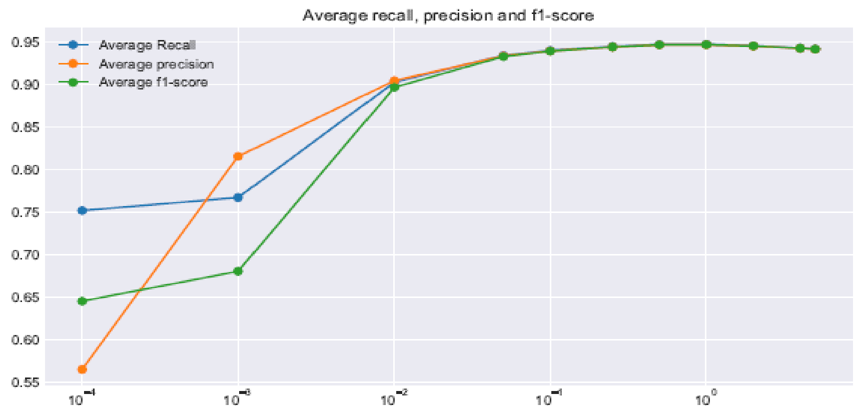


Figure 4.9: Optimal penalty parameters for linear svm

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 Naive-Bayes behaves as it removes overfitting when using a larger number of features in TF-IDF, leading to a few percent of accuracy increase.

Its 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.

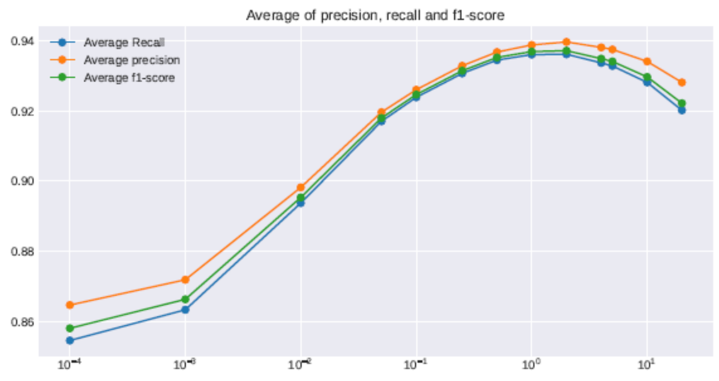


Figure 4.14: Metrics value with respect to the penalty parameter for ridge classifiers when using SMOTE.

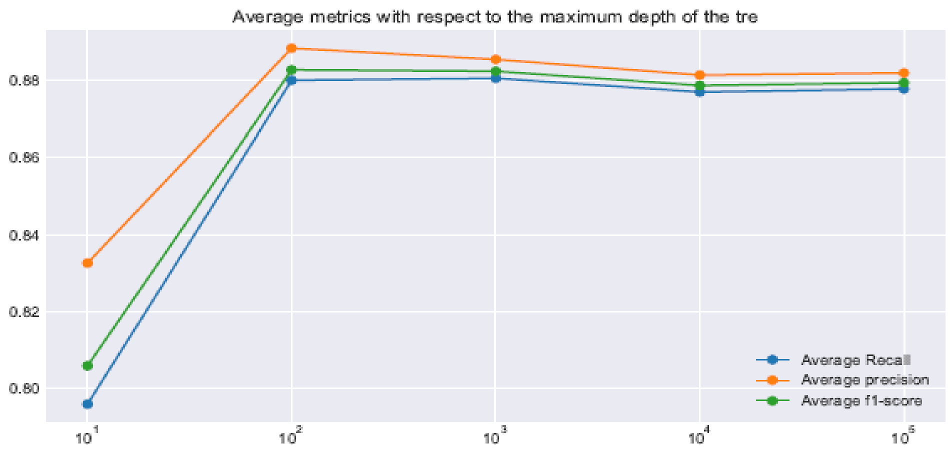


Figure 4.15: Optimal depth of decision tree when using SMOTE.

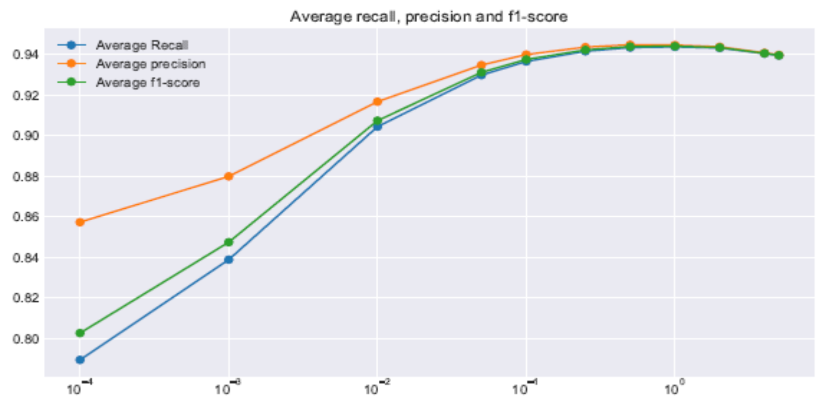


Figure 4.16: Optimal penalty parameters for linear svm when using SMOTE

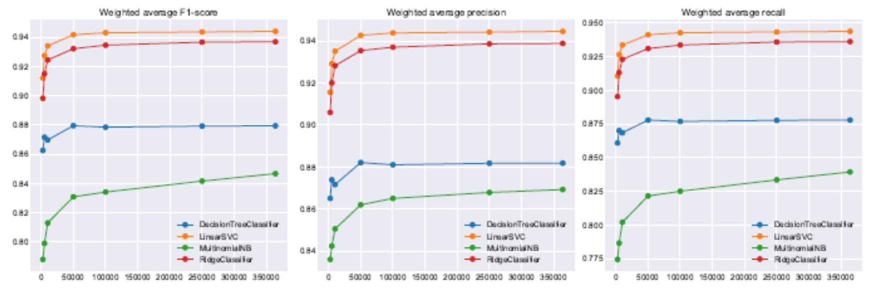


Figure 4.17: Average recall, precision and f1-score wti respect to the maximum number of features.

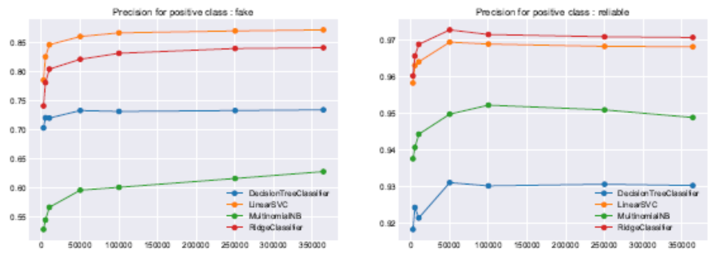


Figure 4.18: Precision for fake and reliable class for each model with respect to the maximum number of features

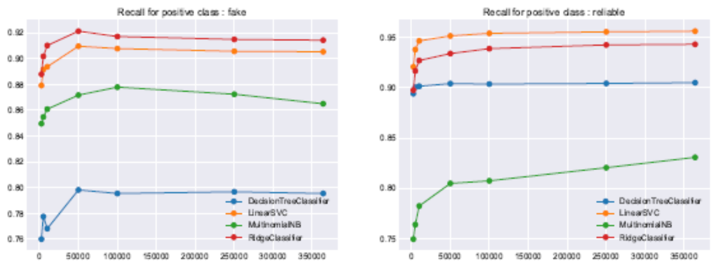


Figure 4.19: Recall for fake and reliable class for each model with respect to the maximum number of features

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.

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 α = 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,
- Naive-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 α = 1,
- Decision Tree with maximum depth of 100.

They will all be trained using 1,00,000 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.

4.8 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.

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.

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 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 ≈ cat - cats but also Paris - France ≈ 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 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 gives 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

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

Where $W^{V\times N}$ 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 $W^{N\times V}$ 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.

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

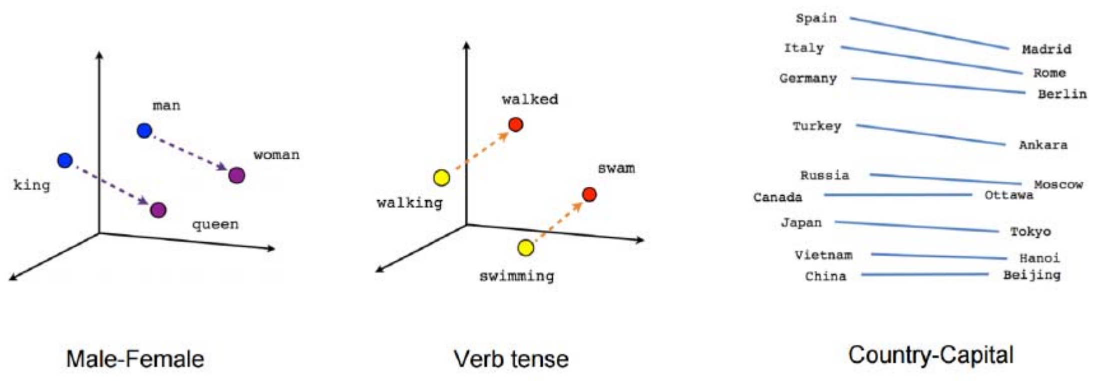


Figure 5.1: Relationships between different words with t-SNE dimensionality reduction.

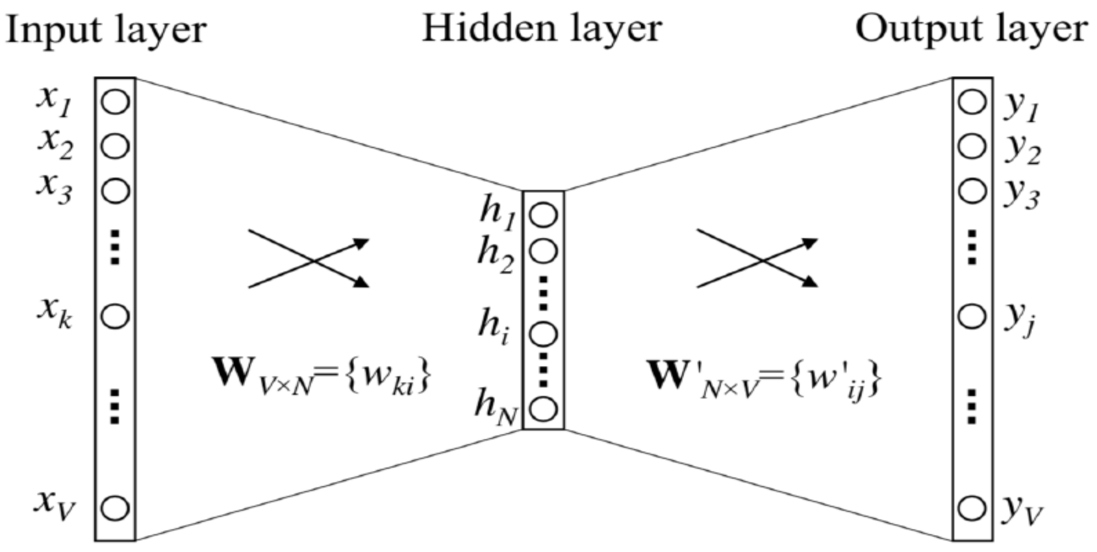


Figure 5.2: A simple CBOW model with only one word in the context

5.3 LSTM

LSTM or Long Short Term Memory 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 work 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 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.

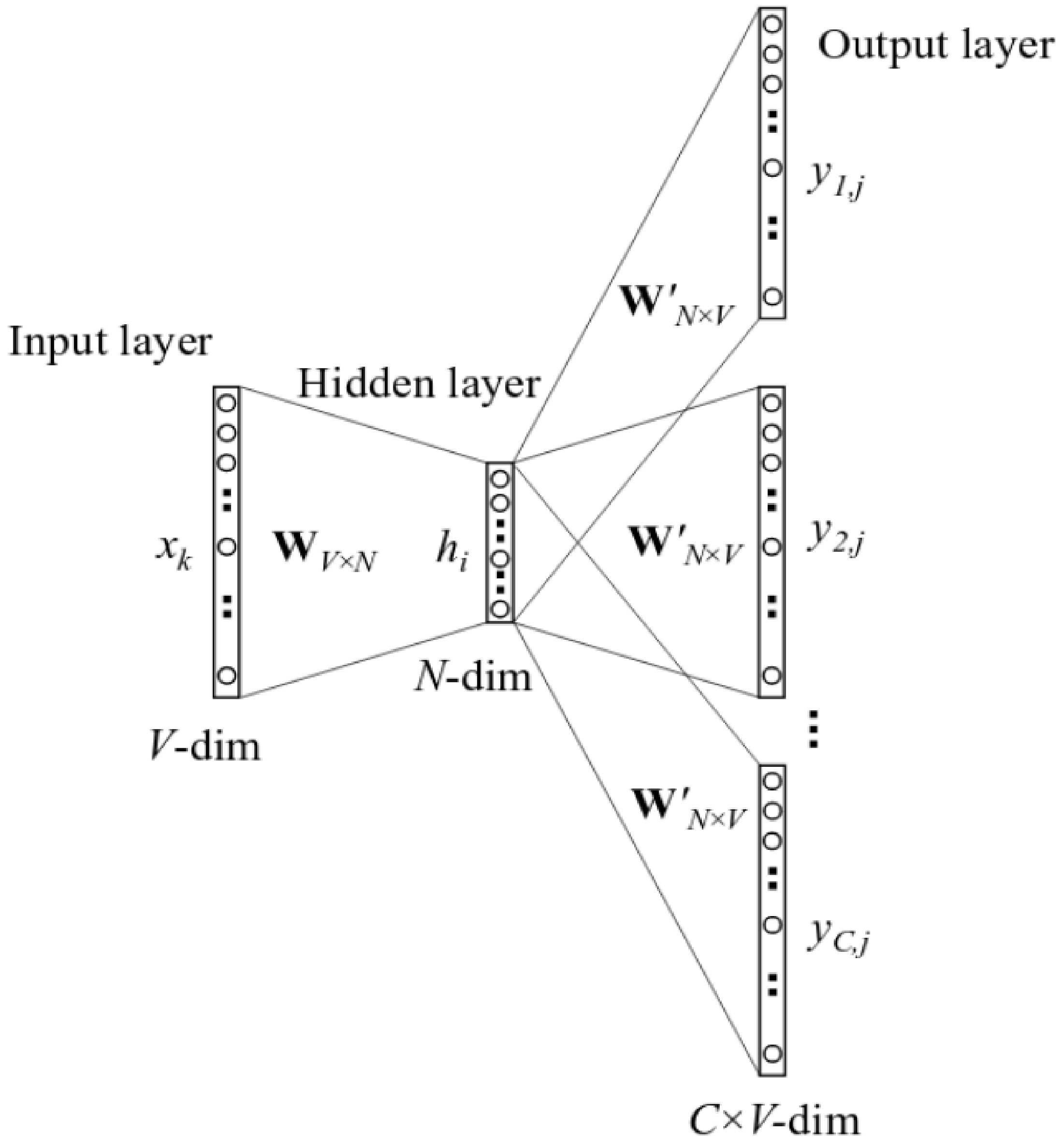


Figure 5.3: Skip-gram model with multiple outputs.

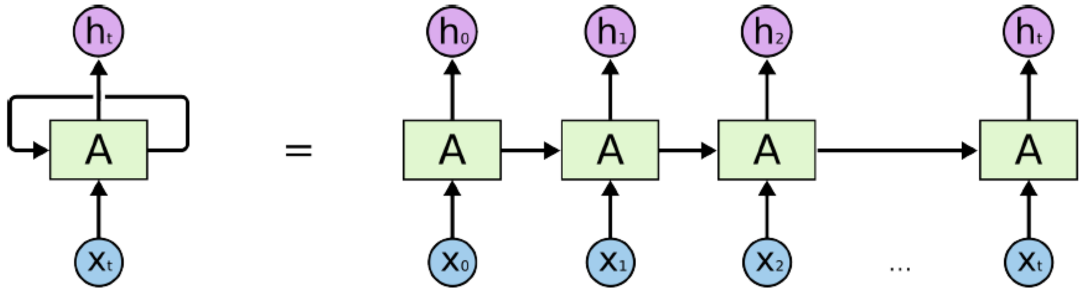


Figure 5.4: Unrolled RNN (Understanding LSTM Networks, `https://colah.github.io/posts/2015-08-Understanding-LSTMs/`)

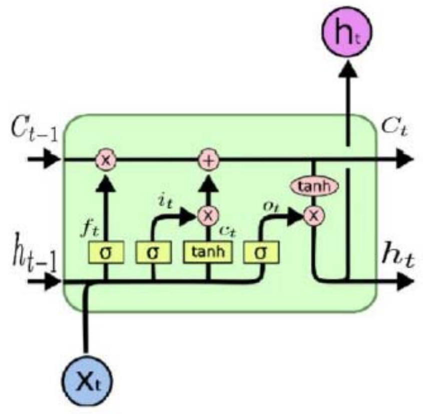


Figure 5.5: LSTM gates,
https://hackernoon.com/understanding-architecture-of-lstm-cell-from-scratch-with-code-8da40f0b71f4)

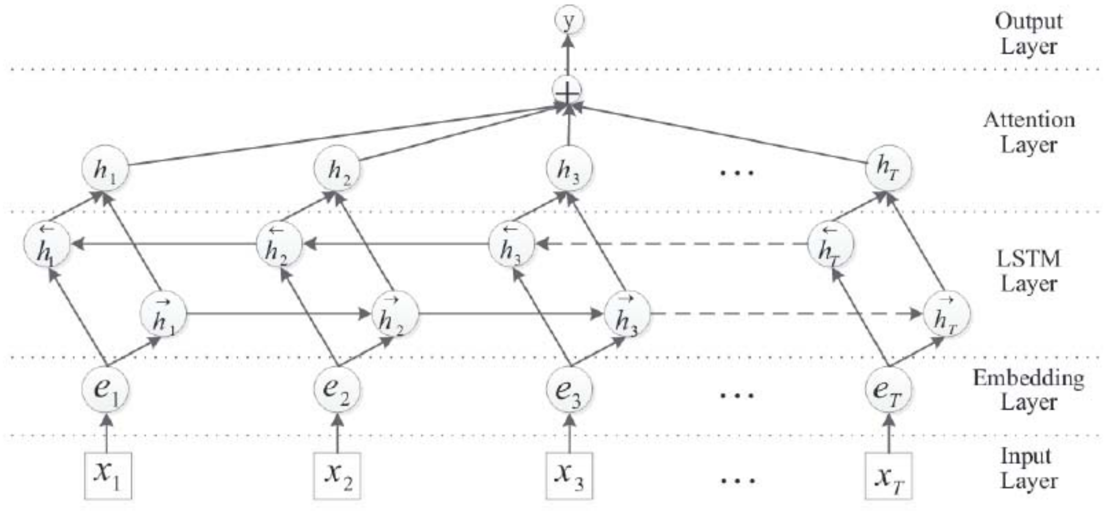


Figure 5.6: Bidirectional LSTM Model With Attention

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 knowledge 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 tuneable parameter with different embedding size.

LSTM

When it comes to LSTM trained on liar-liar dataset, it simply does not work. 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 tuneable 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.

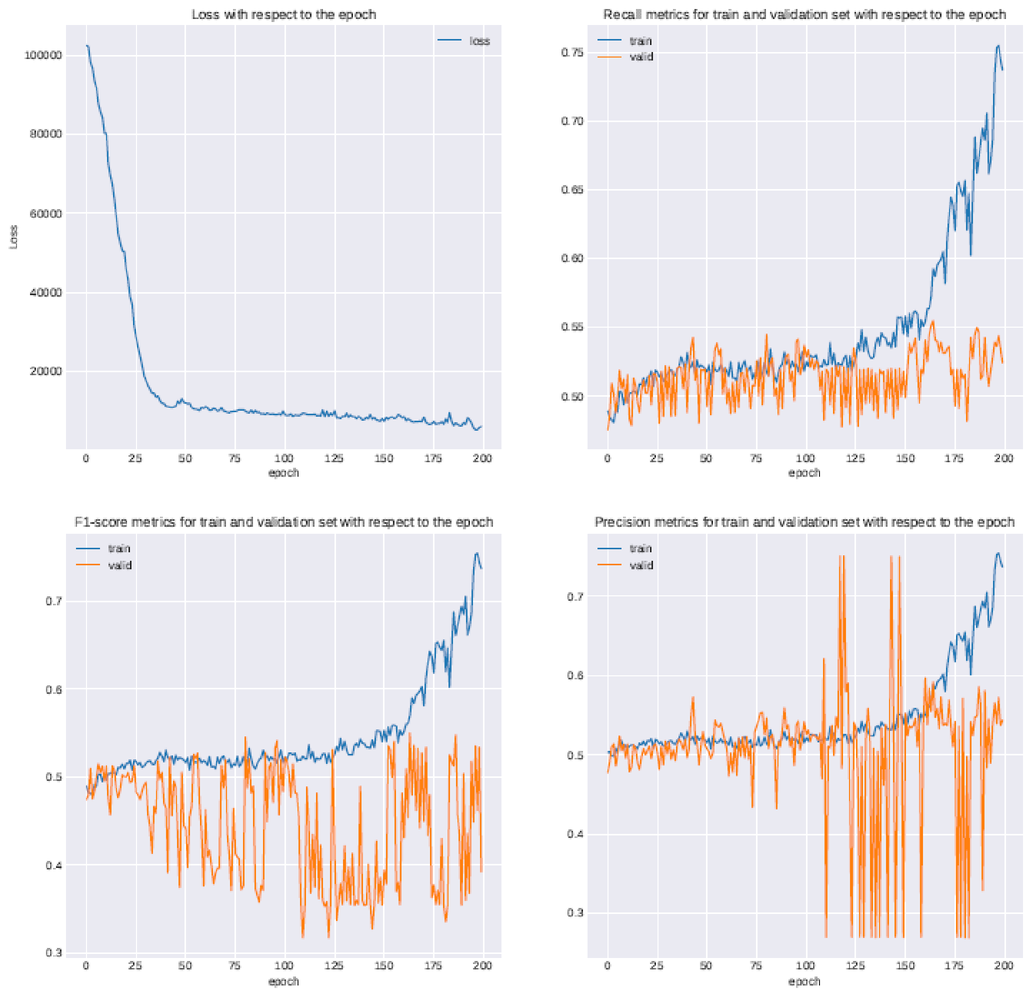


Figure 5.7: Best LSTM With word2vec

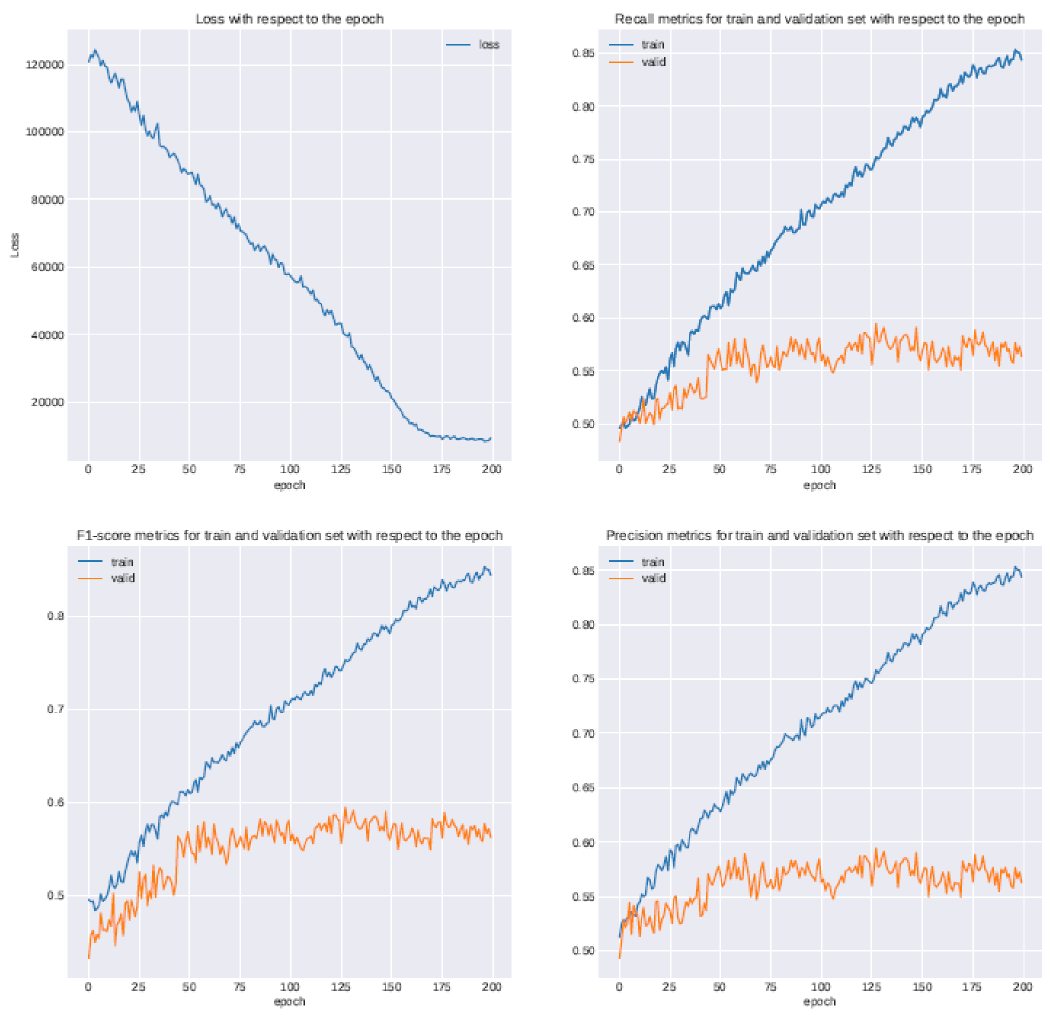


Figure 5.8: Best LSTM with word embedding as tunable parameters.

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

Finally, there is the models where the embedding is a tuneable 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,
- 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,
- batch-normalization
- 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.

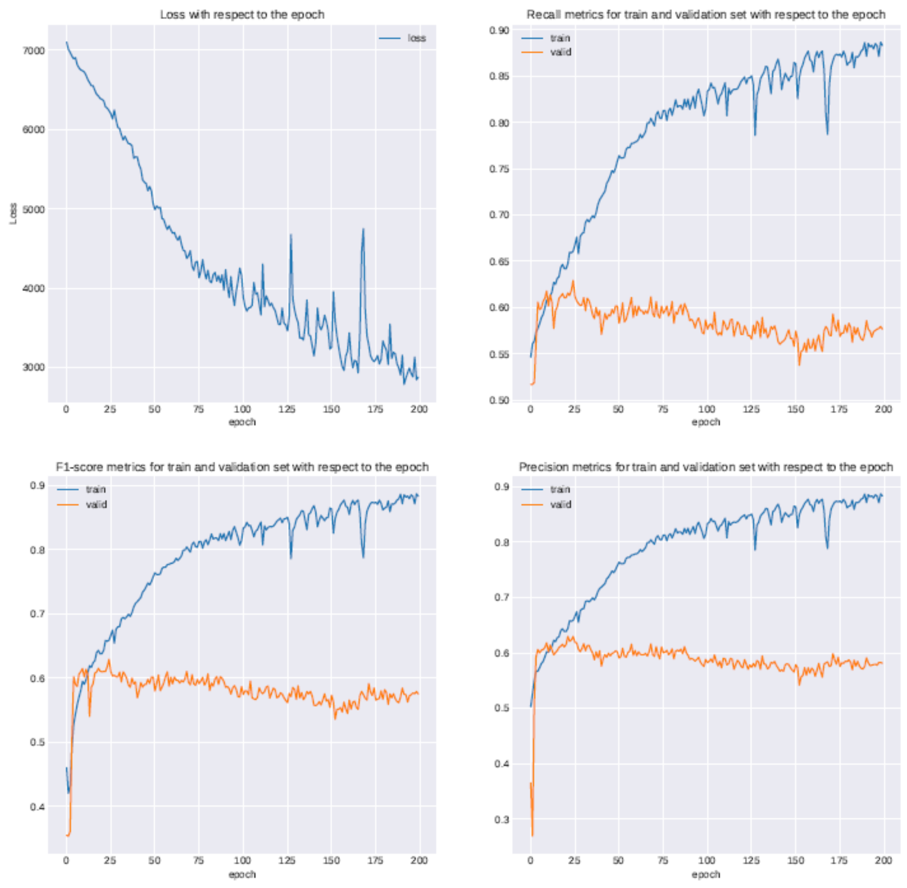


Figure 5.11: Training and validation of the model with top precision trained with word2vec embedding.

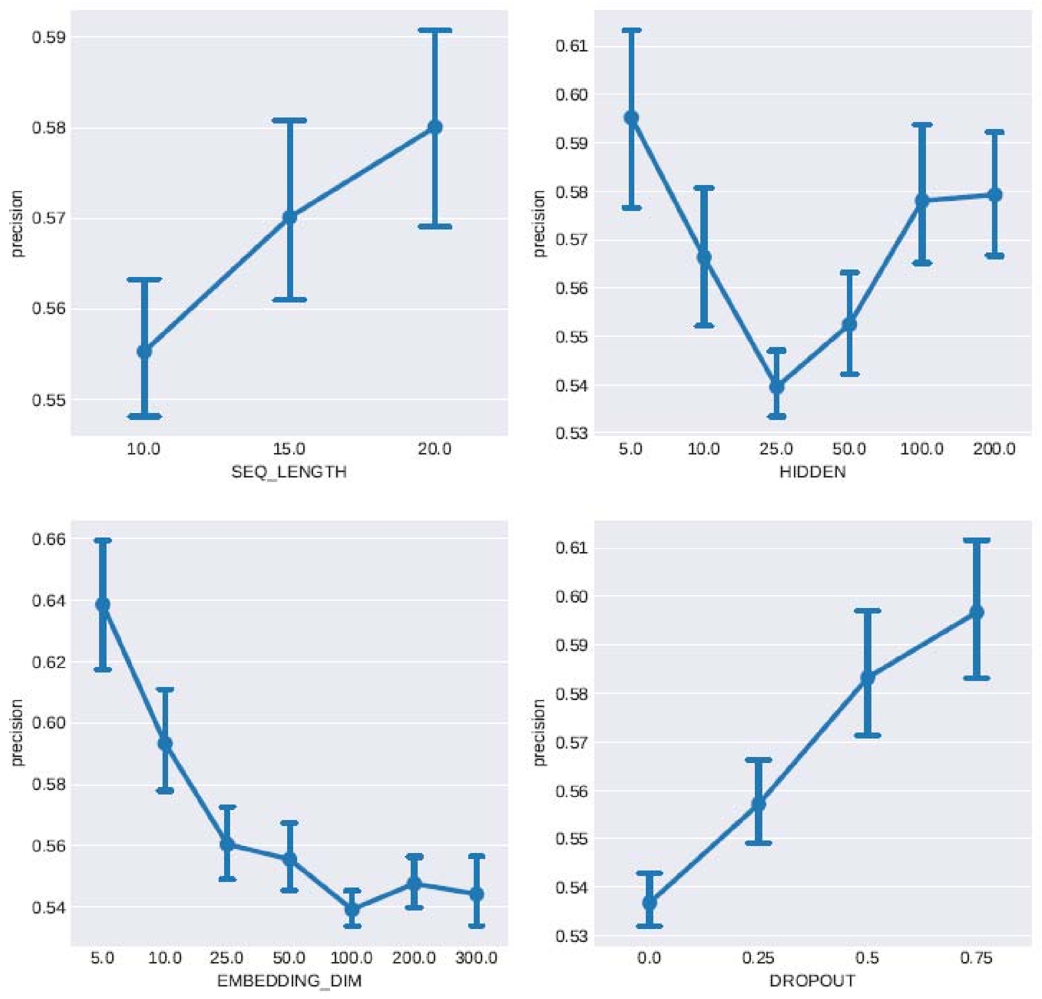


Figure 5.12: Training and Validation of the Model With top Precision

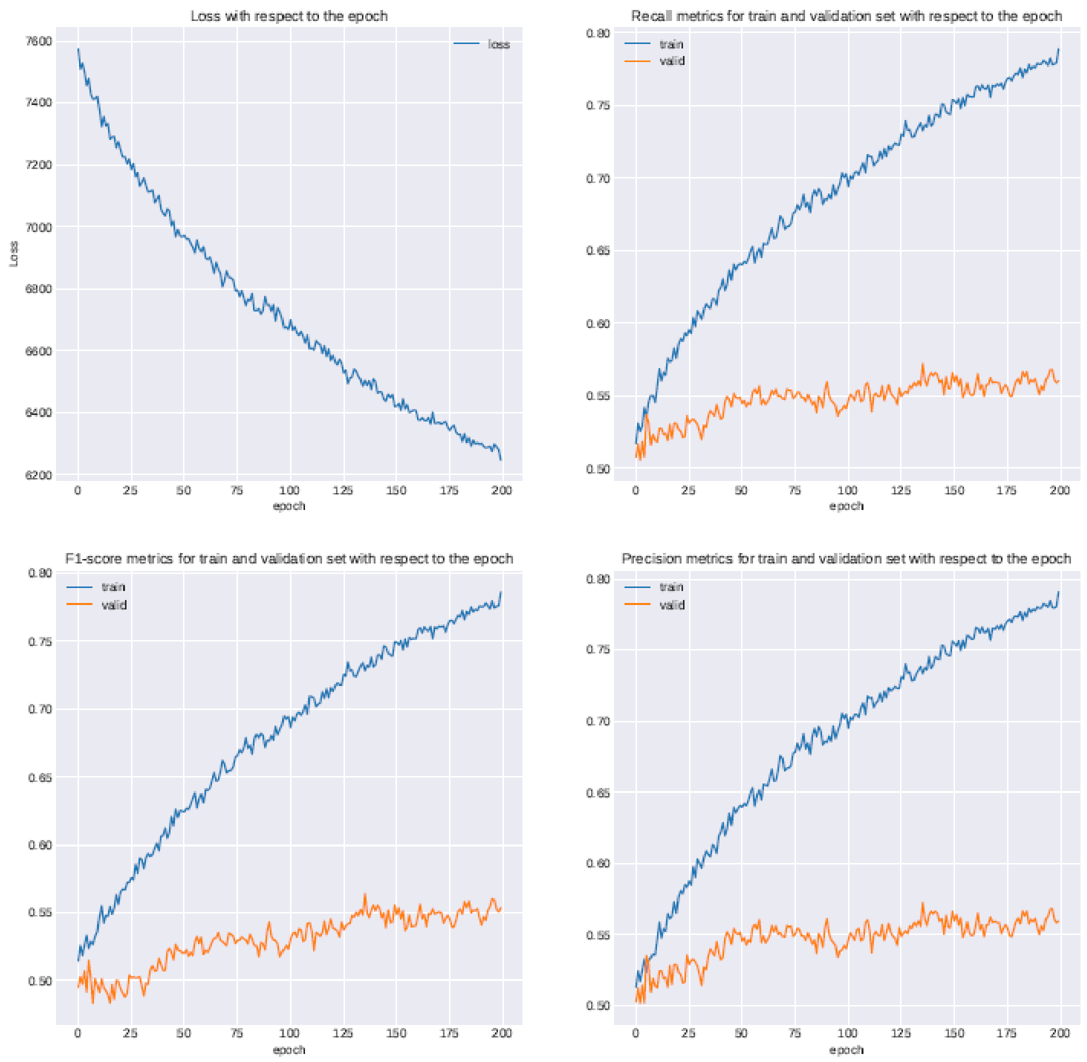


Figure 5.13: Training and validation of the model with top f1-score

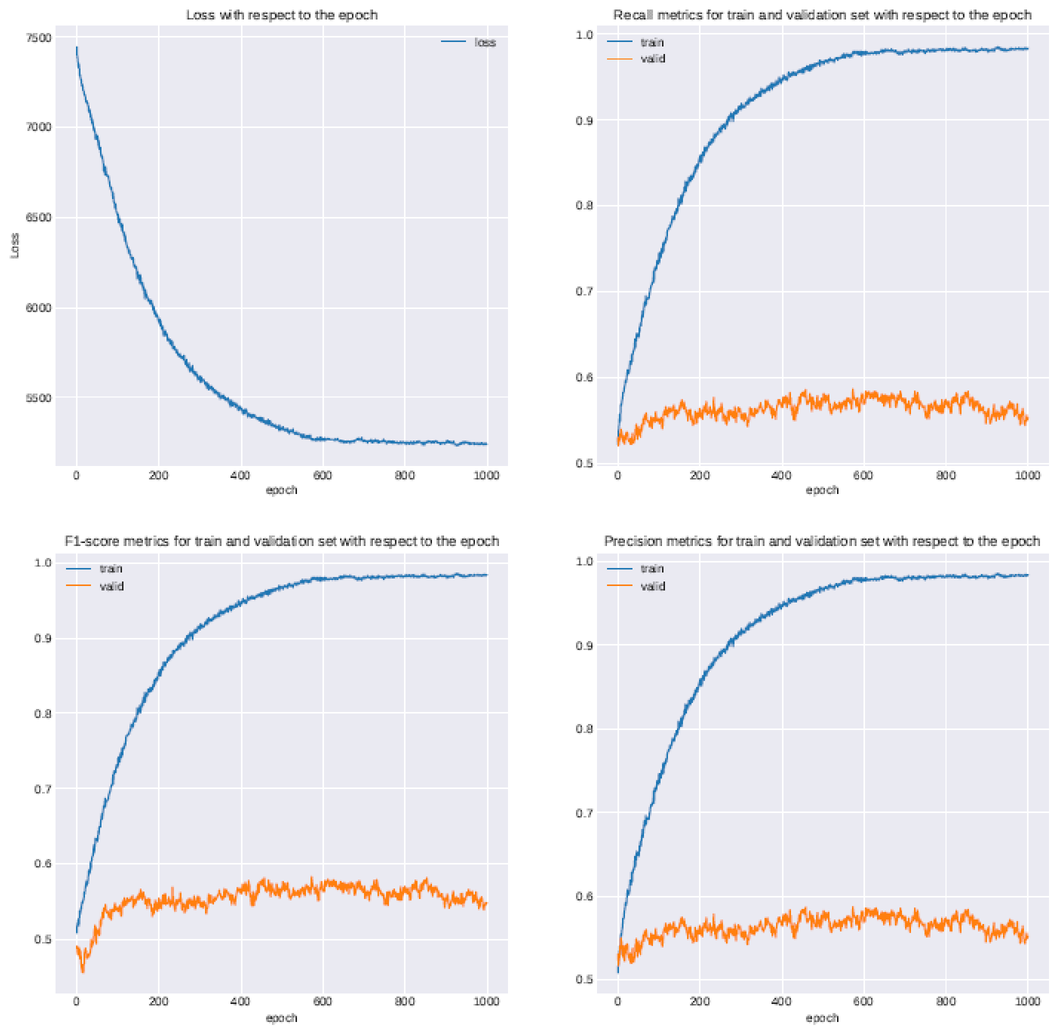


Figure 5.14: Training on 1000 epochs rather than 200

5.5.5 Testing:

The parameters used for training are given at Table 5.1. The results for all four models 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 an average precision around 55%, which is close to being a random classifier.

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

	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**.

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.

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 led to halting the loss decrease.

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 parameter's 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 Naive-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 technical and time constraints we were 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.

	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**

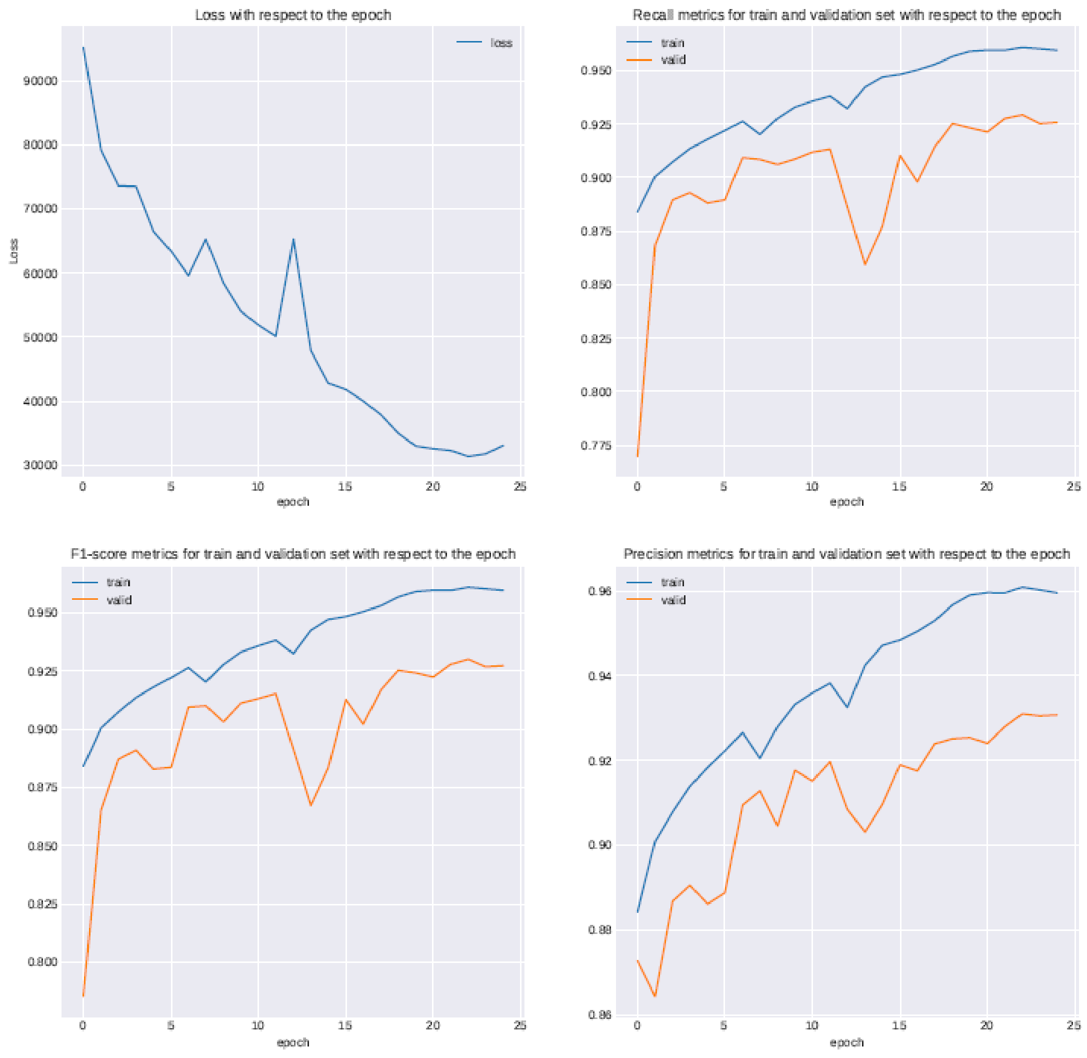


Figure 5.15: Training plots of the best LSTM using word2vec embedding.

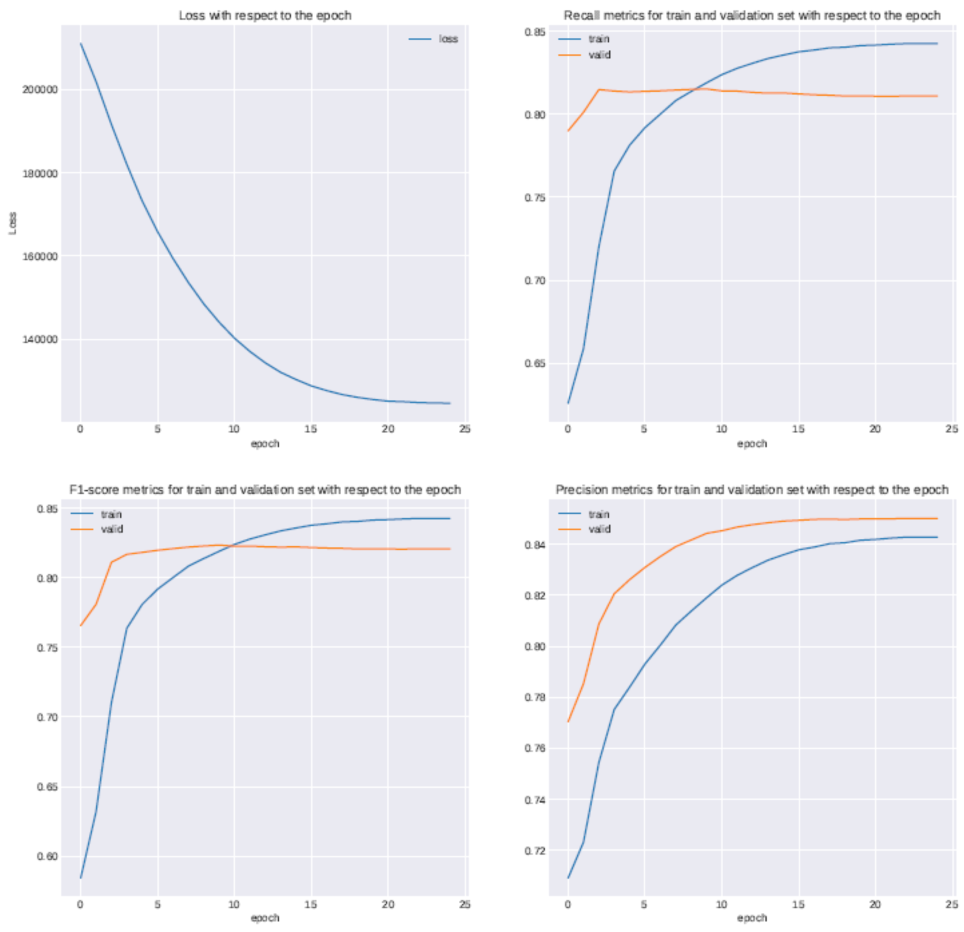


Figure 5.16: Training plots of the best attention network using word2vec embedding.

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.

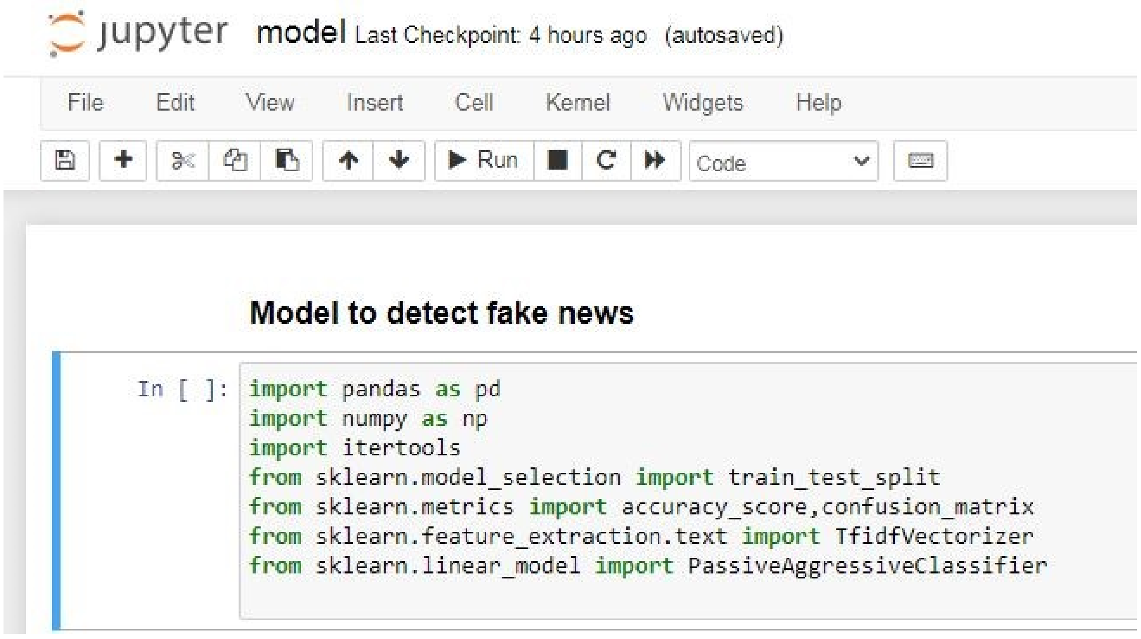
A hypothesis to explain why these two deep learning methods do not works as well as machine learning methods is the fact that in this case text are required to be the same size. Which means that some of them require some padding and the other are srunk. In the second case, information is lost.

In addition, it shows that Liar-Liar Corpus is hard to work on, with 60% precision, when Fake News Corpus still have good results.

6. IMPLEMENTATION AND OUTPUTS

1. Make necessary imports:

Screenshot:

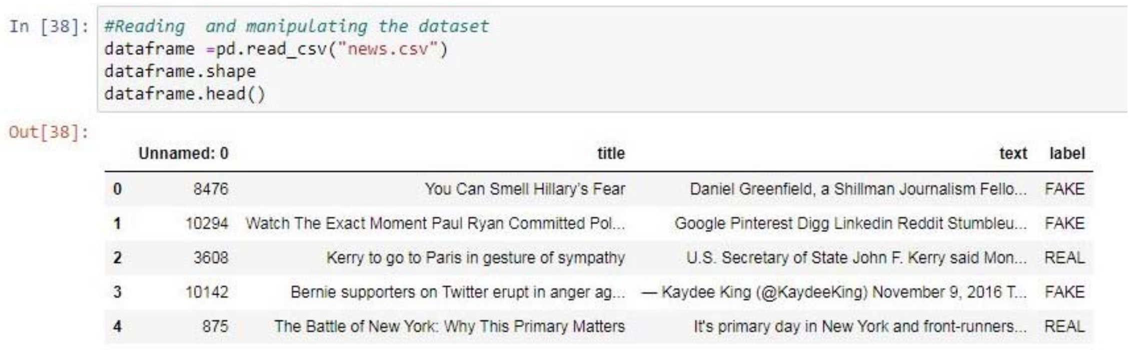

jupyter model Last Checkpoint: 4 hours ago (autosaved)
File Edit View Insert Cell Kernel Widgets Help
Run
Code

Model to detect fake news

```
In [ ]: import pandas as pd
        import numpy as np
        import itertools
        from sklearn.model_selection import train_test_split
        from sklearn.metrics import accuracy_score,confusion_matrix
        from sklearn.feature_extraction.text import TfidfVectorizer
        from sklearn.linear_model import PassiveAggressiveClassifier
```

2. Now, let's read the data into a DataFrame, and get the shape of the data and the first 5 records:

Screenshot:

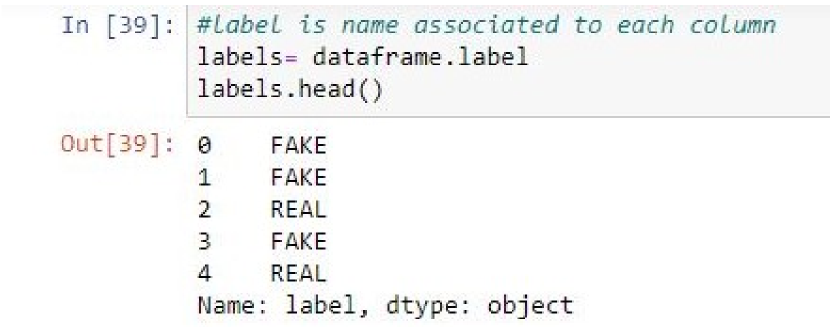


```
In [38]: #Reading  and manipulating the dataset
         dataframe =pd.read_csv("news.csv")
         dataframe.shape
         dataframe.head()
Out[38]:
```

	Unnamed: 0	title	text	label
0	8476	You Can Smell Hillary's Fear	Daniel Greenfield, a Shillman Journalism Fello...	FAKE
1	10294	Watch The Exact Moment Paul Ryan Committed Pol...	Google Pinterest Digg Linkedin Reddit Stumbleu...	FAKE
2	3608	Kerry to go to Paris in gesture of sympathy	U.S. Secretary of State John F. Kerry said Mon...	REAL
3	10142	Bernie supporters on Twitter erupt in anger ag...	— Kaydee King (@KaydeeKing) November 9, 2016 T...	FAKE
4	875	The Battle of New York: Why This Primary Matters	It's primary day in New York and front-runners...	REAL

3. Get the labels from the DataFrame.

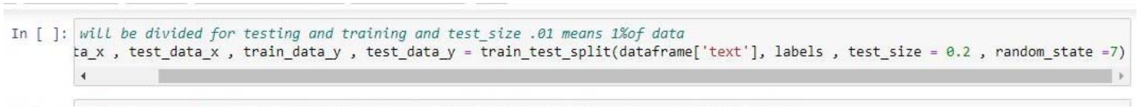
Screenshot:



```
In [39]: #label is name associated to each column
         labels= dataframe.label
         labels.head()

Out[39]: 0    FAKE
         1    FAKE
         2    REAL
         3    FAKE
         4    REAL
         Name: label, dtype: object
```

4. Split the dataset into training and testing sets.

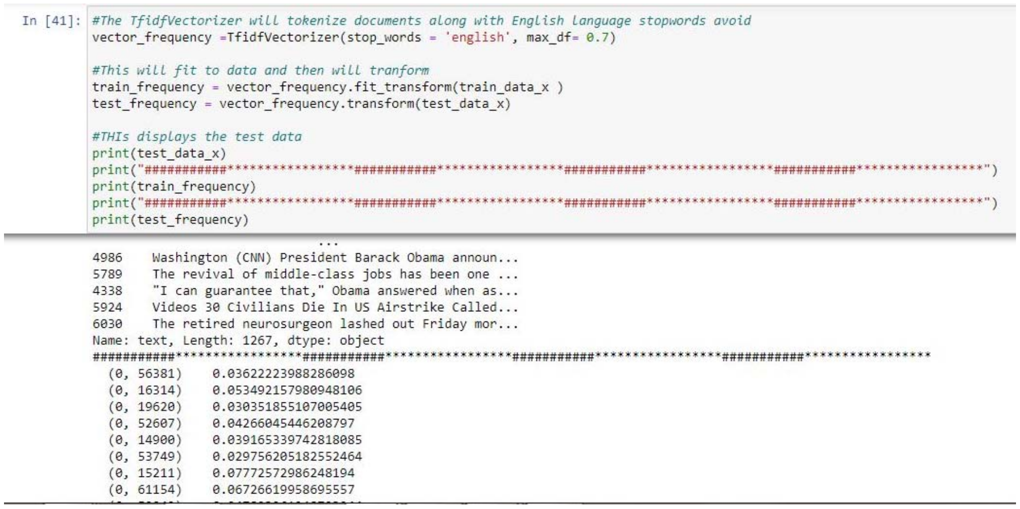
Screenshot:



```
In [ ]: will be divided for testing and training and test_size .01 means 1%of data
        ta_x , test_data_x , train_data_y , test_data_y = train_test_split(dataframe['text'], labels , test_size = 0.2 , random_state =7)
```

5. Let's initialize a TfidfVectorizer with stop words from the English language and a maximum document frequency of 0.7 (terms with a higher document frequency will be discarded). Stop words are the most common words in a language that are to be filtered out before processing the natural language data. And a TfidfVectorizer turns a collection of raw documents into a matrix of TF-IDF features. Now, fit and transform the vectorizer on the train set, and transform the vectorizer on the test set.

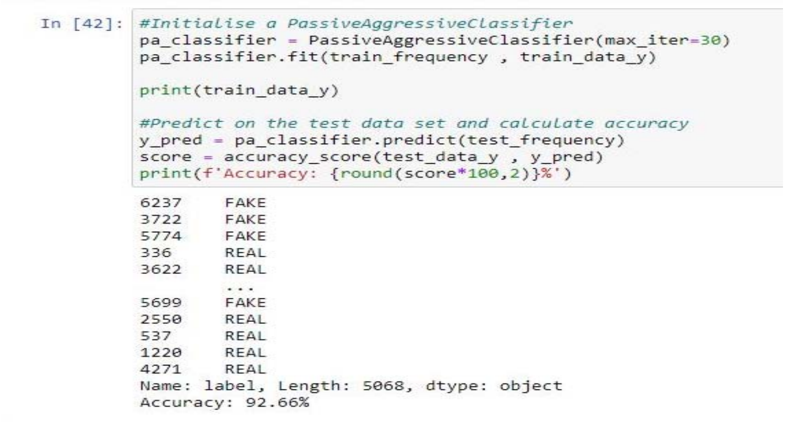
   Screenshot:



```
In [41]: #The TfidfVectorizer will tokenize documents along with English language stopwords avoid
         vector_frequency =TfidfVectorizer(stop_words = 'english', max_df= 0.7)

         #This will fit to data and then will tranform
         train_frequency = vector_frequency.fit_transform(train_data_x )
         test_frequency = vector_frequency.transform(test_data_x)

         #THIs displays the test data
         print(test_data_x)
         print("##########****************##########****************##########****************##########****************")
         print(train_frequency)
         print("##########****************##########****************##########****************##########****************")
         print(test_frequency)

                                  ...
         4986    Washington (CNN) President Barack Obama announ...
         5789    The revival of middle-class jobs has been one ...
         4338    "I can guarantee that," Obama answered when as...
         5924    Videos 30 Civilians Die In US Airstrike Called...
         6030    The retired neurosurgeon lashed out Friday mor...
         Name: text, Length: 1267, dtype: object
         ##########****************##########****************##########****************##########****************
           (0, 56381)    0.03622223988286098
           (0, 16314)    0.053492157980948106
           (0, 19620)    0.030351855107005405
           (0, 52607)    0.04266045446208797
           (0, 14900)    0.039165339742818085
           (0, 53749)    0.029756205182552464
           (0, 15211)    0.07772572986248194
           (0, 61154)    0.06726619958695557
```

6. Next, we'll initialize a PassiveAggressiveClassifier. We'll fit this on tfidf_train and y_train. Then, we'll predict on the test set from the TfidfVectorizer and calculate the accuracy with accuracy_score() from sklearn.metrics.

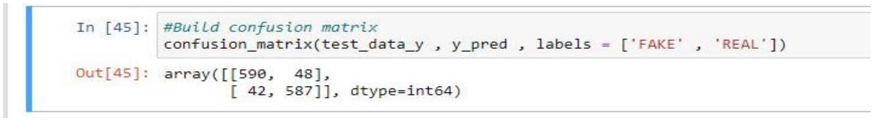
   Screenshot:



```
In [42]: #Initialise a PassiveAggressiveClassifier
         pa_classifier = PassiveAggressiveClassifier(max_iter=30)
         pa_classifier.fit(train_frequency , train_data_y)

         print(train_data_y)

         #Predict on the test data set and calculate accuracy
         y_pred = pa_classifier.predict(test_frequency)
         score = accuracy_score(test_data_y , y_pred)
         print(f'Accuracy: {round(score*100,2)}%')

         6237    FAKE
         3722    FAKE
         5774    FAKE
         336     REAL
         3622    REAL
                 ...
         5699    FAKE
         2550    REAL
         537     REAL
         1220    REAL
         4271    REAL
         Name: label, Length: 5068, dtype: object
         Accuracy: 92.66%
```

7. We got an accuracy of 92.66% with this model. Finally, let's print out a confusion matrix to gain insight into the number of false and true negatives and positives.

Screenshot:



```
In [45]: #Build confusion matrix
         confusion_matrix(test_data_y , y_pred , labels = ['FAKE' , 'REAL'])

Out[45]: array([[590,  48],
                [ 42, 587]], dtype=int64)
```

7. Conclusion

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.

In this project, we learnt to detect fake news with Python. We took a political dataset, implemented a TfidfVectorizer, initialized a PassiveAggressiveClassifier, and fit our model. We ended up obtaining an accuracy of 92.66% in magnitude.

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] Jerey 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 Jurgen 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, Andre Correia, Fabricio Murai, Adriano Veloso, Fabricio 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.