LATEST NEWS CLASSIFIER

An end-to-end Machine Learning Project

Badal Kumar

Table of Contents

1.Introduction

1.1Problem Definition

This project is intended to be a walkthrough on 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 different models and choosing the best one, and using it to predict new web-scraped articles and show a web summary.

1.2 Hardware Specification

RAM :-	Minimum 4GB
Hard Disk :-	100 GB
Processor :-	intel core i5
GPU :-	2GB

1.3 Software Specification

Python Version 3.7
Pandas
Scikit-learn

Natural Language Processing(NLP)
Machine Learning Algorithms
MS Word 97 or later
Web Browser: Microsoft Internet Explorer, Mozilla, Google Chrome or later
Operating System: Windows XP / Windows7/ Windows Vista /Window 10

2. 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

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:

o Business
o Entertainment
o Politics
o Sport
o 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 filtering1. 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.

3. Methodology

As a brief summary, at this point we'll cover the following aspects: creation of the initial dataset, exploratory data analysis, feature engineering, predictive models training, web scraping and the creation of the final web application.

3.1. 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 an Python script, because the package *readtext* simplifies a lot this procedure.

3.2. Exploratory Data Analysis

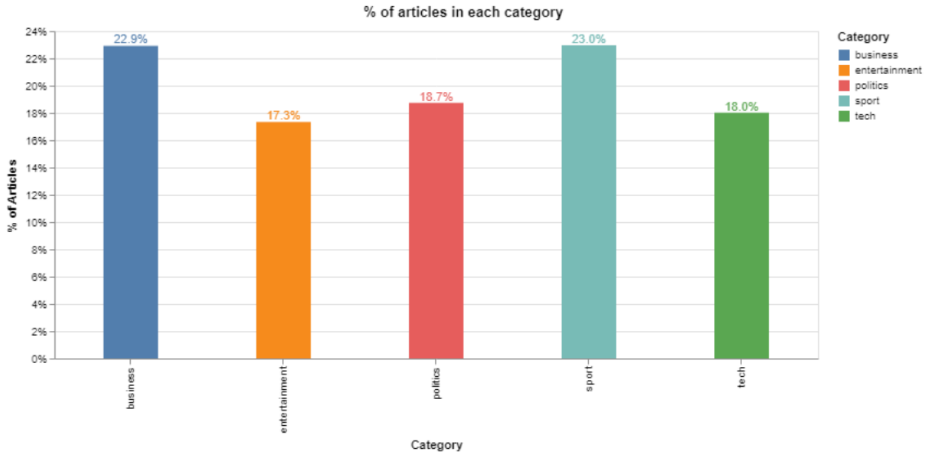
It is a common practice to carry out an exploratory data analysis in order to gain some insights from the data. However, up to this point, we don't have any features that define our data. We will see how to create features from text in the next section (*3.3 Feature Engineering*), but, because of the 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 95% accuracy, although it would fail all the predictions of the minority class.

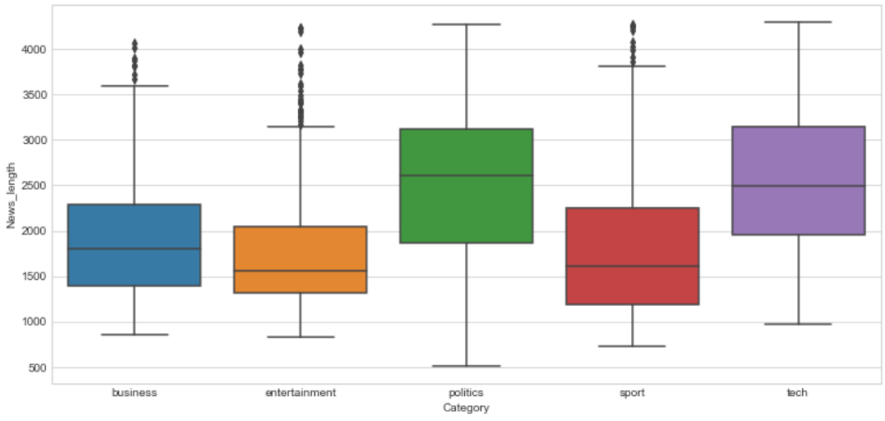
There are several ways of dealing with imbalanced datasets. One first approach is to undersample the majority class and oversample the minority one, so as to obtain a more balanced dataset. Other approach can be using other error metrics beyond accuracy such as the precision, the recall or the F1-score. For more detail on these metrics see *3.4.3. Performance Measurement*.

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[2] 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.

Flow Chart

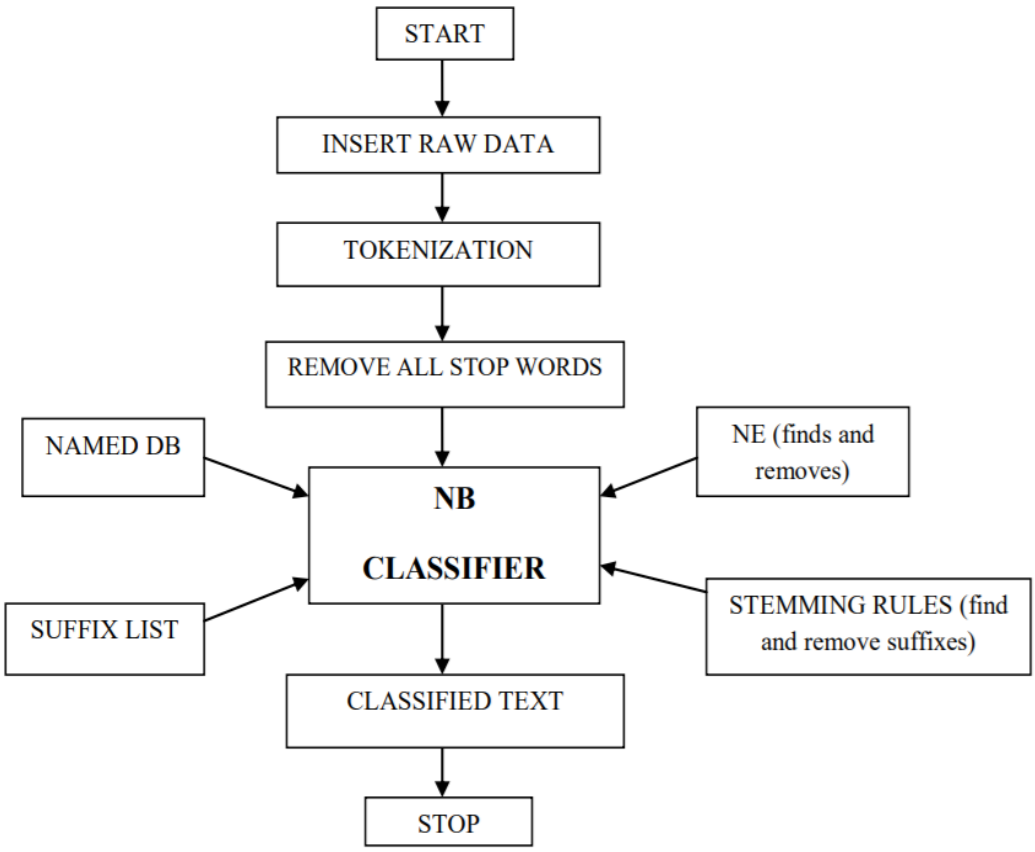


Figure 1: Flow Chart

3.3. 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 methods3 and then choose the most suitable for our needs.

3.3.1. 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:

o **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.

o **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*:

$$TFIDF(t,d) = TF(t,d) \times \log(N/DF(t))$$

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 *TFIDF* 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 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.

o 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.

o 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.

o 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:

o *TF-IDF* is a simple model that yields great results in this particular domain, as we will see in section *3.4. Predictive Models*.

o *TF-IDF* features creation is a fast process, which will lead us to shorter waiting time for the user when using the web application.

o 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:

o N-gram4 range: we are able to consider unigrams, bigrams, trigrams…

o Maximum/Minimum Document Frequency: when building the vocabulary, we can ignore terms that have a document frequency strictly higher/lower than the given threshold

o 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)
Maximun 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.

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.

3.3.2. 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.

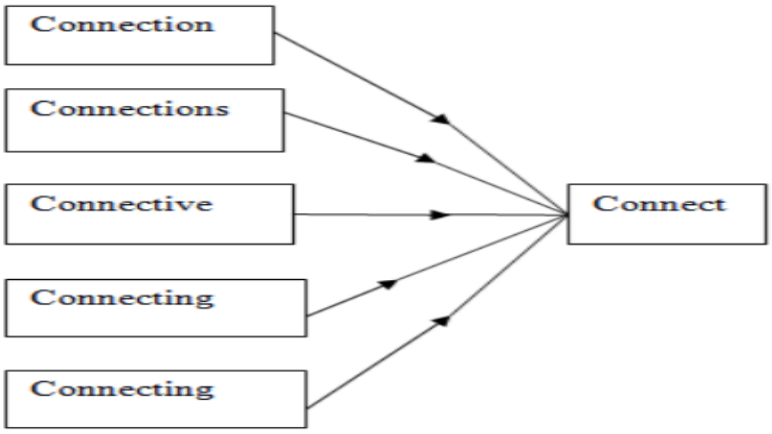


Fig: Stemming Process

- **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.

3.3.3. 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

3.3.4. 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 evaluat

3.4. Predictive Models

3.4.1. Hyperparameter Tuning Methodology

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 (see 3.4.4 for more details).

3.4.2. Machine Learning 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. For each model, we will provide a brief explanation of the logic behind them and the hyperparameters we have tuned according to the previous section's methodology.

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.

Random Forest

Random forest is a bagging technique and not a boosting technique. The trees in random forests are run in parallel. There is no interaction between these trees while building the trees.

It operates by constructing a multitude of decision trees at training time and outputting the class that is the mode of the classes (classification) or mean prediction (regression) of the individual trees.

A random forest is a meta-estimator (i.e. it combines the result of multiple predictions) which aggregates many decision trees, with some helpful modifications:

1. The number of features that can be split on at each node is limited to some percentage of the total (which is known as the hyperparameter). This ensures that the ensemble model does not rely too heavily on any individual feature, and makes fair use of all potentially predictive features.

2. Each tree draws a random sample from the original data set when generating its splits, adding a further element of randomness that prevents overfitting.

Random forests or random decision forests are an ensemble learning method (bagging) that operates by constructing a multitude of decision trees at training time and outputting the class that is the mode of the classes (in the case of classification). Random decision forests correct for decision trees' habit of overfitting to their training set.

The list of hyperparameters we have tuned is:

Hyperparameter	Brief Description
n_estimators	Number of trees in the forest.
max_features	Maximum number of features considered for splitting a node.
max_depth	Maximum number of levels in each decision tree.
min_samples_split	Minimum number of data points placed in a node before the node is split
min_samples_leaf	Minimum number of data points allowed in a leaf node.
bootstrap	Method for sampling data points (with or without replacement).

Support Vector Machine

Support Vector Machine is a supervised machine learning algorithm which is mostly used in classification problems. The classification is performed by finding the hyperplane that best differentiates the classes. When we have non-linear relationships in our data, we can modify the coordinate space with some kernel transformations to capture the relationships.

A Support Vector Machine (SVM) is a discriminative classifier formally defined by a separating hyperplane. In other words, given labeled training data (*supervised learning*), the algorithm outputs an optimal hyperplane which categorizes new examples. In two dimentional space this hyperplane is a line dividing a plane in two parts where in each class lay in either side.

The list of hyperparameters we have tuned is:

Hyperparameter	Brief Description
C	Penalty parameter C of the error term.
kernel	Specifies the kernel type to be used in the algorithm.
gamma	Kernel coefficient
degree	Degree of the polynomial kernel function

The multi-class classification is achieved with a “one-vs-all” scheme.

K Nearest Neighbors

K Nearest Neighbour is a simple algorithm that stores all the available cases and classifies the new data or case based on a similarity measure. It is mostly used to classifies a data point based on how its neighbours are classified.

The K nearest neighbors algorithm is a non-parametric method used for classification and regression. In both cases, the input consists of the K closest training examples in the feature space. In the case of classification, the output is the class membership of the point. The list of hyperparameters we have tuned is:

The list of hyperparameters we have tuned is:

Hyperparameter	Brief Description
K	Number of neighbors to use by default for queries.

In the case of the KNN algorithm, we have only performed a Grid Search Cross Validation process to get the best value of K in terms of accuracy.

Multinomial Naïve Bayes

Naive Bayes is based on applying Bayes theorem (with the strong assumption that every feature is independent of the others) in order to predict the category of a given sample. It is a probabilistic classifier, meaning that it will calculate the probability of each category using Bayes theorem, and the category with the highest probability will be output.

In this case we have not tuned any hyperparameter.

Multinomial Logistic Regression

Multinomial Logistic Regression is a classification method that generalizes logistic regression to multiclass problems.

The list of hyperparameters we have tuned is:

Hyperparameter	Brief Description
C	Inverse of regularization strength. Smaller values specify stronger regularization.
multi_class	We'll choose *multinomial* because this is a multi-class problem.
solver	Algorithm to use in the optimization problem. For multiclass problems, only *newton-cg*, *sag*, *saga* and *lbfgs* handle multinomial loss..
class_weight	Weights associated with classes.
penalty	Used to specify the norm used in the penalization. The *newton-cg*, *sag* and *lbfgs* solvers support only l2 penalties.

Gradient Boosting

Boosting is a sequential technique which works on the principle of ensemble. It combines a set of weak learners and delivers improved prediction accuracy. At any instant t, the model outcomes are weighed based on the outcomes of previous instant $t-1$. The outcomes predicted correctly are given a lower weight and the ones misclassified are weighted higher. In the case of gradient boosting, the weak learners are normally decision trees.

The list of hyperparameters we have tuned is:
- Tree-Specific Parameters: the same ones as in the Random Forest.
- Boosting-Related Parameters:

Hyperparameter	**Brief Description**
learning_rate	Learning rate shrinks the contribution of each tree by *learning_rate*.
subsample	The fraction of samples to be used for fitting the individual base learners.

Baseline Classifier

In order to fix a baseline to be conscious about whether we are developing useful models or not, a baseline classifier that always predicted the majority class would be the simplest classifier we could build.

The majority class represents a 23% of the whole dataset. For that reason, that would be the accuracy of a majority class classifier.

3.4.3. Performance Measurement

As we stated in *3.3.4. Train – Test Split*, 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.

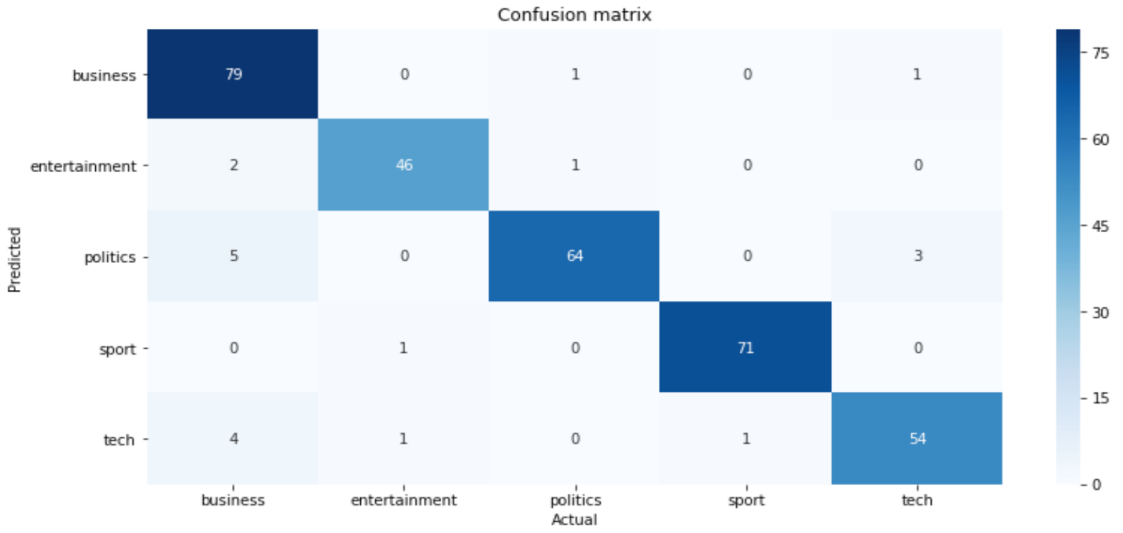
However, we have also obtained the confusion matrix and the classification report (which computes precision, recall and F1-score for all the classes) for every model, so we could further interpret their behavior.

3.4.4. 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.



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

3.4.5. 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.

After a brief study, we find that the model fails to classify articles that do not clearly belong to a unique class.

For example, there is an article that talks about the prohibition of Blackberry mobiles in the Commons chamber. This article is labeled as Politics, but it also talks about a technological issue. Our model has predicted its category as Tech.

These kind of errors are impossible to correct since there can be articles that *truly* belong to two or more categories at the same time.

3.4.6. 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.

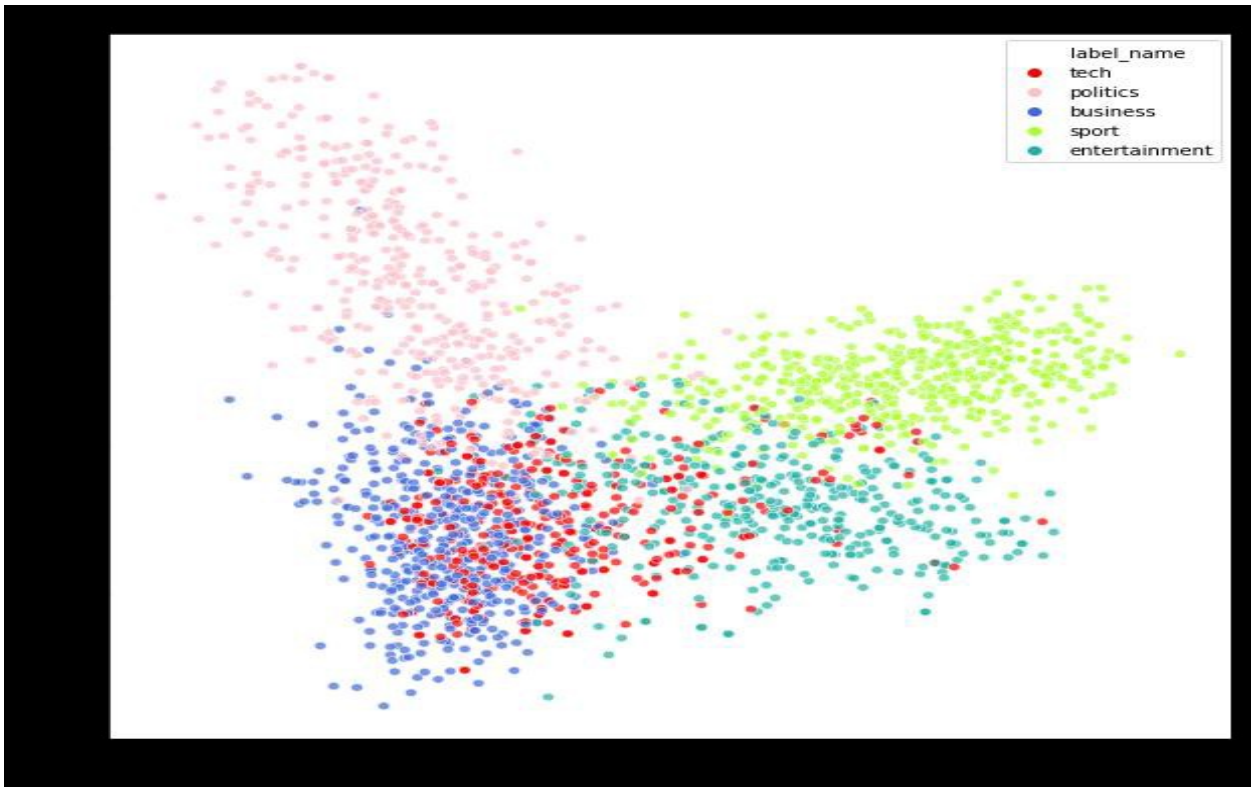
There are many applications of dimensionality reduction techniques in machine learning. One of them is visualization. By reducing the dimensional space to 2 or 3 dimensions that contain a great part of the information, we can plot our data points and be able to recognize some patterns as humans.

We have used two different techniques for dimensionality reduction:

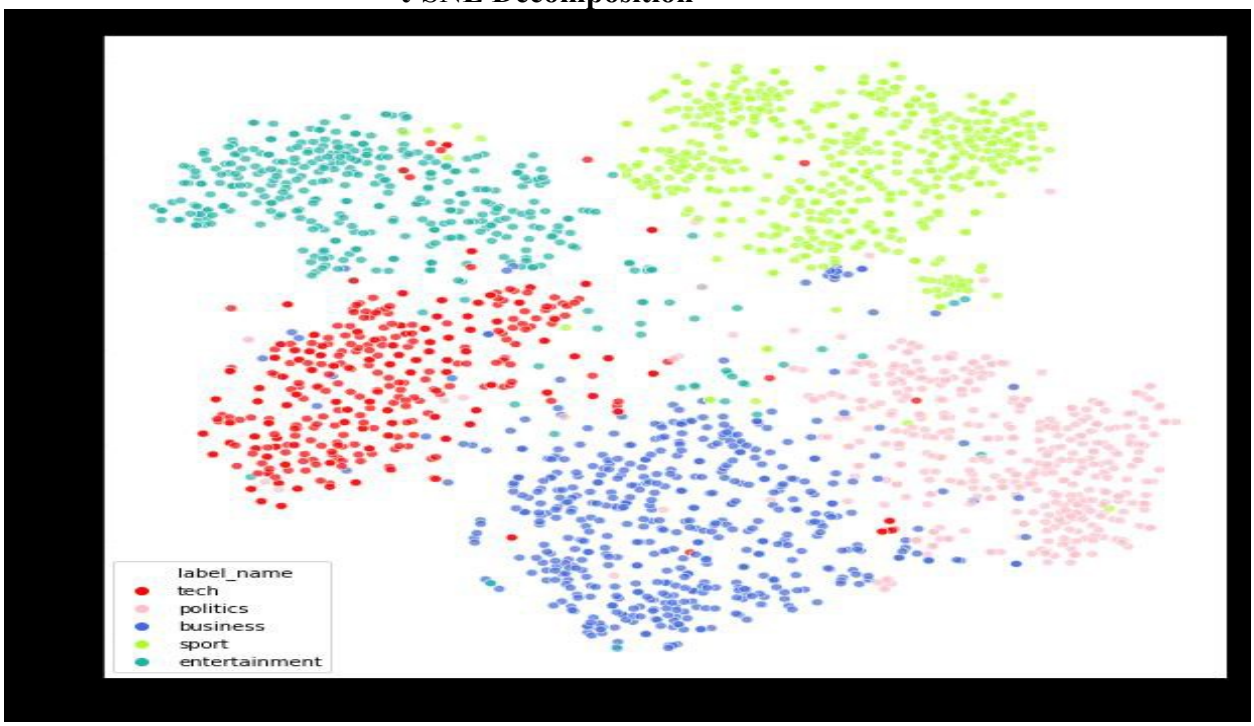
- **Principal Component Analysis:** this technique relies on the obtention of the eigenvalues and eigenvectors of the data matrix and tries to provide a minimum number of variables that keep the maximum amount of variance.
- **t-SNE**: the t-distributed Stochastic Neighbor Embedding is a probabilistic 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.

The plots are shown in the next figures:

PCA Decomposition



t-SNE Decomposition



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.

3.4.7. 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%**.

3.5. Web Scraping

The next step in the creation of our web application is to create some code that gets the latest news from different newspapers and stores them in a readable way.

We have developed a web-scraping code that gathers the news from the coverpage of a newspaper, gets into their link and scrapes the news body paragraphs. The newspapers for which we have done the scraping process are:

- El Pais English
- The Guardian
- Daily Mail
- The Mirror

We are interested in how much time the script takes to get the news because this will impact directly on user experience. In the next table we show the time elapsed to scrape 5 news for every newspaper:

Newspaper	Time
El Pais English	2.64 seconds
The Guardian	1.90 seconds
Daily Mail	31.88 seconds
The Mirror	11.07 seconds

3.6. Web Application

The last part of the project consists in unifying all the steps in a simple web application. The process can be resumed as follows:

1. Run the web scraping process to gather the news articles from the chosen newspapers.
2. Create the TF-IDF features of the input data.
3. Predict the category of each document from their features.
4. Show a summary.

the app can be launched locally. However, we have deployed the application in a web server so any user can have access to it. The detailed process of deploying the app to Heroku is shown in *Annex 5.2. Web App deployment with Heroku.*

4. Results and Conclusions

We have created an application based on machine learning from scratch. Throughout this document we have covered all the necessary steps to develop a service that can be used by any user in a simple way:

1. Getting data
2. Preparing and parsing the data
3. Exploring the data
4. Creating features from data
5. Training a model
6. Evaluating the performance of the model
7. Store the necessary information to predict new input data
8. Create and deploy a service based on the model

In this project we have shown that we can indeed use a text classifier to select interesting articles from a small dataset based on personal opinion. We believe that with further research and testing, the results we have found can be relevant for actors in the newspaper industry. While our tests were performed on data from only one newspaper we believe that the same general solution can be used for other newspaper companies.

One part of the project that we could have done differently was the tests where we used article tags as input. If we had discovered that only a small number of the articles had tags we could have gathered more articles for the datasets. If we had a larger article base we could create the intended datasets with size 50, 100, 150 and 200. With tests run on the same datasets but different inputs, we could draw more concrete conclusions from the comparison.

4.1 Future work

If we could continue this experiment we would want to conduct an experiment on what the best way is to find out if a user thought an article was interesting or not. We would like to compare how the classification confidence change depending on how the classification of articles are gathered from the user. For example; the time spent on an article could be used as an indication on whether the user thought the article was interesting or not. Another could be that the user could classify the articles themselves with the use of like/dislike buttons.

To make the results even more relevant to the news paper industry it would be interesting to research if a classifier trained on one newspaper's articles could be used to classify articles from other newspapers.

It would also be interesting to train a classifier with articles from several different newspapers. We would also like to do further testing on datasets with larger sizes to see if we can further improve the results from this project.

Performance is another thing that could be examined. We did not measure performance for our tests but it could be an important factor when deciding how effective the classifier is.

5. References

[1] F. Sebastiani, “Machine learning in automated text categorization,” ACM Computing Surveys (CSUR), vol. 34, no. 1, pp. 1–47, 2002.

[2] S. Rönnqvist, “Maskininlärning och tillämpningar inom genexpressionsanalys,” Kandidatexamen, Åbo Akademi, 2010.

[3] T. Joachims, “Text categorization with suport vector machines: Learning with many relevant features,” Proceeding ECML ’98 Proceedings of the 10th European Conference on Machine Learning, vol. 1398, pp. 137–142, 1998.

[4] V. T. M. Ikonomakis, S. Kotsiantis, “Text classification using machine learning techniques,” WSEAS TRANSACTIONS on COMPUTERS, vol. 4, no. 8, pp. 966–974, 2005.

[5] B. W. T.A. Meyer, “Spambayes: Effective open-source, bayesian based, email classification,” In Proceedings of the First Conference on Email and Anti-Spam (CEAS), 2004.

[6] J. T. W.B.Cavnar, “N-gram-based text categorization,” In Proceedings of SDAIR94, 3rd Annual Symposium on Document Analysis and Information Retrieval, pp. 161–175, 1994.

[7] D. K. S. Tong, “Support vector machine active learning with applications to text classification,” The Journal of Machine Learning Research, vol. 2, pp. 45–66, 2002.

[8] A. M. L.D. Baker, “Distributional clustering of words for text classification,” SIGIR ’98 Proceedings of the 21st annual international ACM SIGIR conference on Research and development in information retrieval, pp. 96–103, 1998.

[9] D. B. M. Pazzani, “Learning and revising user profiles: The identification of interesting web sites,” Machine Learning, vol. 27, no. 3, pp. 313–331, 1997.

[10] J. Schneider, “Cross validation,” 1997, [Online; accessed 26-October-2016]. [Online]. Available: http://www.cs.cmu.edu/~schneide/tut5/node42.html

[11] K. N. A. McCallum, “A comparison of event models for naive bayes text classification,” 1998.

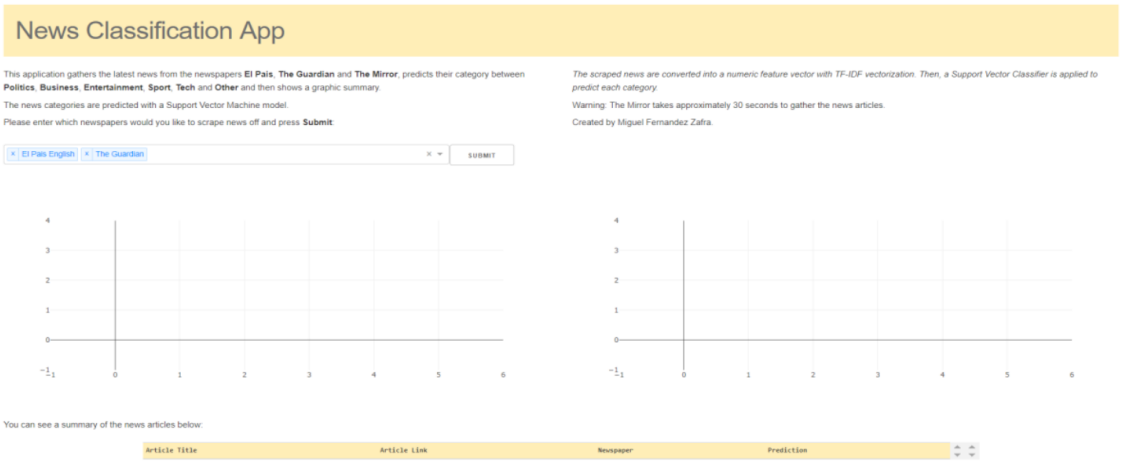
[12] H. C.D. Manning, P. Raghavan, Introduction to Information Retrieval. Cambridge University Press, 2008.

6.Annexes

6.1 Dashboard instructi

The dashboard is really simple to use: just choose the newspapers you want news scraped off and press Submit. A dashboard will be shown with a graphic summary and a table with the articles and predicted categories.

You can see a demo below:



After pressing submit: