BBC News Article Classification Project

Xiaoxiong Xu, Sharis Ochs, and Andie Donovan

PSTAT 192 Spark Programming

Professor Adam Tashman

19 March 2018

Abstract

In 2016, 43% of Americans reported getting news from an online source.[1] As news becomes an increasingly digitized industry and the sheer quantity of information on the Internet grows exponentially, advancing the computer recognition and processing of human news outlets has become an important and interesting task to data scientists. In this project, we used natural language processing and machine learning techniques to classify online news articles into one of five genres. After comparing Random Forest, Naïve Bayes, Logistic Regression, and Neural Network techniques, we found that all four had very high and comparable accuracy rates when predicting labels on our test set. Although a simplification of the problem faced by governments and social media platforms globally, the project serves to investigate the relationship between numerical metrics, such as word counts, and the computer's understanding and predictive abilities in the context of human language.

Data and Methods

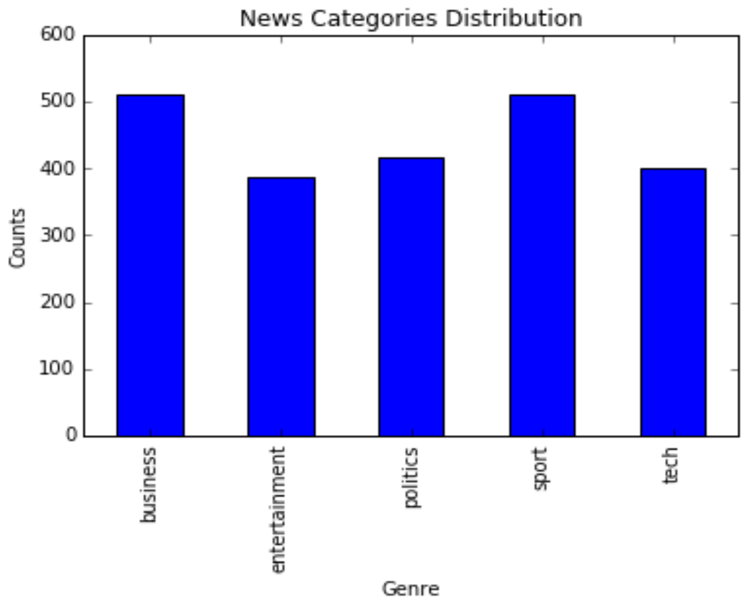
The dataset we used for the project consists of 2,225 news articles extracted from the BBC website between 2004 and 2005.[2] The articles were pre-labeled as having Entertainment, Technology, Business, Sports, or Political content. The dataset was published open source by Insight Resources and was collected by UC Davis for research purposes.

The initial steps in the data preprocessing were to remove non-alphanumeric characters from the text corpus and convert all characters to lowercase. This removes noise from the data and ensures that the models are processing actual text rather than punctuation and other symbols. In plotting

[1] According to a study by the Pew Research Center. Furthermore, the investigation reported that ⅔ of Americans get at least some portion of their news through social media. See Gottfried and Shearer citation.

[2] For the source of the dataset, please see the Greene and Cunningham citation.

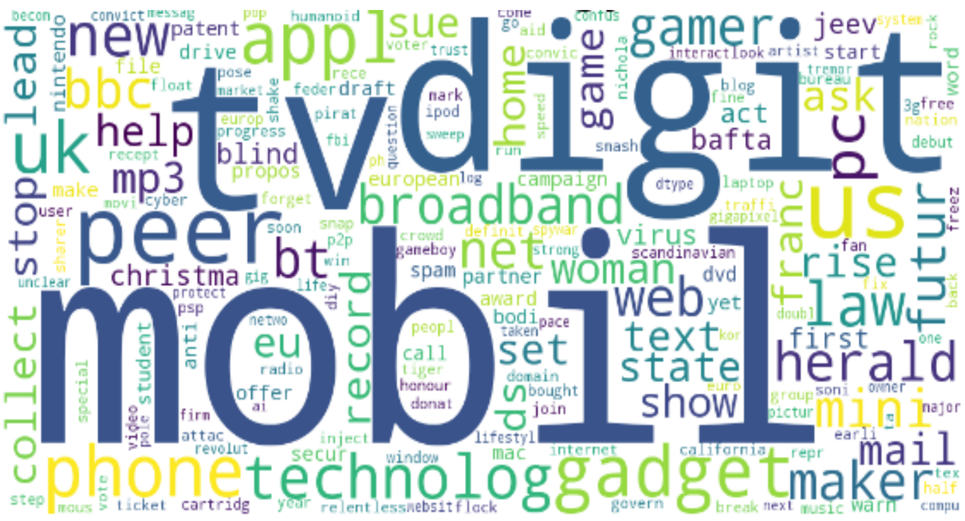
the distribution of the class sizes, or, in other words the number of text files in each news category, it was evident that the data was slightly imbalanced. As shown in the figure below, while business and sports had 510 and 511 files respectively, technology and politics only contained 401 and 417 files each and entertainment only contained 386. At first, we considered under-sampling to make each category exactly equal, but eventually decided to train on all of the data instead. Since the number of articles in each category is not extremely imbalanced, we found it best to keep all the data available, giving the model more input and thus, improving the accuracy.



Next, we used natural language processing to parse and process the text in the news articles. The first step in this stage was to tokenize, or split, the data into smaller chunks by separating the phrases into individual words. To tokenize the data, we used the RegexTokenizer function, which utilizes regular expression pattern recognition, in our case word characters, to parse the text and extract tokens. This is a slightly more accurate form of tokenizing because it depends on regular expression encodings rather than simply parsing words by spaces, therefore minimizing errors caused by nuances in the textual data. After tokenizing the data, we reduced the dimensionality of the data by extracting the stop words. Stop words are words such as "a", "and", "that", and "his" that do not contribute to the underlying meaning, topic, or classification of the text and therefore should be filtered out. Although there is no set declaration of what stop words are or should be, we used the built in function StopWordsRemover, which contains a pre-defined dictionary of English stop words, for simplicity.

After removing stop words, the tokenized words were stemmed to reduce noise in the word frequency counter. Stemming is the process of reducing words to their root or "stem" word form. For example "shopping" might be reduced to "shop". This process, which is similar to lemmatization, is useful for grouping together words that have essentially the same meaning. To perform the stemming, we used the function SnowballStemmer, an NLTK stemming algorithm, with the language parameter set to “English”.

Following the stemming step, we plotted word clouds for each of the news categories to gain a better understanding of which words were most important and common to the class. For example, in the word cloud below for technology, the stemmed forms of the words “mobile”, “game”, “chip”, “video”, “apple”, and “gadget” can all be seen as the most prominent words. We can expect the counts of these features, then, to be the most influential in the classification of a technology article.



In preparation for later processing, the class labels such as entertainment, business, technology, sports, and politics, needed to be transformed into numerical values. To perform this, we used the function StringIndexer from the pyspark.ml.feature module, which is a fast mapping tool for multiclass data. The numerical assignment of each news category can be seen in the table below.

Bin Label	**Label**
0.0	sports
1.0	business
2.0	politics
3.0	technology
4.0	entertainment

Machine Learning Models

After the data points were properly cleaned and pre-processed, we began the step of splitting the data into training and test sets. We decided to use 80% of the data as training data—the information upon which we would fit and build our models—and the remaining 20% of the data as a test set, used to determine the accuracy of the fitted models on data they have not seen before. The tradeoff between the training and test set sizes is that a larger training set typically gives lower parameter estimate variance while a larger test set reduces the variance in the model evaluation metrics.[3] We decided to follow the Pareto Principle and perform an 80-20 split for the training and test sets.[4] When creating the training and test sets, the data points selected for each group were randomly selected by using the RandomSplit function in PySpark. This ensures that the models were not biased due to a sampling error.

Following this, the parsed texts were converted into a numerical format so that they could be processed and read by the machine learning algorithms. Rather than apply a simple word counter, we applied a TF-IDF, or term frequency-inverse document frequency, transformative model. TF-IDF is a more complex method for determining how important a word is to the overall corpus of news articles by penalizing the words that appear more frequently in many of the samples. For our training set, we applied both TF and IDF, whereas for our test set, we applied only TF. This procedure is followed so that the features and their IDF weights are calculated based on the training set and then fitted to the test set using these predefined features.

$$\begin{aligned} tf &= \frac{f_t}{\sum f_t} \\ idf &= \frac{log(N)}{n_t} \\ tfidf &= tf * idf \\ &= \frac{f_t}{\sum f_t} * \frac{log(N)}{n_t} \end{aligned}$$

[3] In other words, a large training set provides more data for the model to learn from, but leaves less data to test the model with, weakening the researcher's diagnostic ability. See Guestrin citation.
[4] The Pareto Principle dictates that "80% of the effects come from 20% of the causes" and has empirical applications in wealth distribution, sales, and gambling scenarios. See Pareto Distribution citation.

Next, we initialized and fit the machine learning algorithms. For our project, we selected four fundamental models to apply to our data. These were: Multinomial Logistic Regression, Naïve Bayes, Random Forest, and Neural Network classifiers. By using a variety of parametric and non-parametric models, we hoped to avoid issues arising from assumptions of feature independence and Normality.

The first model we selected was the Multinomial Logistic Regression, which serves as the base model. Unlike other models such as the Naïve Bayes Classifier, Logistic Regression does not make any assumptions on the independence of the model features. It uses the logit model to assign each class a "score" or probability and then returns the most probable label as the predicted outcome. Our model uses the L-BFGS model for parameter selection by employing the family of quasi-Newton methods, which are used to either find the zero value or minima of the optimization function. This is done while incurring a less costly use of computer memory than the normal BFGS model.[5]

The next model applied was a Random Forest classifier. Random Forest is a type of ensemble learning that uses a collection of decision trees to classify data into groups. Decision trees aim to label the data by splitting the texts into homogeneous groups—as determined by maximum likelihood—based on binary, yes-no branches consisting of significant predictors such as word counts. Local classifications are then made at the terminal nodes, also called leaves, of each tree and then from the collective results of the forest, the model determines the most probable label for the text item at hand. Random Forests are fast and nonparametric, making them a good

[5] L-BFGS stands for Limited-memory Broyden–Fletcher–Goldfarb–Shanno algorithm. See RDD Optimization citation.

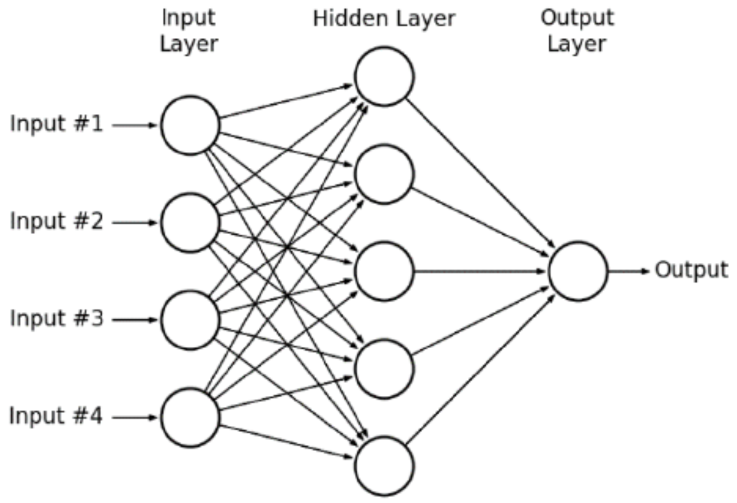
addition to our collection of algorithms. In order to enhance the performance of the Random Forest classifier, we applied a grid search to determine the optimal parameters for the model. After running the grid search on a series of parameter combinations, we concluded that a Random Forest with 200 trees and a maximum depth of 20 provided the best model, with an accuracy of 95.75%. Although a higher number of trees or a higher maximum depth would have likely decreased bias, the changes would have also increased variance, thus increasing the Mean Squared Error of the model.

Following the Random Forest, we utilized a Multiclass Naïve Bayes classification algorithm. Naïve Bayes is a probabilistic classification technique that uses the Bayesian concept of prior probabilities to determine the most likely class of the text. Although Naïve Bayes requires the somewhat dubious assumption of independence between text features, the model has empirically worked well with many other text and multiclass classification problems. To optimize the smoothing parameter for the Naïve Bayes algorithm, we ran a grid search using an array of decimal values. The grid search reported that after a smoothing parameter value of 0.2, there was little change in model accuracy. Nonetheless, a smoothing of 0.8 returned the absolute highest level of accuracy, giving us a rate of 97.32%. The formula for the Naïve Bayes algorithm can be seen below.

$$P(c|x) = \frac{P(x|c)P(c)}{P(x)}$$

Here, $P(c|x)$ represents the probability of class c, given predictor feature x (in our case, the array of word frequencies in the article). The class with the highest probability is then selected as the label.

The final model we used is a feed-forward artificial Neural Network model. We used the MultilayerPerceptronClassifier function from PySpark and then ran a grid search to determine the optimal values for the following parameters: maximum number of iterations, the block size, and the number of hidden layers. After running the grid search on 45 combinations of possible model parameters, we determined that setting the maximum iterations to 50, the block size to 50, and the number of hidden layers to 40 gave us the best and most efficient results for the neural network model with an accuracy rate of 97.76%.



In this diagram, the identity layers serve as the building blocks of the neural network. Because there are multiple hidden layers of perceptrons (artificial neurons), this is a multi-layer perceptron neural network and therefore has a non-linear activation function. The activation function serves to define the node's output given some set of inputs.

Results and Model Evaluation

To evaluate the performance of the Machine Learning algorithms, we calculated a few performance metrics for each model as a whole in addition to the model performance within the individual news class subcategories. Metrics analyzed include: accuracy, recall (sensitivity), precision (positive predictive value), and the F1 score.[6] This large combination of metrics allows us to properly identify certain fields or classification scenarios where our models are failing. The formulas for these model performance metrics can be seen below.

[6] The F1 score is the harmonic mean of precision and recall and simply serves as another measurement of the model's predictive abilities. For multi-classification tasks, such as news genre classification, the F1 score reported represents the average of the F1 scores of each class.

$$precision = \frac{tp}{tp + fp}$$

$$recall = \frac{tp}{tp + fn}$$

$$F_1 = 2 * \frac{(precision * recall)}{(precision + recall)}$$

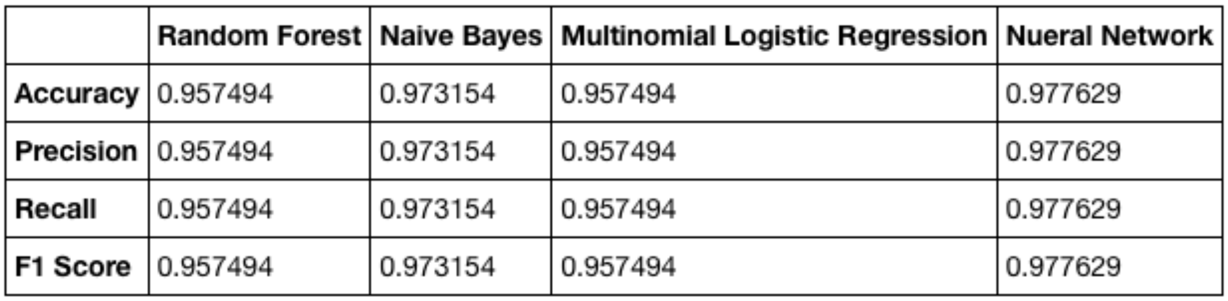
$$accuracy = \frac{tp + tn}{fp + tp + fn + tn}$$

In terms of accuracy, the Neural Network model performed the best. This was followed by the Naïve Bayes model, then the Logistic Regression, and finally the Random Forest model.

To determine how the model performed for each news category individually, we used the function MulticlassMetrics. For the Multinomial Logistic Regression model, politics and sports obtained the highest level of precision, followed by business, technology, and entertainment. When we applied the MulticlassMetrics function to the Naïve Bayes model, politics and business obtained the highest precision, followed by technology, sports, and then entertainment. For the Random Forest model, technology articles were identified with the highest precision, then entertainment, sports, business, and politics. Finally, the Neural Network reported that the highest precision was achieved for the entertainment category, while politics, technology, business and sports followed respectively. As we can see, there was no clear trend in which genres of article were best identified by our test models.

As for accuracy, the Neural Network model returned the highest level of accuracy, at 97.76%. Naïve Bayes returned the second highest at 97.32% accuracy, and Multinomial Logistic

Regression and Random Forest tied for the lowest accuracy rate, at 95.75%. These results can be seen summarized in the table below.



	Random Forest	Naive Bayes	Multinomial Logistic Regression	Nueral Network
Accuracy	0.957494	0.973154	0.957494	0.977629
Precision	0.957494	0.973154	0.957494	0.977629
Recall	0.957494	0.973154	0.957494	0.977629
F1 Score	0.957494	0.973154	0.957494	0.977629

Conclusions

In this work, we found that a Neural Network model trained on the numerical features of BBC news articles created through TF-IDF transformations can successfully classify articles into genres. We remain skeptical of precision and recall rates of 1 since this could have been due to overfitting. Future work would include verifying these rates and further investigation into how this could be corrected. Although the articles published by BBC tend to be relatively clean in terms of foreign languages and misspellings, the ability of our machine learning algorithms to classify documents at such a high level of accuracy is a promising result and a signal of computers' vast ability to comprehend and analyze human language.

Works Cited

Gottfried, J., and Shearer, E. “Americans' Online News Use Is Closing in on TV News Use.” Pew Research Center, 7 September 2017.

Greene, D. and Cunningham, P. "Practical Solutions to the Problem of Diagonal Dominance in Kernel Document Clustering", Proc. ICML 2006.

Guestrin, Carlos. “Overfitting and Logistic Regression.” Machine Learning. Carnegie Mellon University, 19 September 2007.

“Machine Learning Library (MLlib) Guide.” MLlib: Main Guide - Spark 2.1.1 Documentation, The Apache Software Foundation, 2016.

Mueller, A. Word Cloud Generator, 2012, DOI: 10.5281/zenodo.49907

“Optimization - RDD-Based API.” Optimization, RDD-Based API, Spark 2.3.0 Documentation.

“Pareto Distribution.” Statistics Online Computational Resource, UCLA, 18 July 2011. 2007.