A fake news detector

Capstone Project
Udacity Machine Learning Engineer Nanodegree

Submitted: November 26, 2017

Author: Stefano Casasso

1 Definition

1.1 Project Overview

As opposed to *real news*, which is based on true facts and has the goal of informing the audience, *fake news* can be defined as "a made-up story with an intention to deceive" [1]. Fake news is probably as old as real news, but it has become a major problem with the coming of the web which made it extremely easy and fast to circulate contents among a potentially very large number of people. In particular, with the success of social networks in the last decade, fake news are spread around like a virus by people, usually with limited education and background, who share them in their public profiles.

The aim of this project is to build an algorithm to detect and flag fake news, based only on the text of the news. This is a *binary classification problem* which is studied using supervised machine learning (ML) techniques. In order to successfully apply these techniques, a sizeable "labelled" dataset is needed, preferably with a good balance between the different classes in output (fake, real).
A dataset for fake news is already publicly available on the popular data science platform Kaggle [2].
This dataset is built by crawling the news from 244 websites which are tagged as not reliable sources by the BS Detector project. BS Detector ultimately uses a manually compiled list taken from the OpenSources platform. As does Kaggle, we consider this an authority in the domain of fake news and we assume that these data are purely fake news.
For the "real" news, we use the freely downloadable dataset from webhose.io, under "English news articles". Both the fake and real news come from November 2016.

1.2 Problem Statement

The problem under study is the proliferation of fake news in the web. The solution which is described in this project is to build an algorithm which is able to distinguish between fake news and real news based on their content. This is a complex task which requires several steps.
A brief outline is given below. More details about each of these points will be discussed in Sec. 2.3 and 3.2.

- **Dataset preparation**. Pre-processing of the dataset.
- **Text-to-feature conversion**. Convert the text to numerical features to feed ML algorithms.
- **Choice of the model(s)**. Choice of the ML algorithm for the classification task.
- **Choice of the metric**. Choice of the performance metric to evaluate the algorithms.
- **Hyper-parameter fine tuning**. Optimisation of the algorithm against its configurable parameters.
- **Performance validation**. Evaluation of the performance on the test dataset.

1.3 Metrics

The dataset is composed by approximately 1/3 fake news and 2/3 real news. In the presence of unbalanced class populations, simply looking at the *accuracy* of the predictions is not sufficient,

as it will be dominated by the class with the largest population. For instance in this case a dummy algorithm which labels all the instances as "real" will have 66% accuracy on the test set.

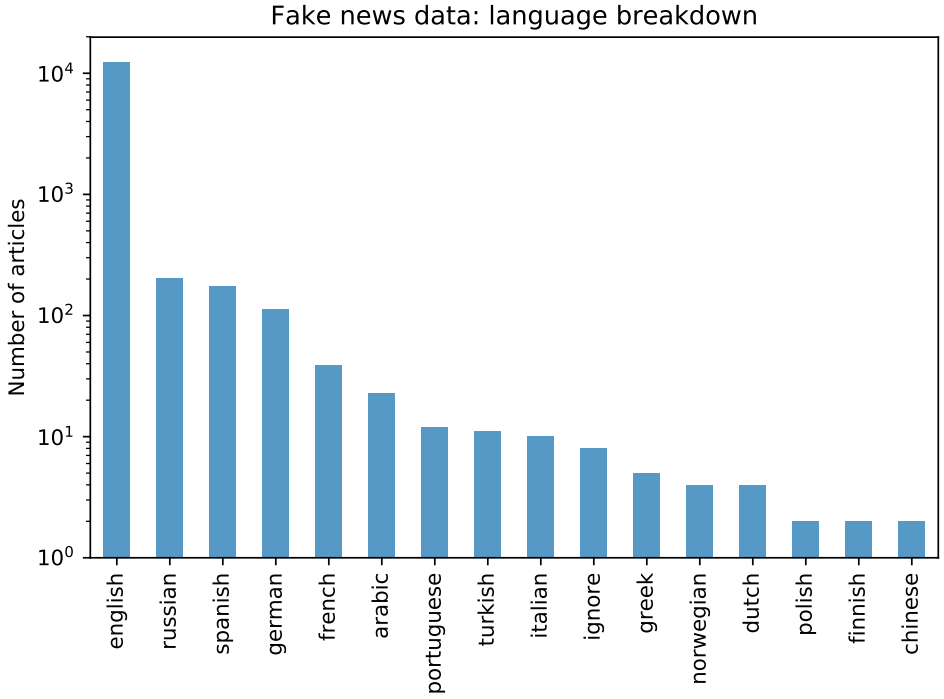
More useful metrics in similar situations are *recall*, *precision* and *F1 score*. The recall (or *sensitivity*) [3] is the fraction of fake news that are retrieved while the precision is the fraction of correctly flagged fake news. The F1 score [4] combines both into one single number, defined as the harmonic average of recall and precision. For this study, all these metrics are important and thus are reported. However, the F1 score will be used to pick the best model and to optimise its hyperparameters.

2 Analysis

2.1 Data Exploration

The dataset used in this project are introduced in Sec. 1.1. The instances belong to two different classes, corresponding to the output labels that the algorithm will eventually predict, i.e. fake/real news.
In total there are 37117 articles, of which 24118 are real news and 12999 are fake news. 2162 rows don't have an entry in the `text` column, 2116 labelled as real and 46 labelled as fake. Since the algorithm needs this attribute (and only this one) to make predictions, these rows are removed from the dataset (see also Sec. 3.1).
All the articles labelled as real news are in English (`language` attribute), while this is not the case for the fake news, as shown in Fig. 1.

Figure 1: Breakdown of the article language for the "fake news" instances.



Name	Type	Description	Comments
author	str	Author of the post	-
comments	int	Number of comments to the post	more in "Exploratory Visualization"
country	str	Country of the post	-
crawled	str	Date/time/timezone crawling information	-
domain_rank	float	NA	-
label	int	1 for fake, 0 for real	Output label
language	str	Language of the post	-
likes	int	Number of likes to the post	more in "Exploratory Visualization"
main_img_url	str	Url of the main image in the article	-
ord_in_thread	int	NA	-
participants_count	int	Number of participants to the thread	more in "Exploratory Visualization"
published	str	Date/time/timezone crawling information	-
replies_count	int	Number of replies to the post	more in "Exploratory Visualization"
shares	int	Number of shares	more in "Exploratory Visualization"
site_url	str	Url of the website of the post	-
spam_score	float	NA	-
text	str	Text of the news	Discriminating feature
thread_title	str	NA	-
title	str	Title of the post	more in "Exploratory Visualization"
uuid	str	NA	-

Table 1: Summary of the 20 columns present in the dataset.

Since the algorithm is based on word frequency, it seems like a good idea to discard all the non-English articles from the dataset ($\sim$2% of the total), as described also in Sec. 3.1.

After this selection, the dataset consists of 34359 rows, 22002 real news articles and 12357 fake news articles. The size of the dataset is about 120 MB.
Table 1 summarises the columns of the dataset. Most of them are not used in any stage of this project, but are kept for consistency. For some columns the description is not provided, because not known or not completely clear to the author. Some of the variables are visualized in Sec. 2.2.

2.2 Exploratory Visualization

Even though only one feature in the dataset is actually used for discrimination in the classification task, it's nonetheless interesting to visualise some of the other columns in the dataset (see Tab. 1). In doing so, the fake/real news are plotted separately to better understand the different trends across the two classes.
In Fig. 2 the number of characters in the text and title of the article are shown. Two observations are worth mentioning:

- Many fake news have a short text. This is somehow expected, as there are not many arguments to describe to support a made up story...
- Fake news tend to have slightly longer titles. One possible explanation for this feature is the average lower level of professionalism among the writers of fake news: a professional journalist, in fact, knows that titles should be short to be more effective.

In Fig. 3 the number of participants to the thread, replies to the thread and likes are respectively

shown. The only striking difference is about the number of likes, which is significantly larger, as one would expect, for real news compared to fake news.
Finally in Fig. 4 two examples of "word cloud" plots are shown, where the words in the document are arranged in order to enphasise the most frequent ones. These plots are generated using the `wordcloud` package [5].

Figure 2: The number of characters in the article text (left) and title (right).

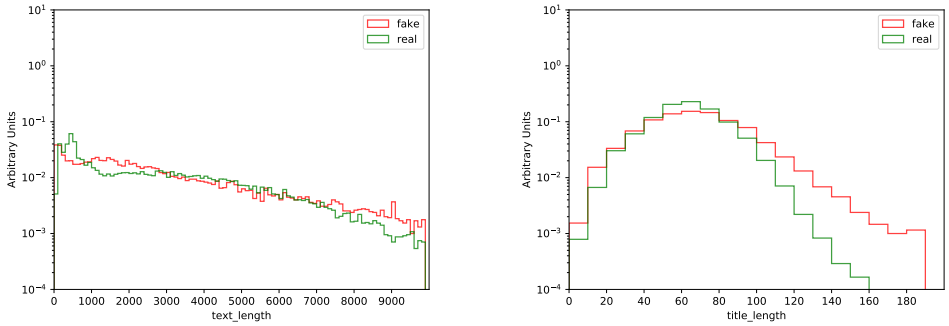


Figure 3: The number of participants in the thread (top left), replies to the thread (top right) and likes to the post (bottom).

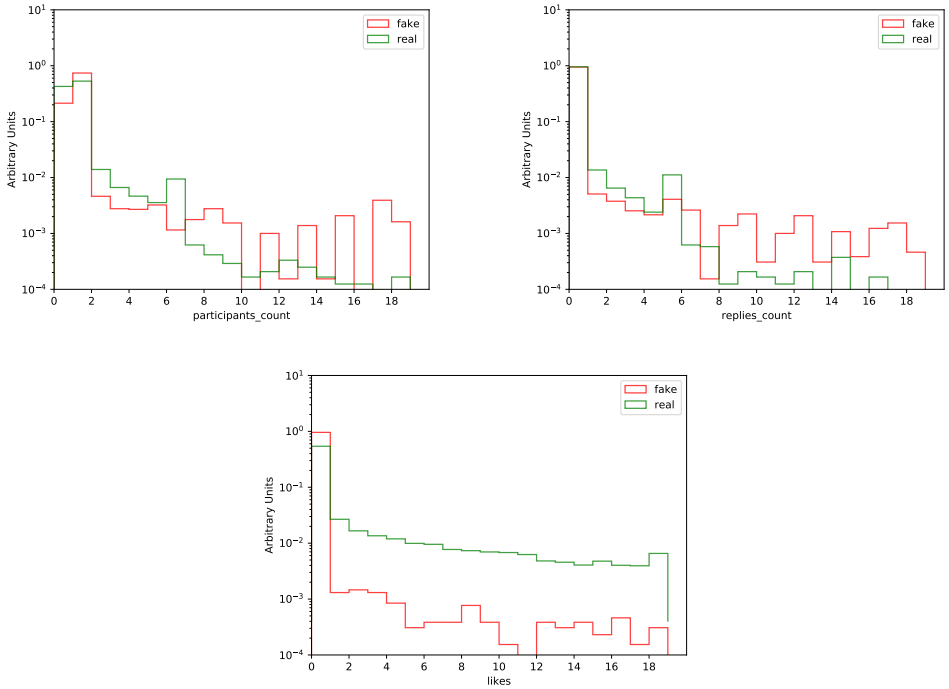
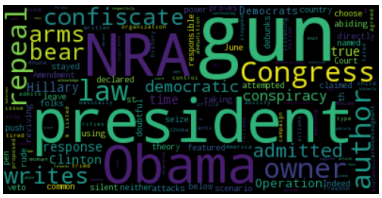
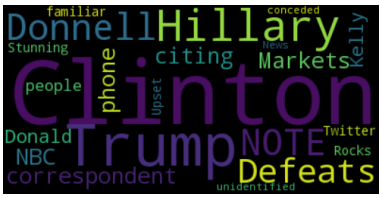


Figure 4: Two examples of "word cloud" plots from a fake (left) and real (right) news.





2.3 Algorithms and Techniques

As mentioned in Sec. 1.2, several steps involving different techniques are implemented, which will be thoroughly described in this section.

The very first transformation has to convert the text into numerical features, as most of the ML algorithms work only with numerical inputs. In this project we chose a popular technique called **Term Frequency-Inverse Document Frequency** or simply **TFIDF** [6]. The TFIDF transformation allows not only to vectorize the text (i.e. the rows of the dataset) into numerical features but naturally filters the most frequent words in the corpus, as it is explained in the following.
The default `scikit-learn` implementation of TFIDF used in this project defines:

$$TFIDF(t,d) = TF(t,d) \times IDF(t) \tag{1}$$

where the term frequency (TF) of term t appearing f_t times in document d is defined as [1]:

$$TF(t,d) = 1 + log(f_t) \tag{2}$$

and the inverse document frequency is defined as:

$$IDF(t) = log\frac{1+N}{1+n_t} + 1 \tag{3}$$

with N being the total number of documents in the corpus and n_t the number of documents containing the term t.
After TFIDF is applied, the word vectors are normalized using the usual euclidean definition.
In order to reduce the number of features, the vocabulary is built discarding the 30% of the most used terms in the corpus.

Another important ingredient is the ML algorithm to use for the classification. We decided to test 3 different algorithms in order to cover different techniques with different assumptions and training complexity.

[1] this definition of TF with the damping logarithm is referred to as *sublinear TF* and is meant to reduce the importance of terms with high frequency.

Multinomial Naive Bayes

Naive Bayes (NB) algorithms are frequently used in text classification, like spam detection, because, despite their simplistic assumptions, they work quite well in practice. They are also very fast to train.

NB use the Bayes' theorem of probability theory assuming that features are independent ("naive"). This leads to the following classification rule:

$$\hat{y} = \arg\max_y P(y) \prod_{i=1}^{n} P(x_i|y) \tag{4}$$

where n is the total number of features (i.e. words in the dictionary), $\hat{y}$ is the predicted class, $P(y)$ is the probability of class y (*prior probability*) and $\prod_{i=1}^{n} P(x_i|y)$ is the *likelihood* of the instance $(x_1, ..., x_n)$ given the hypothesis of class y.
In Multinomial NB the likelihood of the features is a multinomial distribution for the general multi-class problem (in this case a binomial distribution, since there are only 2 classes). The parameters of the multinomial distribution are estimated on the training dataset as relative word frequencies, with the addition of a smoothing factor α:

$$\hat{\theta}_{y,i} = \frac{N_{y,i} + \alpha}{N_y + \alpha n} \tag{5}$$

where $N_{y,i}$ is ther number of times feature i appears for the class y and $N_y = \sum_i^n N_{y,i}$. The hyper-parameter α avoids null probabilities and accounts for features not present in the training set.
Once these parameters are learnt in the training, they are used to define the likelihood and the predicted class is the one which maximise the product in Eq. 4. Notice that when flat prior is specified, this is equivalent to maximise the likehood term.

Logistic Regression Classifier

Linear classifiers are usually a good compromise between complexity and training time and therefore are a good benchmark to compare against more complicated algorithms.

The decision function of linear classifiers is a weighted sum of the input features plus a "bias term":

$$y = b + \sum_{i=1}^{n} w_i x_i \tag{6}$$

The weight factors w_i are learnt during the training step.
In the case of logistic classification [7], the output of Eq. 6 is passed as argument to the *logistic function*:

$$\sigma(t) = \frac{1}{1 + e^{-t}} \tag{7}$$

The output of the logistic function is a number between 0 and 1, which is interpreted as the probability of class y.
Therefore the classification rule is:

$$\hat{y} = \arg\max_y \sigma(b + \sum_{i=1}^{n} w_i x_i) \tag{8}$$

Linear models are usually regularized by adding a penalty term to the cost function, which control the size of the weights (*Ridge*, *Lasso*, *ElasticNet*, etc.).

Random Forest

Random forests [8] are among the most powerful predictive models and have a wide range of applications. The RF can potentially captures more complex patterns in the dataset at the prize of a longer training and higher parameter tuning complexity.

RF are ensemble of many decision trees, where the output of the classification task is the mode of the classes among all the classifiers. Each tree in the ensemble is trained on a different subset of the training set ("bagging"), in order to avoid correlation between the prediction of each classifier. In addition, at each split only a random subset of features is considered: this procedure further reduces possible correlations between the trees, especially in presence of few dominant features.
The trees in the forest are usually highly regularized, in order to both reduce the training time and the variance of the prediction. This can be achieved in several different ways: by limiting the depth of the tree, by requiring a minimum gain at each split, etc.

The three classifiers are finally combined in a *voting classifier* [9]. A voting classifier is an ensemble of different classifiers, where the prediction is the mode of the predictions across the ensemble ("hard voting"). When all the classifier can estimate class probabilities, "soft voting" can also be used, which predicts the class with the highest probability across the classifiers.
The **ensemble classifier** usually gives better performances (smaller variance) compared to each single classifier in the ensemble when there is little or none correlation across the different predictors (different assumptions, different training set, etc.).

2.4 Benchmark

Some attempts have already been perfomed to detect fake news using ML algorithms [10, 11, 12]. In these projects, typical F1 scores on the test set range from 80% to 95%. Similar perfomances are expected also as a result of this project.

3 Methodology

3.1 Data Preprocessing

Since the real news dataset has only articles in English, in the Kaggle dataset (fake news) only the articles in English are retained (about 95% of the total).
The real news dataset is huge and consists of about 500k news articles, in the (quite inconvenient) format of a single json file per news. We apply an additional processing on top of these corpus, which consists of producing a single file and restricting the sources of news to the following list: *New York Times*, *Washington Post*, *The Washington Journal*, *Reuters*, *The Guardian*, *Forbes*, *BBC*, *NPR* and *Bloomberg*. These websites are universally recognised as trustable sources of information and we are confident that they represent an unbiased sample of real news.
As the number of news remaining is still much larger than the fake news ($\sim$100k vs. $\sim$13k), we further skim by "capping" each source to a maximum of 3k articles, chosen randomly. This procedure ensures that there is no source which dominates the others, thus making the dataset more balanced (we noticed that a sizeable fraction of news come from Reuters alone). After this additional step we are left with about $\sim$24k articles.
A few articles have NAs in the `text` column and are also removed.

Finally, the labels are "binarized" as follows:

- **fake news** are assigned `label=1`
- **real news** are assigned `label=0`

The dataset consists of 20 columns. Only the `text` is used as discriminating feature in the algorithm. However, the other variables are also used in the exploratory phase, see Sec. 2.1 and 2.2.

3.2 Implementation

The data is manipulated using `pandas` [13]. The implementation of ML models and metrics relies on the `scikit-learn` API [14].

Once the data is loaded, the first step is to divide it into *training* and *testing* datasets. The common choice of retaining 20% of the data for testing is adopted.
The article text is then transformed into numerical features according to the TFIDF procedure described in Sec. 2.3. The `sklearn.feature_extraction.text.TfidfVectorizer` [15] class is used, with the following arguments:

```
vectorizer = TfidfVectorizer(max_df=0.7,
                             stop_words='english')
```

`max_df` is the fraction of words in the dictionary which are used as features: in this implementation the 30% most frequently used words will be discarded. The `stopwords` option select the English list of "stop-words" implemented in the package.

Then the Multinomial Naive Bayes (MNB) classifier is defined
(class `sklearn.naive_bayes.MultinomialNB` [16]) using the default parameters and fitted to the training data.

The Logistic Regression (LRE) classifier is defined using the class
`sklearn.linear_model.LogisticRegressionCV` [17]. All the parameters of the constructor are the default ones.

For the implementation of the Random Forest (RFO) classifier the class
`sklearn.ensemble.RandomForestClassifier` [18] is used. Since there is a very large number of features and samples, the trees of the forest can potentially grow very complicated. Hence, a high level of regularization is needed. The first attempt of parameter set, which is tweaked afterwards using grid search, is as follows:

```
clf_rndf = RandomForestClassifier(class_weight='balanced_subsample',
                                  n_estimators=30,
                                  max_depth=20,
                                  min_samples_split=20,
                                  max_features=0.01)
```

This choice is driven by the need to:

- account for unbalance across classes (`class_weight`)
- reduce the number of input features (`max_features`)

Model	Parameter	Best fit value
MNB	alpha	0.2
	fit_prior	False
LRE	C	20.4
	dual	True
	fit_intercept	True
	penalty	'l2'
RFO	n_estimators	100
	max_features	0.02
	max_depth	100
	min_samples_split	50

Table 2: Best fit values of the floating parameters in the models.

- limit the size of the forest (`n_estimators`)
- regularize the trees (`max_depth`, `min_samples_split`)

The metrics described in Sec. 1.3 are computed using the implementation in the `sklearn.metrics` module [19].

The `scikit-learn` package is remarkably well designed and documented, making it easy to implement the techniques discussed in this project. Without any particular challenge on the coding part, the main difficulty in the implementation was on the computing side: the optimisation of the parameters required a long time and a fair amount of resources.
First, coarse grids have been used to have a rough estimate of the best parameter values and also to identify the most sensitive parameters. Secondly, a finer grid has been defined to fine-tune the parameters around the output of the previous step.
To speed up this process, we recommend to use all the cores available (`n_jobs=-1`) and to save the models to file once the training is done.

3.3 Refinement

This section describes how the models implemented as of Sec. 3.2 are fine-tuned and how the ensemble of classifier is defined.

The tuning of the hyperparameters is done using the powerful `sklearn.model_selection.GridSearchCV` class [20]. The hyper-parameters in the grids and their best fit values are reported in Tab. 2.

Two approaches for the ensemble are considered:

- Ensemble 1 (EN1): "hard" voting, i.e. majority voting, including all the 3 classifiers.
- Ensemble 2 (EN2): "soft" voting, i.e. voting weighted by probabilities, including the LRE and RFO classifiers. In this case the MNB is excluded because the probabilities computed by naive bayes algorithms are known to be not trustable.

All the solutions and their performances are presented in Tab. 3. Based on these results, the chosen final model is EN2.

Classifier	Accuracy	Precision	Recall	F1 score
before optimisation				
MNB	0.84	0.56	0.97	0.71
LRE	0.94	0.93	0.91	0.92
RFO	0.88	0.91	0.79	0.85
after optimisation				
MNB	0.90	0.87	0.86	0.86
LRE	0.94	0.93	0.91	0.92
RFO	0.93	0.89	0.91	0.90
ensembles				
EN1	0.94	0.92	0.91	0.92
EN2	0.95	0.92	0.92	0.92

Table 3: Performances of the 3 classifiers (before and after optimization) and the 2 ensemble classifiers (in red the final model).

4 Results

4.1 Model Evaluation and Validation

All the results for intermediate and final solutions, based on the test set, are presented in Tab. 3. It can be noticed that the optimisation improved largely the results of the MNB model and the RFO model. The LRE model instead is not improved significantly.
The EN2 has slightly better perfomances, with F1-score of 92% and it's chosen as the final model.
Overall the performances are very good and satisfactory also compared to the benchmarks reported in Sec. 2.4.

To further assess the validity of the model, an additional *sensitivity test* is carried out: in each article of the dataset one sentence is removed, chosen randomly. The results prove to be stable against this check: in particular the F1-score is still 92% even after this perturbation.

4.2 Justification

The final model is a soft-voting ensemble classifier which consists of a Logistic Regression model combined with a Random Forest model, both highly regularized. The final solution has an F1-score of 92% on the test set, with perfectly balanced recall and sensitivity. These performances are in line with the ones reported in the benchmarks [11, 12].

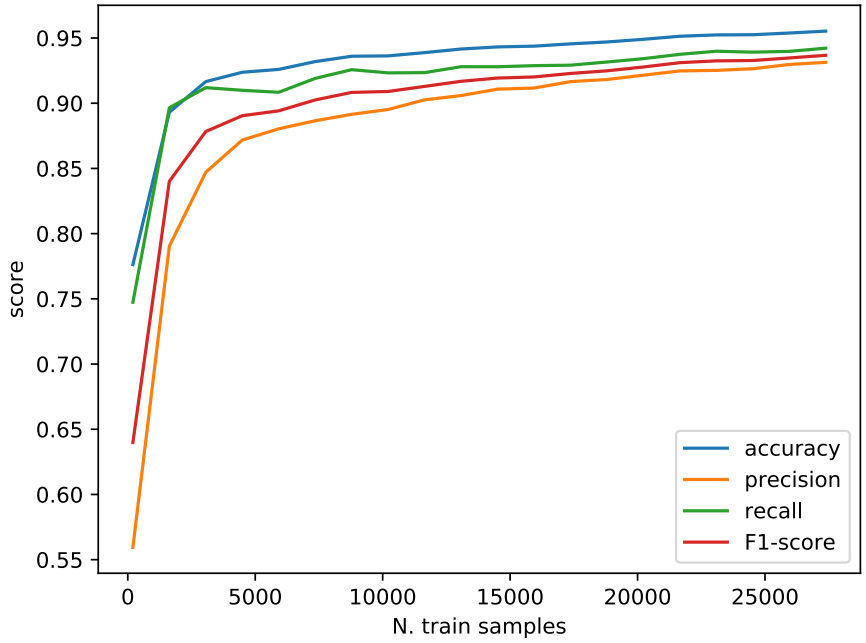
Despite the good results obtained with this dataset, we can't claim that the problem of fake news is solved by this algorithm. There are in fact limitations to its use that are further discussed in Sec. 5.3.

5 Conclusion

5.1 Free-form visualization

One of the most important characteristics of ML models is the number of samples needed for training. In fact, everytime the model has to be re-trained on new data, it's crucial to know the number of samples in the training set which is needed to achieve optimal perfomances. In Fig. 5 several different scores are shown as a function of the number of samples included in the training dataset, namely accuracy, precision, recall and F1-score (see Sec. 1.3). In this analysis the test set is the same for each training session. Looking in particular at the F1-score, which is

Figure 5: Classification scores as a function of the number of training samples.



the main metric used in the project, the following observations can be made:

- with the available size of training dataset there seems to be no complete "saturation" of performances, as the scores don't seem to reach a flat "plateau". However, the increase in performances is very slow above about 10000 training samples.
- on the other hand, the number of training samples needed to achieve good perfomances is quite small: with about 1500 samples the F1-score is already 85%.

5.2 Reflection

In this project the problem of detecting the so-called "fake news" has been studied. A ML algorithm has been developed which by analyzing the text of the article returns the result of the binary classification: fake news/real news.
First, a suitable dataset of fake news and real news have been identified and downloaded. Then

the article text in each instance has been converted to a vector of numerical features, representing a set of weights which take into account the frequency of the word in the document and in the corpus (TFIDF vectorization). The resulting "word vectors" have been feeded to 3 different ML models which are compared using the appropriate metrics. An ensemble including the best 2 classifiers is used as final model.

There are several interesting aspects of this project. In particular the following points have been very stimulating challenges to solve.

- **Find the datasets**
  While the idea for this project came when we stumbled upon the Kaggle fake news dataset, the real news dataset has been more difficult to collect. Several news website do not provide an API to fetch historical articles and most of them don't provide the access to the full article. Only after quite some research, we found the webhose.io dataset which fit all these criteria.

- **Feature extraction**
  Being the first project of "text analysis" that I carry out, I was lacking some knowledge about how to transform the text into features. It took me some time to go through the different options (*bag-of-words*, *word2vec*, *TFIDF*) and understand which one was the best for this case.

- **Model regularization**
  The regularization of the random forest model has been very challenging. In fact, in text analysis the number of features grows very large and a high level of regularization is needed for complicated models. With long training time and different combinations of parameters to validate, it took quite some effort to achieve an optimal solution.

5.3 Improvement

There is room for improvement both in the model itself and in its validation. Some ideas are described below: their implementation is, however, outside the scope of this project.

Possible improvements on the algorithm side are:

- use different, more "complex" models, like **deep neural networks**;
- analyze **n-grams**, i.e. ensemble of n words;
- incorporate also the **headline** of the article, together with the text;

On the validation side, the main limitation is that training/testing data belong to approximately the same time frame. It should be checked how the performaces change when this assumptions is relaxed. If the perfomances degrade, one should make sure that the model is trained on data that is relevant to the new instances that are being classified. This is not simple to handle from the technical point of view: one has to fetch much more data than what has been used in this project and probably needs to build a set of pre-trained models which cover a relevant time range.

References

[1] S. Tavernise, "As fake news spreads lies, more readers shrug at the truth." https://www.nytimes.com/2016/12/06/us/fake-news-partisan-republican-democrat.html, 2016.

[2] Kaggle, "Getting real about fake news: Text and metadata from fake and biased news sources around the web." https://www.kaggle.com/mrisdal/fake-news, 2017.

[3] Wikipedia, "Precision and recall." https://en.wikipedia.org/wiki/Precision_and_recall.

[4] Wikipedia, "F1 score." https://en.wikipedia.org/wiki/F1_score.

[5] A. C. Mueller, "word_cloud." https://github.com/amueller/word_cloud/tree/master/examples.

[6] Wikipedia, "tf-idf." https://en.wikipedia.org/wiki/Tf-idf, 2017.

[7] Wikipedia, "Logistic regression." https://en.wikipedia.org/wiki/Logistic_regression.

[8] Wikipedia, "Random forest." https://en.wikipedia.org/wiki/Random_forest.

[9] scikit learn, "Ensemble methods." http://scikit-learn.org/stable/modules/ensemble.html.

[10] Y. Genes, "Detecting fake news with nlp." https://medium.com/@Genyunus/detecting-fake-news-with-nlp-c893ec31dee8, 2017.

[11] J. Goldstein, "Identifying "fake news" with nlp." https://blog.nycdatascience.com/student-works/identifying-fake-news-nlp/, 2017.

[12] K. Jarmul, "Detecting fake news with scikit-learn." https://www.datacamp.com/community/tutorials/scikit-learn-fake-news.

[13] W. McKinney, "Data structures for statistical computing in python," in *Proceedings of the 9th Python in Science Conference* (S. van der Walt and J. Millman, eds.), pp. 51 – 56, 2010.

[14] 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*, vol. 12, pp. 2825–2830, 2011.

[15] scikit learn, "Tf-idf vectorizer." http://scikit-learn.org/stable/modules/generated/sklearn.feature_extraction.text.TfidfVectorizer.html.

[16] scikit learn, "Naive bayes classifier for multinomial models." http://scikit-learn.org/stable/modules/generated/sklearn.naive_bayes.MultinomialNB.html.

[17] scikit learn, "Logistic regression cv." http://scikit-learn.org/stable/modules/generated/sklearn.linear_model.LogisticRegressionCV.html.

[18] scikit learn, "Random forest classifier." http://scikit-learn.org/stable/modules/generated/sklearn.ensemble.RandomForestClassifier.html.

[19] scikit learn, "Metrics." http://scikit-learn.org/stable/modules/classes.html#sklearn-metrics-metrics.

[20] scikit learn, "Gridsearchcv." http://scikit-learn.org/stable/modules/generated/sklearn.model_selection.GridSearchCV.html.

[21] A. H. Bloomberg, "Facebook has a new plan to curb 'fake news'." http://fortune.com/2017/08/03/facebook-fake-news-algorithm/, 2017.

[22] Wikipedia, "Naive bayes." https://en.wikipedia.org/wiki/Naive_Bayes_classifier.