Naive Bayes Case Study
Craigslist Apartment Hunting

Ethan Eldridge

***Abstract*—In this paper, we show a case study of the use of a Naive Bayes classifier acting on data crawled from the popular website www.craigslist.com. Craigslist provides local classifieds and forums for jobs, housing, for sale, personals, services, local community, and events. Using a custom web-crawling application developed in Ruby by the author, we gather text data from the apartment section of Craigslist for the Burlington area and then classify each posting into what city it falls into based on the text in the title of the Craigslist advertisement. Using the same data, we attempt to use the title of an advertisement to determine which price bracket it falls into. We use an open source Naive Bayes classifier from https://github.com/alexandru/stuff-classifier to perform the classification. Comparing the typical Naive Bayes to the statistical Tf-Idf (term frequency–inverse document frequency). We find that the results indicate that title alone is not a good indicator of what the price range of an advertisement is. Also, that Naive Bayes performs slightly better on Tf-Idf based solely on classification correctness. In order to attain better results, we further crawl the webspace and incorporate the longer textual description of each listing into the classification process. Doing so produces no change in the behavior of either classifier's output.**

Contents

I. Introduction

College students are often faced with the problem of finding cheap affordable housing. One of the many resources students turn to is the popular website www.craigslist.com. Craigslist provides landlords, subletters, and seekers-of-roommates the ability to post advertisements detailing a living situation. However, one of the most difficult tasks when faced with lists of apartments is to determine if the descriptions of the listings are worth the amount they are posted for. In addition to price, often times the search feature of Craigslist is not specific amount to bring back results for a specific area. For example, attempting to find an apartment in Montpellier woudl prompt one to perform a search with the key *Montpellier*, but this often returns results from Montpellier to locations in New Hampshire.

II. A Brief Review of Naive Bayes

Simply put, a Naive Bayes classifier uses the presence of features in data to determine the class of any particular instance. Given training data, the classifier correlates the presence of features towards the class on which the instance it is training on belongs. After training on some amount of data, the classifier is able to generalize any particular instance of data it sees using the frequency of the features it has observed within the training set. The Naive

Bayes assumes that each feature is independent of any other, and usings Bayes Theorem (equation 1)

$$p(C|F_1, \ldots, F_n) = \frac{p(C)p(F_1, \ldots, F_n|C)}{p(F_1, \ldots, F_n)} \quad (1)$$

determines the most likely class for an instance given it's features.

In some regards, the proper classification depends on the domain of the data itself. With discrete and finite number of features in any data sample, it is likely that the classifier will be able to perform the classification correctly after viewing enough training information. With some supervised learning, new instances of each class can be used to further train the classifier resulting in better estimates of the features that determine a class. However, for a data set such as the english language it is often difficult to assess; as the number of features is intractably large (the entire english language). Efforts can be made to reduce the complexity of this dataset through lemmatization[1], however in examples I have seen, Naive Bayes performs well dealing with smaller subsets of language, such as spam filters and detectors. This case study intends to try to use the particular subset of the english language which associates with apartment hunting jargon and find indication of value in these words.

III. The Crawler and Web Structure

Information about apartments can be found at http://burlington.craigslist.org/apa/, upon inspecting this page we can see the structure of the data itself. All postings are location within the html tag blockqoute with an id attribute of *toc_rows*. Nested inside this element are two types of data. Header tags specifying the date of the postings which can be ignored, and the advertisements themselves. Each advertisement is stored in paragraph tags with the class of *row*, within these tags the location of an apartment is within small tags and parenthesis. The title of each posting is located between the opening and closing anchor tags. Since there is no unneccesary nesting of tags within each other, the html can be parsed by regular expressions and then written out to data files during the crawl. In the second part of this case study, we parse the price out of the html by searching for the span tags with the class of *itemprice*.

[1]The process of reducing any complex word to its stem or root

Classifier	Correct	Incorrect	Trained On
Naive Bayes	104	1352	485
Tf-Idf	0	1456	485

Figure 1. Initial Results. No Cleaning

In order to collect enough data, the index page located at http://burlington.craigslist.org/apa/ does not suffice. However, on each page is a hyperlink to the next 100 postings. Inspecting this link results in determining the format of the url's craigslist uses to store the older information. Each 100 postings after the initial index page are stored on pages with the following format: http://burlington.craigslist.org/apa/index(pagenumber)00.html where page number is any number equal to or higher than 1. By noting this structured form of not only the data but the url's themselves we create a crawling program that retrieves an arbitrary number of pages, and using regular expressions, collects relevant text into data files.

IV. Location Classification

A. Initial Results without data cleaning

After collecting 1456 samples[2], a total of 811 distinct classes were found among the locations. As can be seen in figure 1 attempts to classify the data with either method resulted in less than 10% of the data being correctly classified after training on a third of the dataset. This was undoubtedly due to the large amount of locations generated by the automatic crawling and parsing. Using half of the training data instead of a third resulted in no change for either Tf-Idf or Naive Bayes, this implied that the few correct that Naive Bayes had classified were most likely correct due to chance and nothing more. The automatic generation of class labels through the text from only craigslist results in far too specific class labels being created. For example, if an advertisement lists a location of 132 North St, South Burlington this will be an entirely new class, even if the class South Burlington exists as well. Obviously, as humans we know that both posting could be classified as being located in South Burlington, and in fact, it would be beneficial to clean our data in such a way that the number of

[2]Crawling 15 pages and discarding any badly formatted data results in a lower than the expected 1500 samples

Classifier	Correct	Incorrect	Trained On
Naive Bayes	203	1226	485
Naive Bayes	231	1225	728
Tf-Idf	0	1456	485
Tf-Idf	0	1456	728

Figure 2. Results after Cleaning

distinct classes are narrowed down from 811, to a much smaller number.

B. Data Cleaning

In order to clean the data, one must map addresses to towns. This is a tedious task for a human to do, but not one for Google's Geocode API. Using the geocode API one has access to the powerful processing power of maps.google.com. A request is simple to perform, and by parsing the output for the locality of the google-resolved address, one can narrow down the list of classes from 811 to 503. While this is still a large number, an immediate improvement in the classification of the data occurred. As seen in figure 2 Naive Bayes saw a doubling of its correctly classified output but Tf-Idf still failed to classify any. The number of classes was still too high to be able to sort each posting into the correct location.

To bring the number of classes down to a more manageable we change the scope of our API request. Throughout the first round of cleaning, we used the locality specifier for the geological scope of our request. To narrow the number of classes down farther, we used the *administrative_area_level_2* location type from the returned results of the Google API request. This caused the number of classes to drop to 390. However, despite this there was no significant difference in results from figure 2. Upon inspection of the output, it was noted that since the administrative_area_level_2 locale was at the county level, the scope was too broad to use just a title to classify a posting[3].

C. Conclusions on Location Classification

With regards to classifying location, it seems silly to even try to use something as small and noisy as an apartment posting title on craigslist to determine the location of the apartment. However, it was worth an effort to see if the Naive Bayes or Tf-Idf could pick up on some type of pattern that a human could not. Maybe given more training data it might have found some type of pattern. We believe that a smaller set of possible classes would have been highly beneficial as well. Because all data cleaning was automatic the number of classes was not successfully reduced down to just the number of counties as I had hoped it would be. Manually editing the data pulled from the internet seems like the only course of action that would result in a reduced set of classes which might work well. However, given that the title text for an advertisement is very small and often repeated between different locations it simply doesn't seem like it's possible to classify an apartments location by it's craigslist posting.

[3]The Naive Bayes only classified 200 samples correctly because it simply guessed Chittenden county the entire time, while this seems silly, it is actually a good strategy since most training samples it saw were probably from the Chittenden area

Classifier	Correct	Wrong
Naive Bayes	5130	2997
Tf-Tdf	0	8127

Figure 3. The results of training on ≈1600 of 8127 samples

V. Price Classification

A. Price Brackets

In order to classify a posting from its title to its price range it is first necessary to determine the price ranges themselves. For this experiment we first collected 8127 samples of title and price tuples, in order to determine a simple price bracket of high or low, the average of all the prices were computed and the closet hundred was used as the cutoff point for high or low. This average ended up being ≈1030. Figure 3 shows the results of training on a fifth of the data and validating on the rest. [4]

B. Data Cleaning

The price data did not suffer from the same issues as the location data, where I had too many classes. Rather, the price data suffers from a distribution issue. While the average of the prices does stay close to 1000, this is because most apartment prices are between 700-1400. While you would assume the

[4]Tf-Tdf always returned the default class, which was configurable, so not much insight can be seen here.

Classifier	Correct	Wrong
Naive Bayes	5994	5130
Tf-Tdf	5130	5994

Figure 4. The results of training on ≈2000 of 11124 samples

average would be higher because of this, by pulling data not only from apartments, but also subletting advertisements[5] we skew our dataset more towards the lower half. In order to fix this two strategies were attempted. Duplicating samples of the class which was under-represented, and collecting more data in hopes of finding more expensive apartments that would help the distribution be more uniform.

Duplicating the data was easy to do without actually modifying the data crawled from craigslist. Since the distribution of high and low was about 63% to 37%, by simply duplicating samples of class high, the dataset became 6404:7510. Again, training on a fifth of these and then validating on the remaining resulted in the classifiers always guessing a single class, whichever one was most frequent. As seen in figure 4, Naive Bayes always chooses the most frequent class to use as its default choice and Tf-Tdf uses whichever default choice we specify when calling its classification method. Crawling more craigslist postings that have higher costs is as simple as pointing the crawler towards the apartment section, then only writing the expensive listings to the data file being used during classification. By doing this we boosted our sample size up to 14911 with 7504 high classes and 7407 low classes. Training proceeded as usual, with a fifth of the training data being used. Surprisingly, Naive Bayes performed worse than Tf-Tdf during this run, as it consistently guessed the high class but the validation training set only had 5912 such samples. Adding more data into the mix did help the classifier switch from always guessing low to always guessing low. But did not have the desired result of giving more title information to actually train on.

C. Conclusions on Price Classification

As with location classification, it seems like there is simply not enough valuable data in a craigslist job posting title to correctly classify the price range of the apartment. The results of Tf-Tdf having difficulty may be a result of this, as Tf-Tdf essentially determines word importance, there simply may not be enough common words between posting to truly determine the worth of a listing. This similarly applies to Naive Bayes as the large variety of the text results in a rather low probability once the evidence is determined for a given title. It may be the case that there is simply too much data. In order to alleviate this, word stemming was used through built in ruby library that was integrated into the StuffClassifier itself. However, either the data itself is too noisy for this to help or once again, the diversity of the different words is too vast for the methods of frequency counting to really assist in classification.

VI. Deep Crawling

A. Data Acquisition

Since the title for each posting simply does not supply enough data, after the results of sections IV and V were collected, instead of just crawling the index pages that contained only the title of the information, a new crawl was made which used the links of the top level pages to fetch each advertisements longer description data. This increased the running time of each crawl, but the resulting data was more descriptive, and intuitively would increase the ability of both classifiers to determine what price range each listing was in[6]. As noted before, the data on craigslist pages is highly structured, and aquiring the links from the text of the page was not hard. One simple regular expression was able to find all possible links on the page and return the results. By integrating this pattern matching into the title and price collection process, each advertisement object was constructed all at once and could be stored for later use. Unfortunately, as noted before, the data acquisition stage of the experiment increased significantly. From simply crawling 100 or more html pages to 10,000[7], this time increase slowed down research significantly – and occasionally resulted in connection resets by the servers on craigslist. In addition to aquiring more text per sample to work with, during pre-processing of the data certain stopwords were removed. Stop words are just common english words that don't add any valuable contextual

[5]Which tend to have a price range between 400-1000

[6]Deep crawls were not made and tested for location classification because no way to trim down the number of classes without tedious manual data cleaning could be found.

[7]There are 100 links per craigslist post

Classifier	Correct	Wrong	Stop Words
Naive Bayes	1067	667	Removed
Naive Bayes	1067	667	Left In
Tf-Idf	667	1067	Removed
Tf-Idf	667	1067	Left In

Figure 5. Results of Classifiers on Data. Tf-Idf always returned the default class while Naive bayes returned the class with the most examples.

meaning for a sample. Examples of such words are *the, at, am* and *and*. By removing these words we further enhance the differences between possible samples in the hopes that the specific words that might indicate a certain price range will rise to the surface and the classifier will be able to pick up on these key phrases.

B. Results from deep Crawling

With a sample size of 2601 total samples, 1000 belonging to the high price range, and 1601 belonging to the lower price range. Each classifier was given the dataset with and without the stop words removed, Then trained on a third of the data set and given the remaining samples to validate on. Unfortunately, with stop words or not, it seems the data was simply too much to handle. As with previous attempts at classification, Tf-Idf returned only the default class, and Naive Bayes guessed whichever class had the most examples. Noted previously, while this is a good strategy when dealing with hard to classify data[8], it suggests that Naive Bayes simply cannot handle the large diversity of the search space. By having so many words factored into its analysis and so many variables, we find that it is difficult for the classifier to return anything but the default class. It seems that both classifiers had difficulty, and intuitively it makes sense that the sheer volume of the search space would be overwhelming to something as simple as frequency counting.

VII. Future Work

In future work it would be interesting to try to increase the volum of data collected by the deep crawling process. Even though this seemed like the root of the problem for the classifiers, if there was enough data that the words that really mattered to the classification could become frequent enough to overcome the random words shared between classes, this might work. Time constraints and issues with connections being reset by the servers on craigslist made it difficult to collect as much data as we would've liked. So collecting enough data to find those words that might have helped each classifier never occurred, and in future work with this I would definitely spend more time collecting and sanitizing the data. Also, the use of stop words and other cleaning techniques on raw textual data could be employed to try to reduce the variety of words each classifier had to handle. It seemed that the more data passed to the classifiers, the more likely they were to simply resort to guessing whatever class was more frequently. If there could have been a way to study and remove words that were very common to all postings we believe it would have helped[9] classification to be more than just guessing.

VIII. Conclusions

In conclusions, Naive Bayes and Tf-Idf do not perform well on the noisy data of craigslist apartment and rooms-for-rent listings. The data itself is too difficult for the simple classifiers to handle; there may also not be any correlation between what is in a title and description of an advertisement and its price. While the author assumed that the correlation existed because a Human can look at text and images and see value in it, it is not unlikely that a computer can do the same. Performance of each algorithm was fast as expected, however no real valuable results were found. Defaulting to a single guess for the entire dataset –while minimizing error– did not show any improvement for either algorithm given more or less training data. Finally, the author would like to thank Eebs Kobeissi for showing him the Ruby implementation of Naive Bayes by the github user alexandru. Also, thanks to Craigslist for not blacklisting my IP while I crawled their website over 100,000 times.

[8]Getting most of them right with some consistent probability is more desirable than random guessing.

[9]There was an attempt to do this by using the Stop word removal, but it wasn't a complete list.