|  |
| --- |
| Master in Data Science – Final Project |
| Latest News Classifier |
| An end-to-end Machine Learning Project |

|  |
| --- |
| Miguel Fernández Zafra  17/01/2019 |

Table of contents

[1. Introduction 3](#_Toc535267561)

[2. Input data 4](#_Toc535267562)

[3. Methodology 5](#_Toc535267563)

[3.1. Creation of the initial dataset 5](#_Toc535267564)

[3.2. Exploratory Data Analysis 5](#_Toc535267565)

[3.3. Feature Engineering 8](#_Toc535267566)

[3.3.1. Text representation 8](#_Toc535267567)

[3.3.2. Text cleaning 10](#_Toc535267568)

[3.3.3. Label coding 10](#_Toc535267569)

[3.3.4. Train – test split 10](#_Toc535267570)

[3.4. Predictive Models 10](#_Toc535267571)

[3.5. Web Scraping 10](#_Toc535267572)

[3.6. Web Application 10](#_Toc535267573)

[4. Results 10](#_Toc535267574)

[5. Annexes 10](#_Toc535267575)

[5.1. Annex 1: Installation 10](#_Toc535267576)

[5.2. Annex 2: Web App Deployment with Heroku 10](#_Toc535267577)

[5.3. Annex 3: Dashboard instructions 10](#_Toc535267578)

# Introduction

This project is intended to be a walkthrough of the development of a machine learning project used to create an **application** that can be used to obtain some benefit to a set of users.

Concretely, we have created a real-time web application that gathers data from several newspapers and shows a summary of the different topics that are being treated in the news articles.

This is achieved with a supervised machine learning **classification model** that is able to predict the category of a given news article, a **web scraping method** that gets the latest news from the newspapers, and an **interactive web application** that shows the obtained results to the user.

This can be seen as a **text classification** problem. Text classification is one of the widely used natural language processing (NLP) applications in different business problems.

This project is intended to cover the **full process** of creating a service or application based on machine learning. Not only the process of getting features from text data and fitting them into a model is covered, but also the following part of the workflow is: deploying the model to a real-time application that gathers new live data, makes a prediction and returns some insights.

The motivation behind developing this project is the following: as a learning data scientist who has been working with data science tools and machine learning models for not a really long time, I’ve found out that many articles in the internet, books or literature in general strongly focus on the modeling part. That is, we are given a certain dataset (with the labels already assigned if it is a supervised learning problem), try several models and obtain a performance metric. And the process ends there.

But in real life problems, I think that finding the right model with the right hyperparameters is only the **beginning** of the task. What will happen when we deploy the model? How will it respond to new data? Will this data look the same as the training dataset? Perhaps, will there be some information (scaling or feature-related information) that we will need? Will it be available?

Therefore, we will be covering the full process: getting the raw data and parsing it, creating the features, training the models and choosing the best one, and using it to predict new articles and show a summary.

# Input data

The dataset used in this project is the BBC News Raw Dataset. It can be downloaded from:

[*http://mlg.ucd.ie/datasets/bbc.html*](http://mlg.ucd.ie/datasets/bbc.html)

It consists of 2.225 documents from the BBC news website corresponding to stories in five topical areas from 2004 to 2005. These areas are:

* Business
* Entertainment
* Politics
* Sport
* Tech

In the same webpage we can find another dataset (*BBCSport*), which consists of 737 documents from the BBC Sport website. However, in this project it hasn't been used.

In addition, a pre-processed dataset is also provided. This pre-processing includes stemming, stop-word removal and low term frequency filtering[[1]](#footnote-2). Again, it has not been used. The raw dataset has been used instead.

The download file contains five folders (one for each category). Each folder has a single *.txt* file for every news article. These files include the news articles body in raw text.

**References:**

|  |
| --- |
| The downloaded files are placed in 00. Raw dataset\BBC\bbc-fulltext\bbc. There, we can see the five folders, each containing the .txt files. |

# Methodology

## Creation of the initial dataset

The aim of this step is to get a dataset with the following structure:

|  |  |  |
| --- | --- | --- |
| File Name | Content | Category |
| Document 1 Name | Document 1 Content | Document 1 Category |
| … | … | … |

That is, every row will represent a single document and the columns will store its name, content and category.

We have created this dataset with a R script, because the package *readtext* simplifies a lot this procedure.

For further detail please see the references:

**References:**

|  |
| --- |
| * *01. Dataset Creation.R*: the R script used. * *01. Dataset Creation.Rproj*: R project. * *01. Dataset Creation.Rmd*: for documental purposes, a Markdown version with comments has been created. * *01.\_Dataset\_Creation.md*: knit markdown document. |

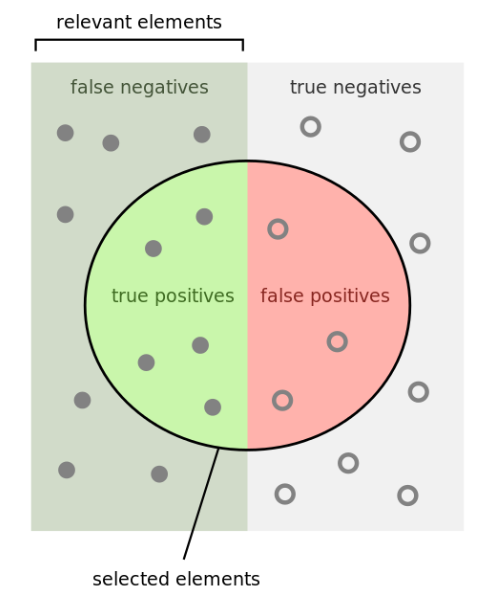
## Exploratory Data Analysis

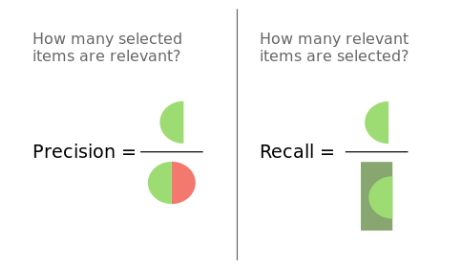
It is a common practice to carry out an exploratory data analysis in order to gain some insights from the data. However, up to this point, we don’t have any features that define our data. We will see how to create features from text in the next section (*3.3 Feature Engineering*), but, because of the way these features are constructed, we would not expect any valuable insights from analyzing them. For this reason, we have only performed a shallow analysis.

One of our main concerns when developing a classification model is whether the different classes are balanced. This means that the dataset contains an approximately equal portion of each class.

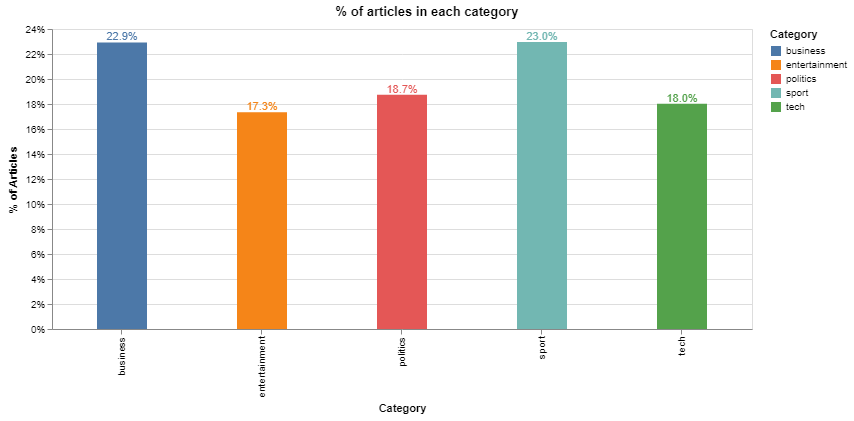
For example, if we had two classes and a 95% of observations belonging to one of them, a dumb classifier which always output the majority class would have a 95% accuracy, although it would fail all the predictions of the minority class.

There are several ways of dealing with imbalanced datasets. One first approach is to undersample the majority class and oversample the minority one, so as to obtain a more balanced dataset. Other approach can be using other error metrics beyond accuracy such as the precision, the recall or the F1-score. The following figure may provide some clarification on these concepts[[2]](#footnote-3):



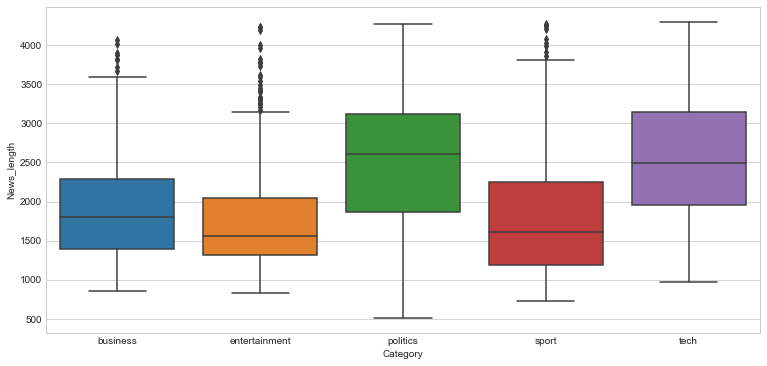


Looking at our data, we can get the % of observations belonging to each class:



We can see that the classes are approximately balanced, so we won’t perform any undersampling or oversampling method. However, we will anyway use precision and recall to evaluate model performance.

Another variable of interest can be the length[[3]](#footnote-4) of the news articles. We can obtain the length distribution across categories:



We can see that politics and tech articles tend to be longer, but not in a significant way. In addition, we will see in the next section that the length of the articles is taken into account by the method we use to create the features. So this should not matter too much to us.

Further detail of the exploratory data analysis can be found in:

**References:**

|  |
| --- |
| The exploratory data analysis process has been developed in *02. Exploratory Data Analysis.ipynb.* In this step, we also save the dataset to a pickle object so as to use it in further sections. |

## Feature Engineering

Feature engineering is an essential part of building any intelligent system. As Andrew Ng says:

*“Coming up with features is difficult, time-consuming, requires expert knowledge. ‘Applied machine learning’ is basically feature engineering.”*

Feature engineering is the process of transforming data into features to act as inputs for machine learning models such that good quality features help in improving the model performance.

When dealing with text data, there are several ways of obtaining features that represent the data. We will cover some of the most common methods[[4]](#footnote-5) and then choose the most suitable for our needs.

### Text representation

Recall that, in order to represent our text, every row of the dataset will be a single document of the corpus. The columns (features) will be different depending of which feature creation method we choose:

* **Word Count Vectors**

With this method, every column is a term from the corpus, and every cell represents the frequency count of each term in each document.

* **TF–IDF Vectors**

TF-IDF is a score that represents the relative importance of a term in the document and the entire corpus. *TF* stands for *Term Frequency*, and *IDF* stands for *Inverse Document Frequency*:

Being:

* : term (i.e. a word in a document)
* : document
* : term frequency (i.e. how many times the term appears in the document )
* : number of documents in the corpus
* : number of documents in the corpus containing the term

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 contain the word, which helps to adjust for the fact that some words appear more frequently in general**.**

It also takes into account the fact that some documents may be larger than others by normalizing the term (expressing instead relative term frequencies).

These two methods (Word Count Vectors and TFIDF Vectors are often named Bag of Words methods, since the order of the words in a sentence is ignored. The following methods are more advanced as they somehow preserve the order of the words and their lexical considerations.

* **Word Embeddings**

The position of a word within the vector space is learned from text and is based on the words that surround the word when it is used. Word embeddings can be used with pre-trained models applying transfer learning.

* **Text based or NLP based features**

We can manually create any feature that we think may be of importance when discerning between categories (i.e. word density, number of characters or words, etc…).

We can also use NLP based features using Part of Speech models, which can tell us, for example, if a word is a noun or a verb, and then use the frequency distribution of the PoS tags.

* **Topic Models**

Methods such as Latent Dirichlet Allocation try to represent every topic by a probabilistic distribution over words, in what is known as topic modeling.

We have chosen *TF-IDF* vectors to represent the documents in our corpus. This election is motivated by the following points:

* TF-IDF is a simple model that yields great results in this particular domain, as we will see in section 3.4.
* TF-IDF features creation is a fast process, which will lead us to shorter waiting time for the user when using the web application.
* We can tune the feature creation process (see next paragraph) to avoid issues like overfitting.

When creating the features with this method, we can choose some parameters:

* N-gram[[5]](#footnote-6) range: we are able to consider unigrams, bigrams, trigrams…
* Maximum/Minimum Document Frequency: when building the vocabulary, we can ignore terms that have a document frequency strictly higher/lower than the given threshold
* Maximum features: we can choose the top N features ordered by term frequency across the corpus.

We have chosen the following parameters:

|  |  |
| --- | --- |
| Parameter | Value |
| N-gram range | (1,2) |
| Maximum DF | 100% |
| Minimum DF | 10 |
| Maximum features | 300 |

We expect that bigrams help to improve our model performance by taking into consideration words that tend to appear together in the documents. We have chosen a value of Minimum DF equal to 10 to get rid of extremely rare words that don’t appear in more than 10 documents, and a Maximum DF equal to 100% to not ignore any other words. The election of 300 as maximum number of features has been made because we want to avoid possible overfitting, often arising from a large number of features compared to the number of training observations.

As we will see in the next sections, these values lead us to really high accuracy values, so we will stick to them. However, These parameters could be tuned in order to train better models.

### Text cleaning

Before creating any feature from the raw text, we must perform a cleaning process to ensure no distortions are introduced to the model. We have followed these steps:

* **Special character cleaning:** special characters such as “*\n*” double quotes must be removed from the text since we aren’t expecting any predicting power from them.
* **Upcase/downcase:** we would expect, for example, “Book” and “book” to be the same word and have the same predicting power. For that reason we have downcased every word.
* **Punctuation signs:** characters such as “?”, “!”, “;” have been removed.
* **Possessive pronouns:** in addition, we would expect that “Trump” and “Trump’s” had the same predicting power.
* **Stemming or Lemmatization:** stemming is the process of reducing derived words to their root. Lemmatization is the process of reducing a word to its lemma. The difference between both methods is that lemmatization provides existing words, whereas stemming provides the root, which may not be an existing word. We have used a Lemmatizer based in WordNet.
* **Stop words:** words such as “what” or “the” won’t have any predicting power since they will presumably be common to all the documents. For this reason, they may represent noise that can be eliminated. We have downloaded a list of English stop words from the *nltk* package and then deleted them from the corpus.

### Label coding

Machine learning models require numeric features and labels to provide a prediction. For this reason we must create a dictionary to map each label to a numerical ID. We have created this mapping scheme:

|  |  |
| --- | --- |
| Category Name | Category Code |
| Business | 0 |
| Entertainment | 1 |
| Politics | 2 |
| Sport | 3 |
| Tech | 4 |

### 

### Train – test split

We need to set apart a test set in order to prove the quality of our models. We have chosen a random split with 85% of the observations composing the training test and 15% of the observations composing the test set.

**References:**

|  |
| --- |
| The whole feature engineering process has been developed in *03. Feature Engineering.ipynb.* See the notebook for further detail. |

## Predictive Models

mñmklmklcds

## Web Scraping

## Web Application

# Results

# Annexes

## Annex 1: Installation

## Annex 2: Web App Deployment with Heroku

## Annex 3: Dashboard instructions

1. These concepts will be covered in *3.1. Predictive Modelling* [↑](#footnote-ref-2)
2. Source: Wikipedia [↑](#footnote-ref-3)
3. The length of an article has been defined as the number of characters in it [↑](#footnote-ref-4)
4. Source: *https://www.analyticsvidhya.com/blog/2018/04/a-comprehensive-guide-to-understand-and-implement-text-classification-in-python/* [↑](#footnote-ref-5)
5. An N-gram is an element consisting of N tokens (i.e. words). [↑](#footnote-ref-6)