Profile-Based Retrieval System

Manuel Scurti

1 Introduction

In many real-world scenarios, the ability to automatically classify documents into a fixed set of categories is highly desirable. Common scenarios include classifying a large amount of unclassified archival documents such as newspaper articles, legal records and academic papers. For example, newspaper articles can be classified as ’politics’, ’sports’ or ’entertainment’. Other scenarios involve classifying of documents as they are created. Examples include classifying movie review articles into ’positive’ or ’negative’ reviews or classifying blog entries using only a fixed set of labels. Natural Language Processing offers powerful tools to address these tasks. These techniques rely all on the hypothesis that documents of different categories distinguish themselves by features of the natural language contained in each document. Such features could include sentence structure and word frequency. The aim of this project is to develop a method based in the vector space model to deliver small text snippets of news articles to different users depending on their profile. At disposal there is a large archive of news articles provided by Kaggle containing around 200k news headlines from 2012 to 2018 obtained from HuffPost. Furthermore, New York Times, BBC News and The Guardian were selected as additional data sources to test the models with unseen data. This project experiments with different models as well as natural language process features. Specifically, Naive Bayes, SVM and Logistic Regression classifiers are being tested.

2 Related Work and Overview of Classification Techniques

There are different types of classification techniques that have demonstrated to show reasonable performances for document classification. Here is a short description of all classifiers used in this project.

2.1 Naive Bayes Classification

Bayesian classifiers are probabilistic approaches that make strong assumptions about how the data is generated, and posit a probabilistic model that embodies these assumptions. Bayesian classifiers usually use supervised learning on training examples to estimate the parameters of the generative model. Classification on new examples is performed with Bayes' rule by selecting the category that is most likely to have generated the example. The naive Bayes classifier is the simplest of these classifiers, in that it assumes that all features of the examples are independent of each other given the context of the category. This is the so-called "“naive Bayes assumption.” While this assumption is clearly false in most real-world tasks, naive Bayes often performs classification very well. Despite this, practical applications of naive Bayes classifier have had high degrees of accuracy in many cases. The naive Bayes classifier is much simpler and much more efficient than other supervised learning. This is largely because of the independence assumption which allows the parameters for each features to be learned separately. In the case of document classification, number of features is document classification is usually proportional to the vocabulary size of the training document set. This number can be quite large in many cases so the efficiency of the naive Bayes classifier is a major advantage over other classification techniques[4]. In this project two different Naive Bayes models were implemented: Bernoulli Naive Bayes and Multinomial Naive Bayes. The differences between the two, is the underlying distribution. Bernoulli models the presence/absence of a feature while Multinomial models the number of counts of a feature.

2.2 Support Vector Machine Classification

SVM is used for text classification tasks such as category assignment, detecting spam and sentiment analysis. It is also commonly used for image recognition challenges, performing particularly well in aspect-based recognition and color-based classification. SVM also plays a vital role in many areas of handwritten digit recognition, such as postal automation services. A Support Vector Machine (SVM) is a supervised machine learning algorithm that can be employed for both classification and regression purposes. SVMs are more commonly used in classification problems and as such, this is what we will focus on in this post. SVMs are based on the idea of finding a hyperplane that best divides a dataset into two classes, but can be extended to more than two classes. Support vectors are the data points nearest to the hyperplane, the points of a data set that, if removed, would alter the position of the dividing hyperplane. Because of this, they can be considered the critical elements of a data set. As a simple example, for a classification task with only two features, you can think of a hyperplane as a line that linearly separates and classifies a set of data. Intuitively, the further from the hyperplane our data points lie, the more confident we are that they have been correctly classified. We therefore want our data points to be as far away from the hyperplane as possible, while still being on the correct side of it. So when new testing data is added, whatever side of the hyperplane it lands will decide the class that we assign to it [7].

2.3 Logistic Regression

Logistic Regression was used in the biological sciences in early twentieth century. It was then used in many social science applications. Logistic Regression is used when the dependent variable(target) is categorical, as in our case for document classification. Logistic regression is named for the function used at the core of the method, the logistic function. The logistic function, also called the sigmoid function was developed by statisticians to describe properties of population growth in ecology, rising quickly and maxing out at the carrying capacity of the environment. It's an S-shaped curve that can take any real-valued number and map it into a value between 0 and 1, but never exactly at those limits [6].

$$\frac{1}{1 + e^{-value}} \tag{1}$$

Where e is the base of the natural logarithms (Euler's number) and value is the actual numerical value that you want to transform

3 Research Design

3.1 Problem specifications

The aim of this project is to develop a method based in the vector space model to deliver small text snippets of news articles to different users depending on their profile. The space of possible news category has been restricted, as a design choice, to 7 categories. Furthermore, the users dataset is randomly generated from scratch.

3.2 Approach

The project development has been divided in two phases: Model Building, and Use-Cases Implementations. In the first phase, the objective is to obtain a model, with reasonable accuracy, to classify news articles in 7 categories: Politics, Entertainment, Business, Health, Travel, Sports, Science. In this phase different models and feature sets have been tried. In the second phase, the obtained model is used as part of an application to solve two possible real case scenario: Profile-Driven Search

Engine and Profile-Based Newsletter which they will be described later. Model building phase was implemented using Scikit-learn while the application phases uses Scikit-learn, NLTK modules and GenSIM modules.

4 Data Sources

News articles were retrieved in English language, to be able to use the most advanced toolkits available in the NLP research field. Two data sources were exploited: RSS Feeds, Existing Dataset.

4.1 RSS Feeds

RSS feed is a protocol for web contents distribution. It is widely adopted by content creators on the web and gives the users the possibility to subscribe to any RSS feed to get the latest available contents on the web. In the case of a newspaper website they usually provide the latest news. In this way, news articles were retrieved from the RSS feeds of 3 different, popular, english-written, news sources:

- New York Times - https://www.nytimes.com/
- BBC News - https://www.bbc.com/news
- The Guardian - https://www.theguardian.com/us

Initially, the seven categories were different from the ones used in the final model. The total number of news articles in english language retrieved, in the retrieval date, are 810, this is obviously time dependent. Initially this set was used to train classifiers but due to limited number of samples, no good performances were obtained with this amount. Another data source was then considered. Articles from RSS feeds will be used in the use case scenario described below.

4.2 Existing Dataset

The design choice was then to rely on existing datasets of news article. For this purpouse, the Kaggle website has given access to a large set of news articles called "News Category Dataset"[8]. This dataset contains around 200k news headlines from the year 2012 to 2018 obtained from HuffPost. The model trained on this dataset could be used to identify tags for untracked news articles or to identify the type of language used in different news articles. Each row of the article contains the headline, a short description and the category of the article. The total number of distinct categories are 41, but from them, only 7 were chosen to build the model. This choice was made also to mantain consistency with the set of categories already present in the RSS feeds. Furthermore, the data was filtered by date, to include only articles starting from the mid 2017. This choice was made to address the problem of time dependency described later on in the performance evaluation.

4.3 Preprocessing

Preprocessing consisted in five phases:

- Lowercase text
- Word extraction by removing special characters
- Stopwords removal
- Stemming
- Tokenization

Stemming is the process of reducing inflected (or sometimes derived) words to their word stem, base or root form[2]. This is useful because some words can map to the same meaning and thus, with stemming we reduce the number of words in the dictionary. After preprocessing, all documents are represented by vectors of tokens, which are then passed to the feature extraction phase. Before training, the existing dataset was cutoff to 2000 samples per category.

4.4 Generating Users Dataset

Since there is no need of real users data. The project experimented results using random user data, generated from scratch. The assumption is that each user can choice at most 5, out of 7, categories from which they would like to receive news about. The generation of random numbers is provided in a built-in package in Python called **random**.

5 Feature Extraction

First step in model building is to transform the raw data into numerical data. This is a key-choice of every model design. The goal is to select the most informative features from the observations. The more discriminative they are the more accurate will be the classifier trained on this data. This is because you're describing to the classifier how to distinguish the different target labels in the data. Furthermore the raw data, cannot be fed directly to the algorithms themselves as most of them expect numerical feature vectors with a fixed size rather than the raw text documents with variable length[3]. In order to address this, I decided to use two of the most common ways to extract numerical features from text content:

- **counting** the occurrences of tokens in each document
- **TF-IDF score**

Using these measures, features and samples can be represented as follows:

- each individual word occurrence frequency (normalized or not) is treated as a feature.
- each document becomes a vector of all the word frequencies.

A set of documents can thus be represented by a matrix with one row per document and one column per word occurring in the corpus. This process is called vectorization, i.e. the process of transforming a set of documents into numerical feature vectors. This representation is called the Bag of Words or "Bag of n-grams" representation. Documents are described by word occurrences while completely ignoring the relative position information of the words in the document. In this project the Bag of Words is composed by 1-grams and 2-grams i.e. all the distinct words appearing in the text and all possible couples of words appearing in the text.

5.1 Possible Issues - Sparsity

As most documents will typically use a very small subset of the words used in the corpus, the resulting matrix will have many feature values that are zeros. In order to be able to store such a matrix in memory but also to speed up algebraic operations matrix / vector, Scikit-learn already provides sparse matrix representation to save memory and computational cost. This is the main reason of using Scikit-learn instead of NLTK for Model Building.

5.2 Word Frequency - Counting

Each document is represented by a vector of M columns, where M is the total number of words in the dictionary. Each entry of the vector contains an integer representing the number of occurences of that dictionary word in the document.

Trump	Brexit	Real Madrid	Bruno Mars
5	2	0	0

5.3 TF-IDF Score

TF-IDF score, often used in information retrieval and text mining, stands for *Term Frequency - Inverse Document Frequency*. The TF-IDF weight is a statistical measure used to evaluate how important a word is to a document in a collection or corpus. The importance increases proportionally to the number of times a word appears in the document but is offset by the frequency of the word in the corpus (dataset). Tipically the TF-IDF measure is a combination of two weights:

- Term Frequency (TF)
- Inverse Document Frequency (IDF)

IDF measures how important a term is. Since TF considers all terms equally important, we can't only use term frequencies to calculate the weight of a term in the document. Term frequency is the occurrence count of a term in one particular document only; while document frequency is the number of different documents the term appears in, so it depends on the whole corpus. The two terms can be defined as following:

$$TF_{ij} = \frac{n_{ij}}{|d_j|} \tag{2}$$

n_{ij} = number of occurences of the word i in the document j
d_j = length, in terms of number of words, of the document

$$IDF_j = log\frac{|D|}{|\{d : i \epsilon d\}|} \tag{3}$$

D = number of documents in the dataset
$denominator$ = number of documents containing the word i

$$TFIDF_{ij} = TF_{ij} \times IDF_i \tag{4}$$

6 Models Implementations

All models were trained and tested using Scikit-Learn, a popular library, written in Python, used mainly for Machine Learning tasks. This library already implements some of the most used classifiers for supervised classification. Naive Bayes Classifier with its variants can be implemented by means of the packages **sklearn.naive_bayes.BernoulliNB** and **sklearn.naive_bayes.MultinomialNB**. Logistic Regression and Support Vector Machine were implemented instead with **sklearn.linear_model**

.LogisticRegression and **sklearn.svm.LinearSVC**. The choice of Linear SVC instead of **sklearn.svm .SVC** was driven by the fact that the algorithm used in LinearSVC by the liblinear implementation is much more efficient than its libsvm-based SVC counterpart and can scale almost linearly to millions of samples and/or features[1]. To compare the different models with different feature sets, the training process was conducted using K-Fold Validation. K-Fold validation is a resampling procedure used in applied machine learning to estimate the skill of a machine learning model on unseen data. That is, to use a limited sample in order to estimate how the model is expected to perform in general when used to make predictions on data not used during the training of the model. It generally results in a less biased or less optimistic estimate of the model skill than other methods, such as a simple train/test split. With this method each sample is given the opportunity to be used in the hold out set 1 time and used to train the model k-1 times[5]. In the following described training process, K was chosen to be equal to 8:

```
for each feature extractor do
    for each model do
        for each K-Fold do
            1. create dictionary from training set;
            2. vectorize training set using feature extractor;
            3. vectorize test set using feature extractor;
            4. train model;
            5. test model;
            6. save performances;
        end
    end
end
```

Algorithm 1: Training process

Note that step 1 and 2 are already implemented by means of a scikit-learn method called **fit_transform()**.

6.1 Feature Extractors Configuration

Skikit-Learn package has already built-in functions to convert a collection of raw documents to a matrix of TF-IDF features or Word Frequency features. These two methods accepts some parameters to be configured and fitted to the problem requirements. These are the parameters used to configure both feature extractors.

```
CountVectorizer(ngram_range=(1, 2), token_pattern=r'\b\w+\b', min_df=5)
```

```
TfidfVectorizer(sublinear_tf=True, min_df=5, norm='l2',encoding='latin-1',
ngram_range=(1, 2))
```

6.2 Naive Bayes Configuration

Two different naive bayes classifier were tested in this project: Bernoulli Naive Bayes and Multinomial Naive Bayes. The Bernoulli Model assumes that all our features are binary such that they take only two values. Means 0s can represent “word does not occur in the article” and 1s as "word occurs in the article". Multinomial Naive Bayes uses instead the frequency of each word to predict the class or label. The parameters of the two models were leaved as default values.

6.3 Logistic Regression Configuration

Logistic Regression model offers different solvers, 'lbfgs' is indicated for multiclass problems. Max number of iterations was setted as shown below.

```
    LogisticRegression(solver='lbfgs', multi_class = 'auto', max_iter = 400)
```

6.4 Support Vector Machine Configuration

No modifications was done on default values, apart from the number of maximum iterations.

```
    LinearSVC(max_iter = 1000)
```

7 Performance evaluation and comparisons

Different measures were computed for each confiuration of the model. This includes: confusion matrix, precision, recall, accuracy for each K-fold in the training process. The following tables resume the results obtained. First of all let’s figure it out which feature set is performing better.

Comparison of performances using different feature sets				
Best accuracy score	Logistic Regression	BernoulliNB	LinearSVM	MultinomialNB
TF-IDF	**0,808**	**0,8028**	**0,8005**	**0,8131**
Word Frequency	0,7811	0,7954	0,7434	0,8011

TF-IDF feature set seems to perform slightly better than Word Frequency feature set, in particular in the SVM model. Let's then visualize which model performs better with this feature set, considering also the repeatability of the performances during the K-Folds.

Comparison of performances using TF-IDF feature set				
Accuracy score	Logistic Regression	BernoulliNB	LinearSVM	MultinomialNB
Max	0,808	0,8028	0,8005	**0,8131**
Average	0,7938	0,7881	0,7836	**0,7962**
Std Dev	**0,0089**	0,0116	0,0125	0,0106

As showed in the table, the Multinomial Naive Bayes model slightly outperforms the other models. This model is also the final choice to be implemented in the application, because showed a fast training time with respect to the other models. Finally let's see how the model behaves with the differents categories.

Beahaviour of the chosen model			
	Precision	Recall	f1-score
BUSINESS	0.75	0.72	0.73
ENTERTAINMENT	0.81	0.84	0.83
HEALTH	0.72	0.77	0.74
POLITICS	0.83	0.85	0.84
SCIENCE	0.89	0.77	0.82
SPORTS	0.86	0.86	0.86
TRAVEL	0.84	0.88	0.86
micro avg	0.81	0.81	0.81
macro avg	0.82	0.81	0.81
weighted avg	0.82	0.81	0.81

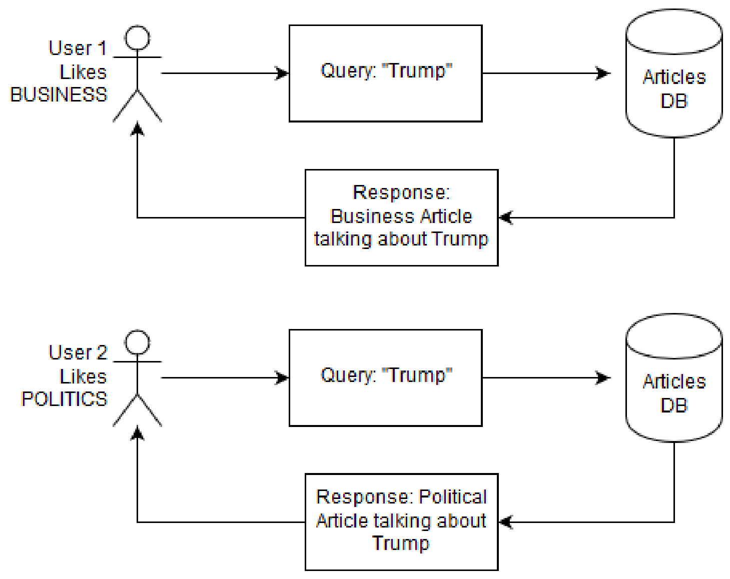
It is interesting to notice that Business and Health are the two worst predicted categories. This could be due to some semantic interference between the category "Politics" and "Business" because they are not completely uncorrelated. e.g. one article could talk both of politics and business when talking about some economic improvements applying political changes. The same could happen for "Health" and "Science", because it can happen that a science article talk about health improvements. The final model chosen is then a Multinomial Naive Bayes classifier with Tf-Idf feature set. The final deploy version of the model achieved an accuracy score of **0.8032** on the test set. Then the model was tested on unseen data retrieved from the daily news of the New York Times and it achieved **0.6648** of accuracy. This is lower than expected, but this can be explained as a generalization problem of the model. Data collected in the training dataset is coming from HuffPost while unseen data is coming from New York Times, and thus, they can adopt a different set of terminologies to describe the same thing. Furthermore, since this model rely on a dictionary based feature set, it is time dependent on the terminologies used by newspaper writers, and world facts at that time, e.g. the word "Brexit" is most used nowadays than before, and could have been used by the model as a key discriminatory term for articles talking about politics. It should be mentioned also that Training score is affected by the relations between the categories, a proper selection of very distinct categories between each other can boost the performances of the model, given also the fact that if we reduce the number of target labels the problem will get easier. This has been tested but not reported here.

8 Use Cases in Real World Scenario

Two use cases were designed for the document classifier. Both were implemented in Python using NLTK and Scikit-Learn packages.

8.1 Profile-Driven Search Engine

Personalized search refers to web search experiences that are tailored specifically to an individual's interests by incorporating information about the individual beyond specific query provided. In this case, for each individual we know which are the categories to which it is interested to. One example is Google. Google often shows the ability to personalize searches in its use of Google News. They implement this feature in its news to show everyone a few similar articles that can be interesting, but as soon as the user scrolls down, it can be seen that the news articles begin to differ. Google takes into account past searches as well as the location of the user to make sure that local news gets to them first. The goal of this first use case is to differentiate the behaviour of the ranking score by including the user profile. This translates to a modification of the ranking formula, that gives a score for each result to describe the relevance with respect to a given query.



In my implementation, the formula for computing the relevance score takes into account the cosine similarity between two vectorized documents and penalizes by a factor of 0.5 the articles not belonging to the set of categories that the user likes, otherwise it increases the score by a factor of 1.5. The application works in two step, as every basic information retrieval system. The first step is to create an archive of articles to be indexed for further retrieval. To do this, articles are retrieved, transformed into a vector representation, labeled by the classifier and stored in memory. Then the user, with its known profile, queries the database and the learning algorithm assign a relevance score to the search results.

8.2 Profile-Based Newsletter

In this era of information explosion, providing the right information to the right person within reasonable time duration is a very important goal for today's information retrieval (IR) systems. In this use case, a user wants to receive all the news available on the web, that are related to his interests. The system retrieves a set of news articles, classify them and finally distributes the news to the most appropriate users. This behaves like a newsletter, but with the integration of an automatic system to estimate the probability of relevance of that article for that user. A critical component in this application is the performance of the classifier, because all the articles received by the users rely on its ability to correctly identify the article category. The first step of the application is to retrieve all the articles available in the provided RSS feeds, then, the classifier estimates the category for each article, so that it will be then dispatched to the interested users.

9 Conclusion and Future Work

In this paper I introduced a comparison of different approaches to build a document classification model, comparing different feature sets and classifiers, then, i concluded by showing two possible ways of application in real world scenario. As we've seen, the performances showed by the Multinomial Naive Bayes Classifier clearly outperforms the others, both in accuracy measures and in computational cost. The model, by the way, is still far from being adopted for commercial use, since generalization error are quite evident when using unseen data. This could be for sure addressed in further analysis, by using more data and more computational power. Also other models were not tested here, like Deep Neural Networks, which they showed great performances in several NLP tasks. I believe that Profile-Based retrieval system will become more and more useful in the era of Big Data and thus, there will be always room for making improvements in these systems.

References

[1] Scikit Learn documentation. https://scikit-learn.org/stable/modules
/generated/sklearn.svm.LinearSVC.html.

[2] Stemming wikipedia. https://en.wikipedia.org/wiki/Stemming.

[3] Stemming wikipedia. https://scikit-learn.org/stable/modules/feature$_{e}xtraction.html.$

[4] K. N. Andrew McCallum. A comparison of event models for naive bayes text classification. 1998.

[5] e. a. Gareth James. *An Introduction to Statistical Learning*. 2014.

[6] A. Genkin, D. D. Lewis, and D. Madigan. Large-scale bayesian logistic regression for text categorization. *Technometrics*, 49(3):291–304, 2007.

[7] T. Joachims. Text categorization with support vector machines: Learning with many relevant features. 1998.

[8] R. Misra. News category dataset, 06 2018.