Progetto di Sistemi Intelligenti per Internet e Machine Learning

Fake News Detection: Multinomial Naive Bayes classifier applied to a dataset constructed from American newspapers' articles

Alessandra Flaccavento

Matricola: 472445

Abstract

The purpose of this project was to train a classifier to distinguish an actual news (Ham) from a fake one (Spam). This was achieved starting from gathering the information and up to showing the most common words for both cases of news.

The macro-steps followed throughout the project were:

- Data gathering
- Data cleaning
- TF-IDF text representation and indexing
- Multinomial Naive Bayes, Linear Support Vector, Decision Tree , Logistic Regression and Stochastic Gradient Descent classifiers training on the train and test sets built
- WordCloud representation

The code is available at the following repository:

https://github.com/Tireswind/SII-ML-project

Data gathering

The first step was creating a JSON file which included several American newspapers' websites. Whenever possible, the RSS link of the website was also included, as RSS feeds often make available more consistent and correct data.

A limit of articles to be retrieved from each website is defined: then, the algorithm iterates through each news company site. A check on the availability of the publish date is performed: if this piece of information does not exist, the article is skipped. This was done to keep consistency in the data. After ten downloaded articles from the same newspaper without publish date, the entire source is skipped.

Finally, the articles are saved in a JSON file.

Code:

- https://github.com/Tireswind/SII-ML-project/blob/master/NewsPapers.json
- https://github.com/Tireswind/SII-ML-project/blob/master/news_retriever.ipynb
- https://github.com/Tireswind/SII-ML-project/blob/master/scraped_articles.json

Data cleaning

All of the scraped articles were used to create a Pandas DataFrame, in which every row stored an article.

Pandas is a Python library that provides high-performance, easy-to-use data structures: among these, there are DataFrames. A DataFrame is a two-dimensional size-mutable, potentially heterogeneous tabular data structure with labeled axes (rows and columns).

In this DataFrame, a column for the label of the article was created: a label of 0 is ham, a label of 1 is spam. Then, a custom Python module was responsible for cleaning the input texts. Here, the Natural Language Toolkit (NLTK), an open source Python library for Natural Language Processing, was used to collect stopwords for the English language. Other than tokenizing the text, additional transformations were applied in order to have a better cleaned input text, such as the removal of punctuation and the extended interpretation of a small set abbreviations. The resulting text was saved as a character stream through the Pickle module to be able to save the dataset for future improvements.

Python pickle module is used for serializing and de-serializing a Python object structure. Pickling is a way to convert a Python object (list, dict, etc.) into a character stream. The idea is that this character stream contains all the information necessary to reconstruct the object in another Python script.

Code:

- https://github.com/Tireswind/SII-ML-project/blob/master/dirty_df.pkl
- https://github.com/Tireswind/SII-ML-project/blob/master/cleaned_df.pkl
- https://github.com/Tireswind/SII-ML-project/blob/master/cleaning_helper1.py
- https://github.com/Tireswind/SII-ML-project/blob/master/read_and_clean_news.ipynb

Model and Classifier

In the third part of the project, the first step was preparing the target and predictors for modeling.

The convertion of the text into a TF-IDF was performed. TF-IDF, short for term frequency–inverse document frequency, is a numerical statistic that is intended to reflect how important a word is to a document in a collection or corpus. The value increases proportionally to the number of times a word appears in the document and is offset by the number of documents in the corpus that contains the word, which helps to adjust for the fact that some words appear more frequently in general.

Then, the SciKit library provides a tool, called the Model Selection library. There is a class in the library which is, aptly, named 'train_test_split': this class was used to easily split the dataset into the training and the testing sets. As a common practice in data science, the dataset was split in a 80-20 ratio. To try to understand how the results could have varied upon changing the size of the sets, later the models were also built on a 60-40 ratio.

From the same library, at first, a classifier was also imported: 'MultinomialNB'. The Multinomial Naive Bayes is mostly used classifier for document classification problems. The features/predictors used by this classifier are the frequency of the words present in a document: this is the reason behind the TF-IDF conversion. Such conversion was also used to train other classifiers (*please, refer to the explaination that will shortly follow*).

After training the classifier, the F1 and the Accuracy scores were computed on both train and test sets. Of course there were a few misclassified articles (Ham misclassified as Spam).

In order to have a taste of different classifiers' behavior, some modifications in the parameters and additional classifiers were also applied:

- Linear Support Vector → a Support Vector Machine model is a representation of the examples as points in space, mapped so that the examples of the separate categories are divided by a clear gap that is as wide as possible; this is the linear variant

- Decision Tree → from observations about an item (represented in the branches) to conclusions about the item's target value (represented in the leaves). In this case, as a decision tree, leaves represent class labels and branches represent conjunctions of features that lead to those class labels

- Logistic Regression → a statistical model that uses a logistic function to model a binary dependent variable (as it happens in this case), although many more complex extensions exist

- Stochastic Gradient Descent → (SGD) is an iterative method for optimizing an objective function with suitable smoothness properties (e.g. differentiable or subdifferentiable). It can be regarded as a stochastic approximation of gradient descent optimization, since it replaces the actual gradient (calculated from the entire data set) by an estimate thereof (calculated from a randomly selected subset of the data)

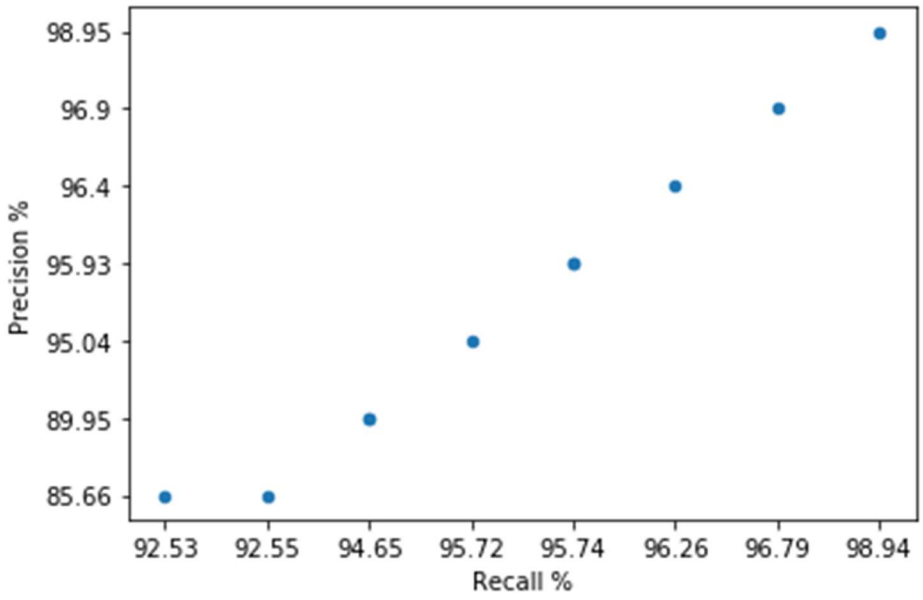
The following table shows each classifier with the values used for each one of their parameters:

Classifier	Parameters			
Multinomial Naive Bayes	*alpha=1.0*	*fit_prior=True*	*fit_prior=False*	*class_prior=None*
Linear Support Vector	C=1.0 intercept_scaling=1 penalty='l2'	class_weight=None loss='squared_hinge' random_state=None	dual=True max_iter=1000 tol=0.0001	fit_intercept=True multi_class='ovr' verbose=0
Decision Tree	criterion='gini' max_depth=None presort='deprecated' splitter='best'	min_samples_split=2 min_weight_fraction_leaf=0.0 ccp_alpha=0.0 random_state=None	max_features=None max_leaf_nodes=None min_impurity_split=None	min_impurity_decrease=0.0 class_weight=None min_samples_leaf=1
Logistic Regression	C=1.0 intercept_scaling=1 n_jobs=None tol=0.0001	class_weight=None l1_ratio=None penalty='l2' verbose=0	dual=False max_iter=100 random_state=0 warm_start=False	fit_intercept=True multi_class='auto' solver='lbfgs'
Stochastic Gradient Descent	alpha=0.0001 epsilon=0.1 loss='hinge' n_jobs=None shuffle=True warm_start=False	average=False eta0=0.0 learning_rate='optimal' penalty='l2' tol=0.001	class_weight=None fit_intercept=True max_iter=1000 power_t=0.5 validation_fraction=0.1	early_stopping=False l1_ratio=0.15 n_iter_no_change=5 random_state=None verbose=0

Here are reported the resulting metrics, for each classifier:

	TEST set size %	F1 %	Precision %	Recall %	Accuracy %	Misclassified articles
Decision Tree	20	98.9	98.95	98.94	98.94	[88]
Decision Tree	40	98.84	95.04	95.72	95.72	[3,19,50,62,79,99,108,184]
Stochastic Gradient Descent	40	96.13	96.9	96.79	96.79	[3,50,62,79,108,184]
Linear Support Vector	40	95.29	96.4	96.26	96.26	[3,50,62,79,99,108,184]
Linear Support Vector	20	94.94	95.93	95.74	95.74	[3,50,62,79]
Stochastic Gradient Descent	20	94.94	95.93	95.74	95.74	[3,50,62,79]
Multinomial Naive Bayes	40	92.05	89.95	94.65	94.65	[3,47,50,62,79,80,88,99,108,184]
Logistic Regression	40	92.05	89.95	94.65	94.65	[3,47,50,62,79,80,88,99,108,184]
Multinomial Naive Bayes	20	88.97	85.66	92.55	92.55	[3,47,50,62,79,80,88]
Logistic Regression	20	88.97	85.66	92.53	92.53	[3,47,50,62,79,80,88]

As the table shows, the Decision Tree classifier is the one who has performed best. Moreover, as shown in the following diagram, the Precision metric value increases as its Recall metric value also increases.



It's also been clear that there were some articles within the datasets difficult to classify for almost every classifier (*ham misclassified as spam in the test set*):

- Multinomial Naive Bayes (20%) → [**3**,47,**50**,**62**,**79**,80,88]
- Multinomial Naive Bayes (40%) → [**3**,47,**50**,**62**,**79**,80,88,99,108,184]
- Linear Support Vector (20%) → [**3**,**50**,**62**,**79**]
- Linear Support Vector (40%) → [**3**,**50**,**62**,**79**,99,108,184]
- Decision Tree (20%) → [88]
- Decision Tree (40%) → [**3**,19,**50**,**62**,**79**,99,108,184]
- Logistic Regression (20%) → [**3**,47,**50**,**62**,**79**,80,88]
- Logistic Regression (40%) → [**3**,47,**50**,**62**,**79**,80,88,99,108,184]
- Stochastic Gradient Descent (20%) → [**3**,**50**,**62**,**79**]
- Stochastic Gradient Descent (40%) → [**3**,**50**,**62**,**79**,108,184]

A deeper study on the Multinomial Naive Bayes was also conducted: it was observed how the metrics moved after changing the classifier's parameters, especially when modifying the "fit_prior" value from "True" to "False". This parameter representes whether to learn class prior probabilities or not; when set to "False", a uniform prior is applied.

	TRAIN / TEST set size %	TRAIN / TEST F1 %	TRAIN / TEST Precision %	TRAIN / TEST Recall %	TRAIN / TEST Accuracy %	TRAIN / TEST Misclassified articles
Multinomial Naive Bayes (fit_prior=True)	80 / 20	94.79 / 88.97	93.13 / 85.66	96.51 / 92.55	96.51 / 92.55	[5,14,90,118,156,183,225,285, 302,323,326,334, 363] / [3,47,50,62,79,80,88]
Multinomial Naive Bayes (fit_prior=True)	60 / 40	94.66 / 92.05	92.66 / 89.95	96.42 / 94.65	96.42 / 94.65	[25,63,90,132,192,209,230,233, 241,270] / [3,47,50,62,79,80,88,99,108,184]
Multinomial Naive Bayes (fit_prior=False)	80 / 20	98.15 / 96.4	98.41 / 96.91	98.39 / 96.81	98.39 / 96.81	[14,118,183,285,302,326] / [50,79,80]
Multinomial Naive Bayes (fit_prior=False)	60 / 40	97.92 / 96.9	98.24 / 97.4	98.21 / 97.33	98.21 / 97.33	[25,90,192,209,233] / [50,79,80,108,184]

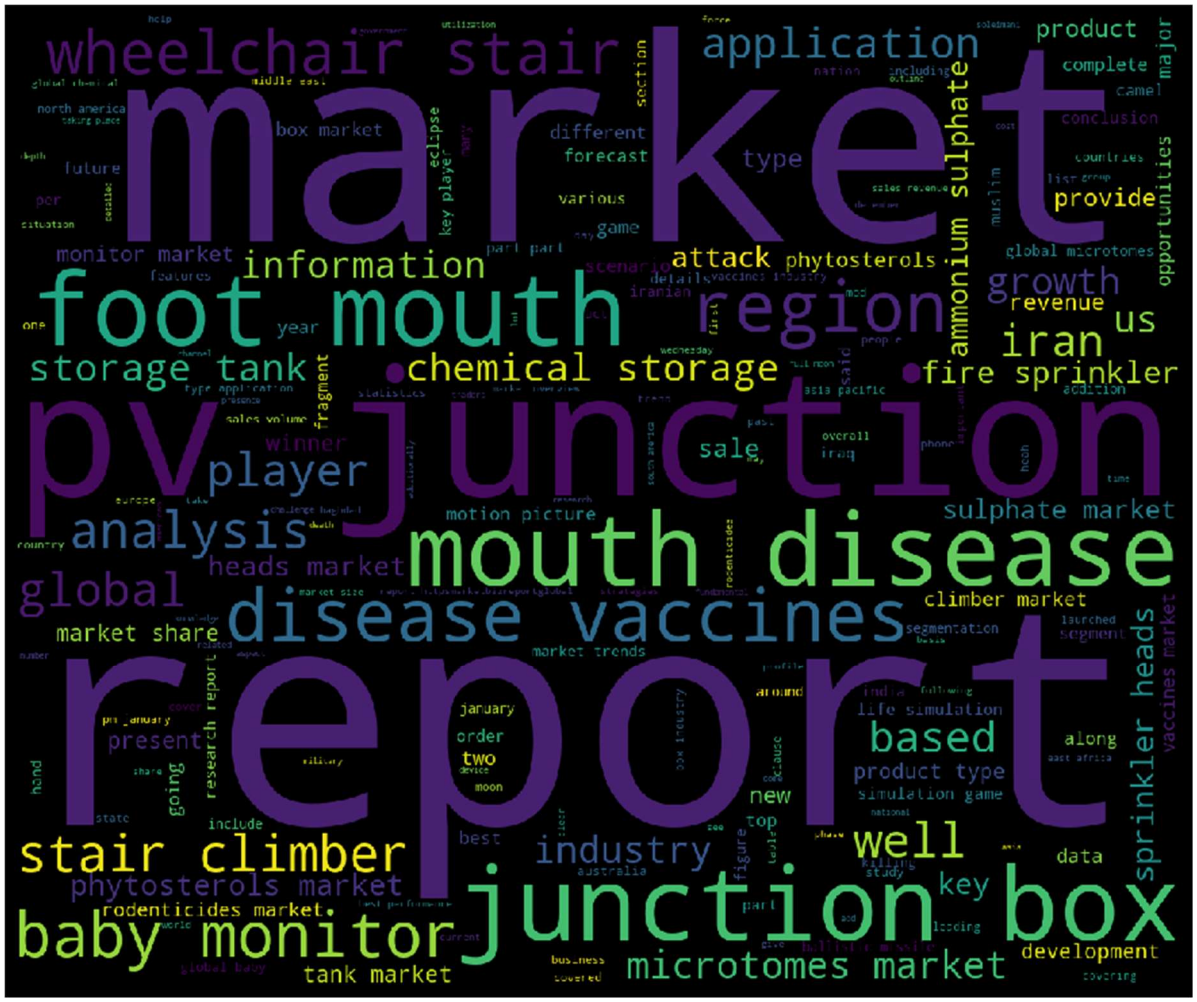
Code:

- https://github.com/Tireswind/SII-ML-project/blob/master/DecisionTree.ipynb
- https://github.com/Tireswind/SII-ML-project/blob/master/LinearSVC.ipynb
- https://github.com/Tireswind/SII-ML-project/blob/master/LogisticRegression.ipynb
- https://github.com/Tireswind/SII-ML-project/blob/master/Metrics_and_results.ipynb
- https://github.com/Tireswind/SII-ML-project/blob/master/Multinomial_Naive_Bayes.ipynb
- https://github.com/Tireswind/SII-ML-project/blob/master/Multinomial_Naive_Bayes_different_parameters.ipynb
- https://github.com/Tireswind/SII-ML-project/blob/master/StochasticGradientDescent.ipynb

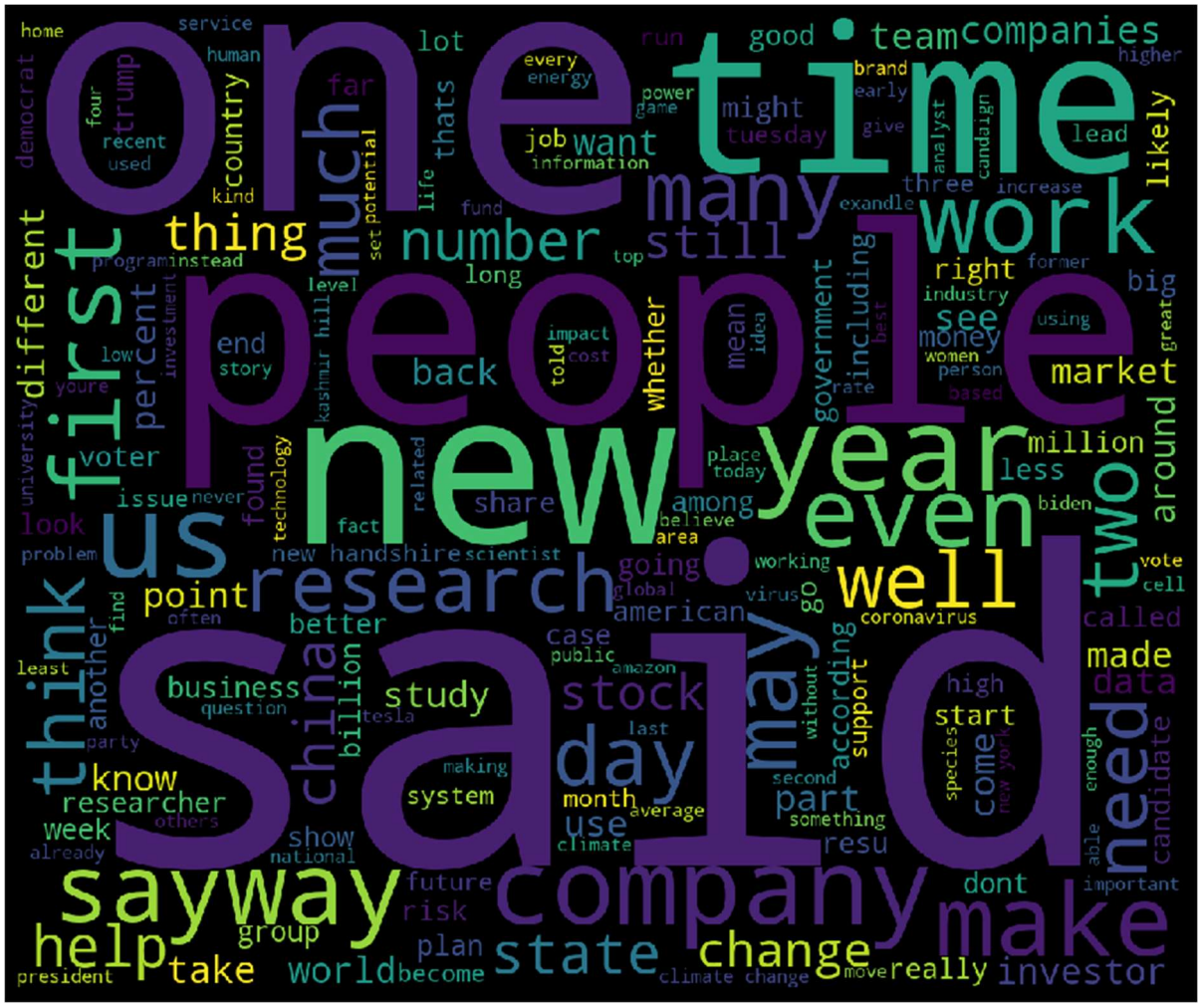
WordCloud

The final output was achieved using the WordCloud library. Here, the fifty most common words for both Ham and Spam news were displayed.

Spam:



Ham:



Confronting both images, what came to be observed is that one of the most common words for Ham news was “said”: basically, actual news cite the sources of their text. This does not happen for Spam news, where a more vague approach to the text is experienced (*e.g.: the word “report” without any additional pieces of information*).

Conclusions

This project has focused its attention on the detection phase of fake news, where standard machine learning approaches and classifiers were used to understand how the frequency of different words allows us to tell a good news from a fake one.

The dataset was built in a completly randomic way, with recent news, so to have the most various set of treated topics.

Further and deeper training of the model will of course improve the result of the involved evaluation metrics.

References

"Fake News Detection on Social Media: A Data Mining Perspective" - Kai Shuy, Amy Slivaz, Suhang Wangy, Jiliang Tang, and Huan Liuy.

"A Benchmark Study on Machine Learning Methods for Fake News Detection" - Junaed Younus Khan, Md. Tawkat Islam Khondaker, Anindya Iqbal, Sadia Afroz.

"FAKEDETECTOR: Effective Fake News Detection with Deep Diffusive Neural Network" - Jiawei Zhang, Bowen Dong, Philip S. Yu.

https://github.com/eChanClarityInsights - Fake-News-Classifier.

Various Python libraries' online documentation.