Building a Fact-Checked Classifier with Naive Bayes… and More!

Kyle Otstot - Social Media Mining

1. Abstract

In this report, I first define the task assigned to me, then outline my approach to solve the task. More specifically, I was tasked with the implementation of a classification algorithm that is expected to learn on a feature-sparse, class-imbalanced training set. In later sections, I describe the data-preprocessing, algorithm structure, and experimental protocol of my solution, and end with an evaluation of several notable shortcomings.

2. Introduction

For this project, I was given the task of developing a machine learning (ML) algorithm that classifies the conclusions of fact-checking articles (paired with other data, such as the topic, source, etc.) as one of the following labels: FALSE (0), MISLEADING (1), TRUE (2), UNPROVEN (3). I was provided with a train set of ~6,000 examples and an unlabeled test set of ~700 examples. My objective was to train a model on the train set and predict the labels of the examples in the test set, which would be evaluated for accuracy on Kaggle. The two major obstacles in the task were (1) the heavy imbalance of labeling (mostly 0's), and (2) the need for feature engineering, as very little information was provided in the datasets. In summary, I decided to extract more features by requesting the HTML pages of each article in the dataset, and develop an algorithm that would primarily learn on these new features. My journey to an effective ML solution has been documented in the following sections.

3. Related Works

In this section, I list and provide an overview of two works related to the task of text classification. Although I did not go these specific routes, I found them to be good resources in general.

Text Classification Using Naive Bayes: Theory & A Working Example [1]. This resource provides a step-by-step implementation of Naive Bayes in the setting of text classification, using the Sci-Kit Learn library in Python. Naive Bayes did not work the best in the context of my task, but it still serves as an effective solution to the general problem of text classification.

TF IDF | TFIDF Python Example [2]. This resource provides a step-by-step implementation of TF-IDF (term frequency - inverse document frequency) in the setting of text classification, also using Sci-Kit Learn. I decided to not use TF-IDF but it pairs well with Naive Bayes to form an effective text-classification solution.

4. Model Description

In this section, I highlight each aspect of my solution and provide brief justification for implementation choices.

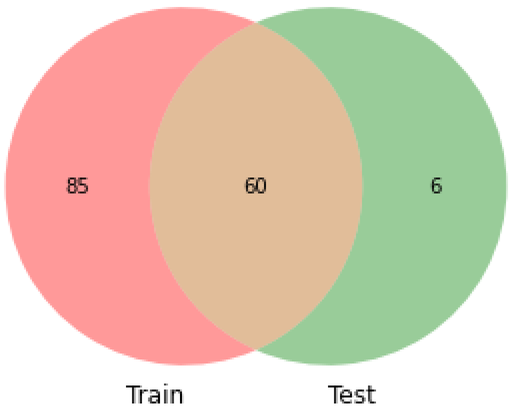
https://fullfact.org/health/800-killed-misinformation/ → fullfact.org (4.1)

Websites in the Dataset. Websites were extracted from the URLs in order to group similar articles (4.1). The model was constructed under the assumption that the fact-checked articles coming from the same website have similar underlying HTML structures. As a result, a narrowed focus on articles from the same website may reveal patterns in the HTML that the model can learn. Each dataset is composed of a manageable number of websites with very significant overlap (shown by 4.2). This allows the model to learn website patterns in the train set and apply this knowledge to websites in the test set .
Website Data Collection. Data-preprocessing includes the HTML request for each article in the train/test sets. Then I use the BeautifulSoup Python Library **[3]** to parse each HTML file for the following attributes:

- **Class names (**class**).** These help identify certain blocks of HTML for styling/scripting/etc. It is possible that same-labeled, same-website articles have more class names in common.

- **Image source links (**src**).** These are the URLs that contain images referenced by the article. It is possible that same-labeled, same-website articles include the same images (i.e. an image that says 'False').
- **Image alternate text (**alt**).** These are descriptions of images in the article. If articles have similar images, it is possible that the alt text is the same.
- **URL links (**href**).** These are clickable links to other websites/articles. It is possible that same-labeled, same-website articles share the same links.
- **Text within tags (**text**).** These include entire blocks of text within HTML tags as single tokens. Same-labeled articles of the same website may contain similar phrases that define its label.

(4.2) Websites in Train/Test sets



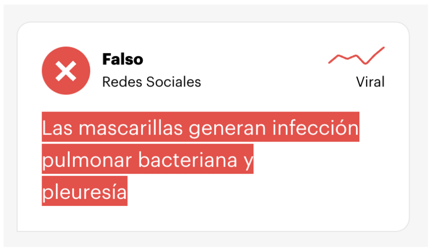
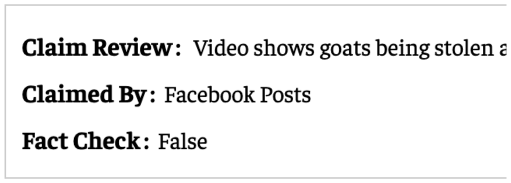
- **Website article content (**content**).** These attempt to reflect the actual text content of the articles. After parsing, the data includes a list of uni-grams, a list of bi-grams, and a list of tri-grams. The idea is that N-grams (when $N > 1$) **[4]** do a better job at preserving meaning since the ordering of words can be very important and must be taken into account.

Text Classification Models. The outcome of the data-preprocessing step includes a "bag" of words (tokens) for each attribute in the previous section. For instance, one article's HTML file will be parsed into a bag of class names, bag of image source links, bag of alt text, and so on. Then all the bags of the same attribute (across all articles on the website) + the label data will be passed to the text classification model for training. I use the following two models:

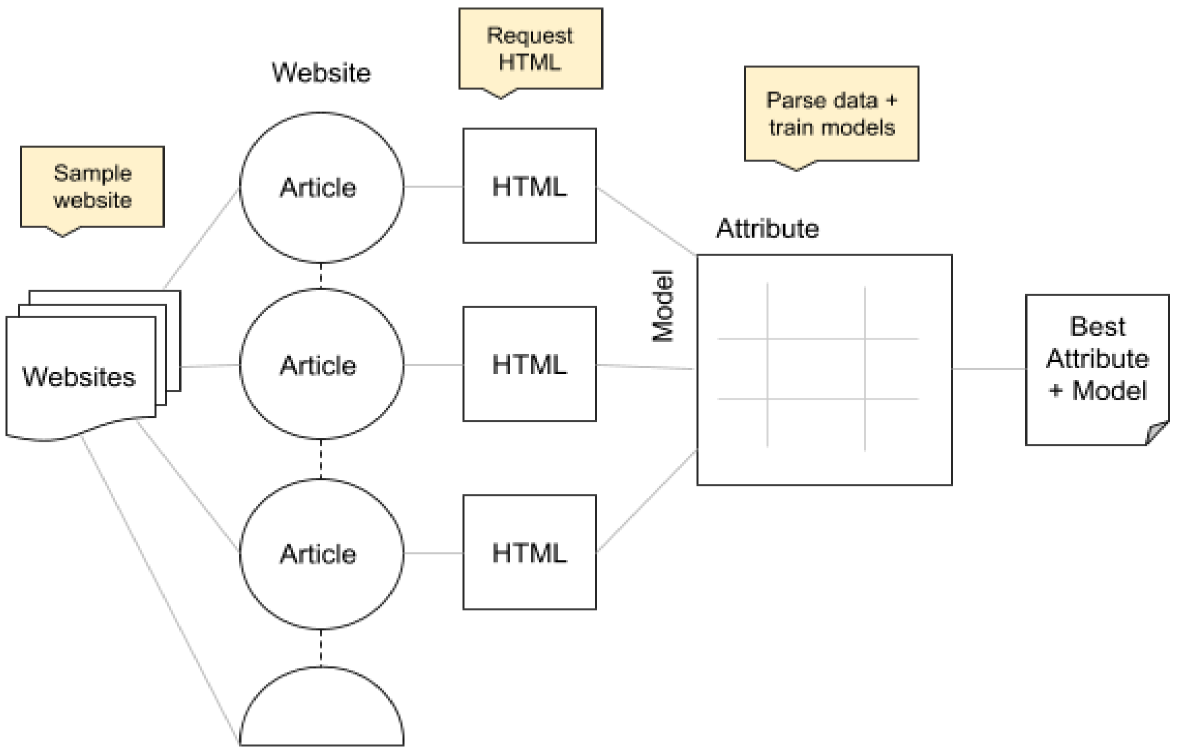
- **"Switches".** This original model ranks the attribute tokens based on its likelihood to be present in one label and absent in all the other labels. The model attempts to find tokens that best act as "switches" (that is, its presence/absence solely dictates the label of the article). This was created from the observation that many websites have specific keywords (classes, links, text, etc.) that define the labeling, and nothing else matters. Details on the implementation can be found in the source code.

- **Variant of Naive Bayes.** This model is loosely inspired by Naive Bayes **[1]** in the sense that it attempts to estimate a posterior distribution (label likelihood conditioned on a bag of tokens). However, it also takes into account the tokens that are *not* present in the article, since in this context, absent tokens can be just as telling as present tokens.

An example of a switch (model 1) classification on two different attributes is provided in the table below.

	src	text
Website	 Falso Redes Sociales Viral Las mascarillas generan infección pulmonar bacteriana y pleuresía	 **Claim Review:** Video shows goats being stolen a **Claimed By:** Facebook Posts **Fact Check:** False
"Switches" Results	**FALSE:** 'https://www.newtral.es//wp-content/uploads/2020/07/fake.svg?x97913%1' **0.99999999998** 'https://www.newtral.es//wp-content/uploads/2020/07/fake.svg?x97913%2' **0.99999999998** **MISLEADING:** 'https://www.newtral.es//wp-content/uploads/2020/07/fact-check-engañoso.svg?x97913%2' **0.99999999998** 'https://www.newtral.es//wp-content/uploads/2020/07/fact-check-engañoso.svg?x97913%1' **0.99999999998**	**FALSE:** 'False#span%1' **0.99999999998** 'COVID-19Pandemic\xa0#a%1' **0.47999999999279996** **MISLEADING:** 'Misleading#span%1' **0.99999999998** 'This#a%1' **0.599999999988**

ML Pipeline Overview. Here I provide a visualization of the ML pipeline applied to the task at hand, from website grouping to data preprocessing to optimal attribute+model search. First, the articles are grouped by website and a website is then sampled. Then the group of articles belonging to the sampled website are split into train and test sets. Attribute data is then extracted from each article, and one model (at a time) trains on the data for one attribute (at a time) from the articles in the train set, and is evaluated by the articles in the test set. Each attribute+model combination is tested, and the combination with the best evaluation accuracy is used to predict the labels for the unlabeled articles (from the real test set) for that website.



5. Experiment

In this section, I document the experimental procedure and results. The procedure follows the ML pipeline described above and can be broken into the following four steps:

Train-Test Split. Each website begins with a list of articles and a corresponding list of labels. The goal is to split up the articles into two mutually-exclusive groups- the train set and test set- where (1) the train set contains *N*% of the articles and the test set contains (100 - *N*)% of the articles; (2) the label distribution in the train set are consistent with that in the test set, and both are approximately equal to the overall label distribution. For this experiment, I used *N* = 70 because having a sufficient amount of test data is very important in considering the optimal attribute+model combination.

Model Training. Each attribute-model combination trains on data from articles in the train set. If the switch model is used, the train step passes a label,token->score dictionary to the evaluation step; if the Bayesian-like model is used, the train step passes a label-token posterior distribution to the evaluation step.

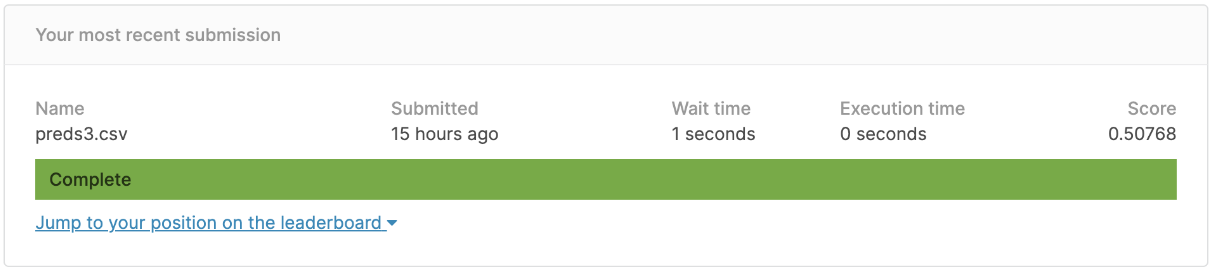
Model Evaluation. This step receives (1) data structures (depending on the model) from the training step, and (2) a test set of articles. The data for each test article is used with the data structures to make label predictions, which are then evaluated against the true labels provided in the test set. Each attribute-model combination is given an accuracy score, and the best combination is selected for each website. The table below shows the optimal accuracy scores (compared to the baseline of guessing all 0's) for the five most frequent websites.

Website	Baseline	Optimal	Method
factcheck.afp.com	0.7333	0.7556	3-gram, Bayes
politifact.com	0.6316	0.6316	Baseline
maldita.es	1.0000	1.0000	Baseline
newtral.es	0.8571	1.0000	1-gram, Switch
cekfakta.tempo.co	0.9091	0.9091	Baseline
factuel.afp.com	0.8479	0.8696	3-gram, Bayes
boomlive.in	0.9333	1.0000	3-gram, Switch

Prediction & Kaggle Contest Results. After determining the most accurate methods for each website, the unlabeled articles in the real test dataset are classified using these optimal methods. One important observation is that for the majority of the websites, the baseline method of guessing 0 is the most accurate method; this means that for many websites, there is no underlying structure in the HTML that determines the label of each article. Rather, in many cases, the label is determined by the meaning of the content, which cannot be easily represented by a bag of tokens. Nevertheless, this learning algorithm performed better than merely guessing 0s; I ran the algorithm three times (with different random seeds) and obtained the following results from Kaggle:

	Trial #1	Trial #2	Trial #3	Baseline
Mean F1 Score	0.51594	0.51730	0.50768	0.52137

The best result, though submitted after the deadline, would have ranked ***3rd*** on the leaderboard. This really underscores how difficult it is to make predictions on a heavily imbalanced dataset.


Your most recent submission

Name	Submitted	Wait time	Execution time	Score
preds3.csv	15 hours ago	1 seconds	0 seconds	0.50768

Complete

Jump to your position on the leaderboard

6. Future Works

In this section, I discuss ideas that can be further addressed in future works. The following three topics are noteworthy shortcomings to my solution and could be studied further in order to potentially improve it.

Generalizing to unforeseen websites. This algorithm takes advantage of the overlap in websites between the train and test sets. However, if the test set primarily consists of articles from unforeseen websites, this algorithm would be rendered useless. Therefore, a more effective algorithm for this task would have the ability to classify articles independent of its website.

Accounting for limitations of *N*-grams. Although *N*-grams were introduced to preserve meaning (at least to some extent) in the article content itself, for most websites this approach failed to find patterns that could predict the labeling more accurately than the baseline method. Extracting meaning from a body of text is a notable issue in the field of Natural Language Processing.

Limiting the variability across random seeds. I found that the three trials had (pretty significantly) different results for the optimal methods. More data may resolve this shortcoming, but in the future, it would be beneficial to robustify the two text classification models under varied train-test splits.

7. References

[1]
https://towardsdatascience.com/text-classification-using-naive-bayes-theory-a-working-example-2ef4b7eb7d5a

[2]
https://towardsdatascience.com/natural-language-processing-feature-engineering-using-tf-idf-e8b9d00e7e76

[3]
https://beautiful-soup-4.readthedocs.io/en/latest/

[4]
https://towardsdatascience.com/understanding-word-n-grams-and-n-gram-probability-in-natural-language-processing-9d9eef0fa058