An end to end Machine Learning project

Text Classification in Python

Learn to build a text classification model in Python

Miguel Fernández Zafra Follow
Jun 16 · 17 min read ★

This article is the first of a series in which I will cover the **whole process** of developing a machine learning project.

In this article we focus on training a supervised learning **text classification** model in Python.

The motivation behind writing these articles is the following: as a learning data scientist who has been working with data science tools and machine learning models for a fair amount of 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? Will the user allow and understand the uncertainty associated with the results? We have to ask ourselves these questions if we want to succeed at bringing a machine learning-based service to our final users.

For this reason, I have developed a project that covers this full process of creating a ML-based service: getting the raw data and parsing it, creating the features, training different models and choosing the best one, getting new data to feed the model and showing useful insights to the final user.

The project involves the creation of a real-time web application that gathers data from several newspapers and shows a summary of the different topics that are being discussed 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.

These article is aimed to people that already have some understanding of the basic machine learning concepts (i.e. know what cross-validation is and when to use it, know the difference between Logistic and Linear Regression, etc…). However, I will briefly explain the different concepts involved in the project.

The github repo can be found here. It includes all the code and a complete report. I will not include the code in this post because it would be too large, but I will provide a link wherever it is needed.

I will divide the process in three different posts:

- Classification model training (this post)
- News articles web scraping (will be published soon)
- App creation and deployment (will be published soon)

This post covers the first part: **classification model training**. We'll cover it in the following steps:

1. Problem definition and solution approach
2. Input data
3. Creation of the initial dataset

4. Exploratory Data Analysis
5. Feature Engineering
6. Predictive Models

1. Problem definition and solution approach

As we have said, we are talking about a supervised learning problem. This means we need a labeled dataset so the algorithms can learn the patterns and correlations in the data. We fortunately have one available, but in real life problems this is a **critical** step since we normally have to do the task manually. Because, if we are able to automate the task of labeling some data points, then why would we need a classification model?

2. Input data

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

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

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.

3. 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
Document 2 Name	Document 2 Content	Document 2 Category
...	...	...

We have created this dataset with an R script, because the package *readtext* simplifies a lot this procedure. The script can be found here.

4. Exploratory Data Analysis

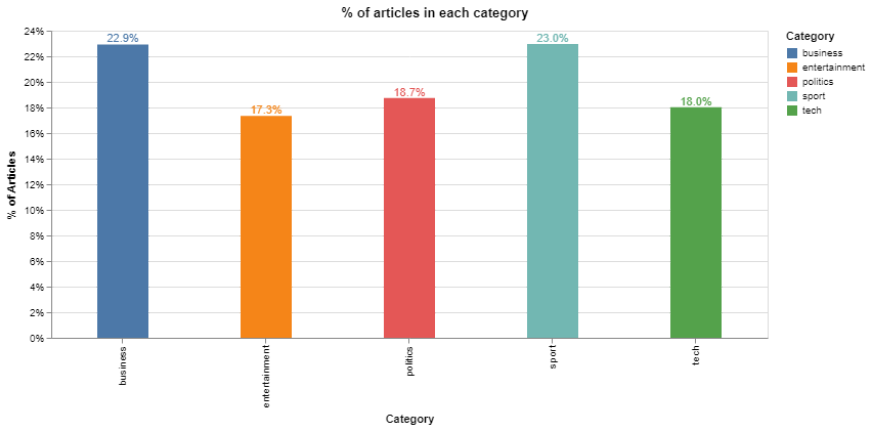
It is a common practice to carry out an exploratory data analysis in order to gain some insights from the data. However, up to this point, we don't have any features that define our data. We will see how to create features from text in the next section (*5. Feature Engineering*), but, because of the 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 95% accuracy, although it would fail all the predictions of the minority class.

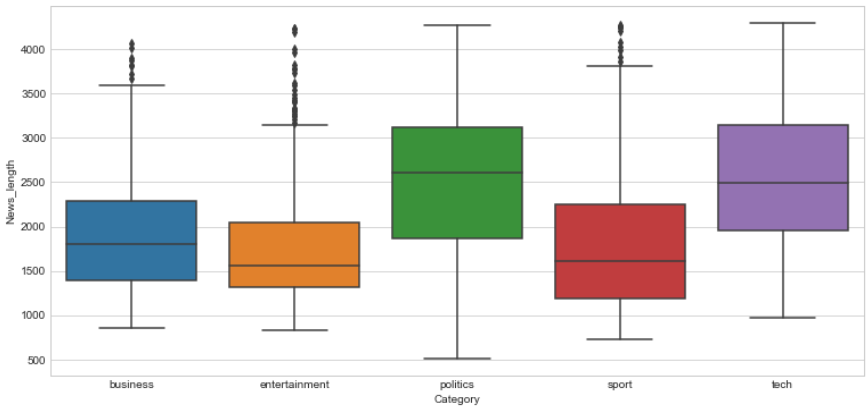
There are several ways of dealing with imbalanced datasets. One first approach is to **undersample** the majority class and **oversample** the minority one, so as to obtain a more balanced dataset. Other approach can be using other error metrics beyond accuracy such as the **precision**, the **recall** or the **F1-score**. We'll talk more about these metrics later.

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 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 and corrected by the method we use to create the features. So this should not matter too much to us.

The EDA notebook can be found here.

5. 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 and then choose the most suitable for our needs.

5.1. 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*:

$$TFIDF(t,d) = TF(t,d) \times \log\left(\frac{N}{DF(t)}\right)$$

Being:

- t: term (i.e. a word in a document)
- d: document
- $TF(t)$: term frequency (i.e. how many times the term t appears in the document d)
- N: number of documents in the corpus
- $DF(t)$: number of documents in the corpus containing the term t

The *TF-IDF* 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 *TF* term (expressing instead relative term frequencies).

These two methods (Word Count Vectors and *TF-IDF* 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 later.
- *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 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	1
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.

There is one important consideration that needs to be mentioned. Recall that the calculation of *TF-IDF* scores needs the presence of a **corpus** of documents to compute the Inverse Document Frequency term. For this reason, if we wanted to predict a single news article at a time (for example once the model is deployed), we would need to define that corpus.

This corpus is the set of training documents. Consequently, when obtaining *TF-IDF* features from a new article, only the features that existed in the training corpus will be created for this new article.

It is straight to conclude that the **more similar** the training corpus is to the news that we are going to be scraping when the model is deployed, the more accuracy we will presumably get.

5.2. 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 main 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.

There is one important consideration that must be made at this point. We should take into account possible distortions that are **not only** present in the training test, but also in the news articles that will be scraped when running the web application.

5.3. 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

5.4. Train—test split

We need to set apart a test set in order to prove the quality of our models when predicting unseen data. We have chosen a random split with 85% of the observations composing the training test and 15% of the observations composing the test set. We will perform the hyperparameter tuning process with cross validation in the training data, fit the final model to it and then evaluate it with **totally unseen** data so as to obtain an evaluation metric as less biased as possible.

The complete and detailed feature engineering code can be found here.

6. Predictive Models

6.1. Hyperparameter tuning methodology and models

We have tested several machine learning models to figure out which one may fit better to the data and properly capture the relationships across the points and their labels. We have only used classic machine learning models instead of deep learning models because of the insufficient amount of data we have, which would probably lead to overfit models that don't generalize well on unseen data.

We have tried the following models:

- Random Forest
- Support Vector Machine

- K Nearest Neighbors
- Multinomial Naïve Bayes
- Multinomial Logistic Regression
- Gradient Boosting

Each one of them has multiple hyperparameters that also need to be tuned. We have followed the following methodology when defining the best set of hyperparameters for each model:

Firstly, we have decided **which hyperparameters** we want to tune for each model, taking into account the ones that may have more influence in the model behavior, and considering that a high number of parameters would require a lot of computational time.

Then, we have defined a grid of possible values and performed a **Randomized Search** using 3-Fold Cross Validation (with 50 iterations). Finally, once we get the model with the best hyperparameters, we have performed a **Grid Search** using 3-Fold Cross Validation centered in those values in order to exhaustively search in the hyperparameter space for the best performing combination.

We have followed this methodology because with the randomized search we can cover a much wider range of values for each hyperparameter without incurring in really high execution time. Once we narrow down the range for each one, we know where to concentrate our search and explicitly specify every combination of settings to try.

The reason behind choosing $K = 3$ as the number of folds and 50 iterations in the randomized search comes from the trade-off between shorter execution time or testing a high number of combinations. When choosing the best model in the process, we have chosen the accuracy as the evaluation metric.

6.2. Performance Measurement

After performing the hyperparameter tuning process with the training data via cross validation and fitting the model to this training data, we need to evaluate its performance on totally unseen data (the test set). When dealing with classification problems, there are several metrics that can be used to gain insights on how the model is performing. Some of them are:

- Accuracy: the accuracy metric measures the ratio of correct predictions over the total number of instances evaluated.
- Precision: precision is used to measure the positive patterns that are correctly predicted from the total predicted patterns in a positive class.
- Recall: recall is used to measure the fraction of positive patterns that are correctly classified
- F1-Score: this metric represents the harmonic mean between recall and precision values
- Area Under the ROC Curve (AUC): this is a performance measurement for classification problem at various thresholds settings. ROC is a probability curve and AUC represents degree or measure of separability. It tells how much a model is capable of distinguishing between classes.

These metrics are highly extended an widely used in binary classification. However, when dealing with multiclass classification they become more complex to compute and less interpretable. In addition, in this particular application, we just want documents to be correctly predicted. The costs of false positives or false negatives are the same to us. For this reason, it does not matter to us whether our classifier is more specific or more sensitive, as long as it classifies correctly as much documents as possible. Therefore, we have studied the **accuracy** when comparing models and when choosing the best hyperparameters. In the first case, we have calculated the accuracy on both training and test sets so as to detect overfit models. However, we have also obtained the confusion matrix and the classification report (which computes precision, recall and F1-score for all the classes) for every model, so we could further interpret their behavior.

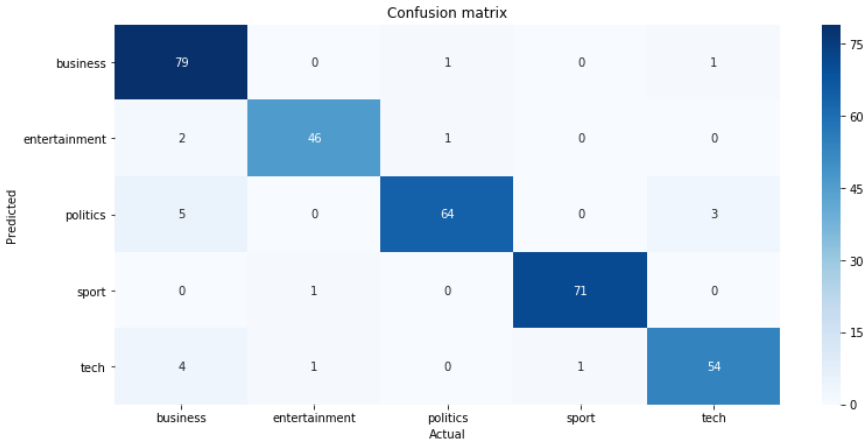
6.3. Best Model Selection

Below we show a summary of the different models and their evaluation metrics:

Model	Training Set Accuracy	Test Set Accuracy
Gradient Boosting	100%	94%
Multinomial Logistic Regression	98%	94%
SVM	95%	94%
Multinomial Naïve Bayes	95%	93%
K Nearest Neighbors	95%	92%
Random Forest	100%	92%

Overall, we obtain really good accuracy values for every model. We can observe that the Gradient Boosting, Logistic Regression and Random Forest models seem to be overfit since they have an extremely high training set accuracy but a lower test set accuracy, so we'll discard them.

We will choose the SVM classifier above the remaining models because it has the highest test set accuracy, which is really near to the training set accuracy. The **confusion matrix** and the **classification report** of the SVM model are the following:



Classification report

	precision	recall	f1-score	support
0	0.88	0.98	0.92	81
1	0.96	0.94	0.95	49
2	0.97	0.89	0.93	72
3	0.99	0.99	0.99	72
4	0.93	0.90	0.92	60
avg / total	0.94	0.94	0.94	334

6.4. Model Interpretation

At this point we have selected the SVM as our preferred model to do the predictions. Now, we will study its behavior by analyzing misclassified articles, in order to get some **insights** on the way the model is working and, if necessary, think of new features to add to the model. Recall that, although the hyperparameter tuning is an important process, the most critic process when developing a machine learning project is being able to extract good features from the data.

Let's show an example of a misclassified article. Its actual category is politics, although the model predicted tech.

> *MPs issued with Blackberry threat*

> *MPs will be thrown out of the Commons if they use Blackberries in the chamber Speaker Michael Martin has ruled.*

> *The Â£200 handheld computers can be used as a phone, pager or to send e-mails. The devices gained new prominence this week after Alastair Campbell used his to accidentally send an expletive-laden message to a Newsnight journalist. Mr Martin revealed some MPs had been using their Blackberries during debates and he also cautioned members against using hidden earpieces.*

> *The use of electronic devices in the Commons chamber has long been frowned on. The sound of a mobile phone or a pager can result in a strong rebuke from either the Speaker or his deputies. The Speaker chairs debates in the Commons and is charged with ensuring order in the chamber and enforcing rules and conventions of the House. He or she is always an MP chosen by colleagues who, once nominated, gives up all party political allegiances.*

This article talks about the prohibition of Blackberry mobiles in the Commons chamber. It involves both politics and tech, so the misclassification makes sense.

6.5. Dimensionality reduction plots

Dimension Reduction refers to the process of converting a set of data having vast dimensions into data with lesser dimensions ensuring that it conveys similar information concisely.

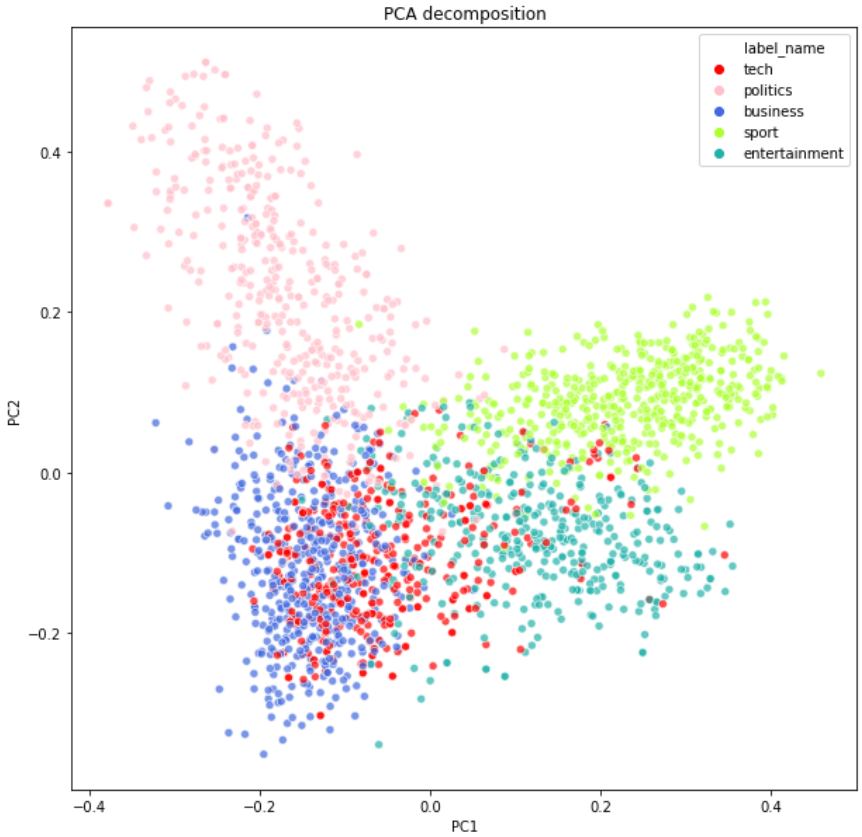
There are **many applications** of dimensionality reduction techniques in machine learning. One of them is visualization. By reducing the dimensional space to 2 or 3 dimensions that contain a great part of the information, we can plot our data points and be able to recognize some patterns as humans.

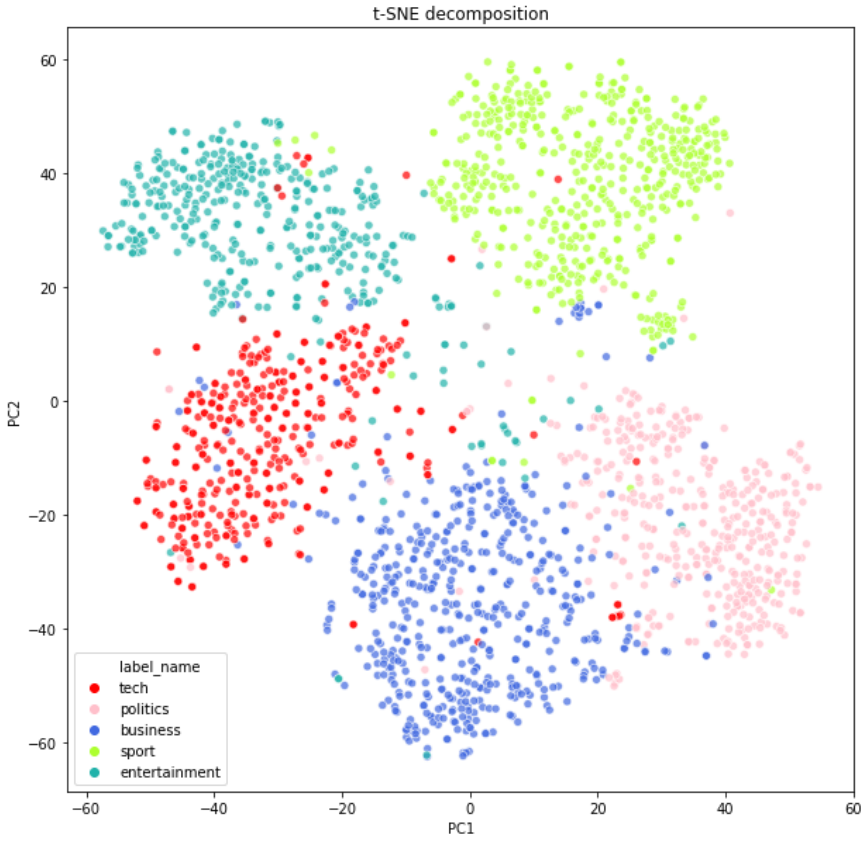
We have used two different techniques for dimensionality reduction:

- **Principal Component Analysis**: this technique relies on the obtention of the eigenvalues and eigenvectors of the data matrix and tries to provide a minimum number of variables that keep the maximum amount of variance.

- **t-SNE**: the t-distributed Stochastic Neighbor Embedding is a probabilistic technique particularly well suited for the visualization of high-dimensional datasets. It minimizes the divergence between two distributions: a distribution that measures pairwise similarities of the input objects and a distribution that measures pairwise similarities of the corresponding low-dimensional points in the embedding.

Let's plot the results:





We can see that using the t-SNE technique makes it easier to distinguish the different classes. Although we have only used dimensionality reduction techniques for plotting purposes, we could have used them to shrink the number of features to feed our models. This approach is particularly useful in text classification problems due to the commonly large number of features.

6.6. Predicted Conditional Probabilities

We have to make an additional consideration before stepping into the web scraping process. The training dataset has articles labeled as Business, Entertainment, Sports, Tech and Politics. But we could think of news articles that don't fit into any of them (i.e. a weather news article). Since we have developed a supervised learning model, these kind of articles would be wrongly classified into one of the 5 classes.

In addition, since our training dataset is dated of 2004–2005, there may be a lot of new concepts (for example, technological ones) that will appear when scraping the latest articles, but won't be present in the training data. Again, we expect poor predicting power in these cases.

A lot of classification models provide not only the class to which some data point belongs. They can also provide the conditional probability of

belonging to the class *C*.

When we have an article that clearly talks, for example, about politics, we expect that the conditional probability of belonging to the Politics class is very high, and the other 4 conditional probabilities should be very low.

But when we have an article that talks about the weather, we expect all the conditional probability vector's values to be equally low.

Therefore, we can specify a threshold with this idea: if the highest conditional probability is lower than the threshold, we will provide no predicted label for the article. If it is higher, we will assign the corresponding label.

After a brief study exploring different articles that may not belong to any of the 5 categories, we have fixed that threshold at 65%.

For further detail on all the steps of the model training process, please visit this link.

At this point, we have trained a model that will be able to classify news articles that we feed into it. We are a step closer to building our application!