IMT574 Problem Set 5: Naïve Bayes

February 12, 2020

Introduction

This problem set has two aims: a) learn to understand Naive Bayes, and b) learn to handle text, a form of data that does not come as a table of numbers. You will implement your own Naive Bayes classifier and use this for categorizing Rotten Tomatoes reviews into rotten and fresh ones. Finally, you also find the optimal smoothing parameter.

Please submit a) your code (notebooks, rmd, whatever) *and* b) the results in a final output form (html or pdf).

Note that this is groupwork, you should find 2-3 student groups (or wait until we assign you to one).

Rotten Tomatoes

Our first task is to load, clean and explore the Rotten Tomatoes movie reviews data. Please familiarize yourself a little bit with the webpage. Briefly, approved critics can write reviews for movies, and evaluate the movie as "fresh" or "rotten". The webpage normally shows a short "quote" from each critic, and whether it evaluates the movie as fresh or rotten. You will work on these quotes below.

The central variables for our purpose in *rotten-tomatoes.csv* are the following:

fresh evaluation: 'fresh' or 'rotten'

quote short version of the review

There are more variables like links to IMDB.

1 Explore and clean the data

First, let's load data and take a closer look at it.

1. Take a look at a few lines of data (you may use `pd.sample` for this).
2. print out all variable names.
3. create a summary table (maybe more like a bullet list) where you print out the most important summary statistics for the most interesting variables. The most interesting facts you should present should include: a) number of missings for *fresh* and *quote*; b) all different values for fresh/rotten evaluations; c) counts or percentages of these values; d) number of zero-length or only whitespace *quote*-s; e) minimum-maximum-average length of quotes (either in words, or in characters). (Can you do this as an one-liner?); f) how many reviews are in data multiple times. Feel free to add more figures you consider relevant.

4. Now when you have an overview what you have in data, clean it by removing all the inconsistencies the table reveals. We have to ensure that the central variables, *quote* and *fresh* are not missing, *quote* is not an empty string (or just contain spaces and such), and all rows are unique.

   I recommend to do it as a standalone function so you can use the same function for another similar dataset (such as test data).

2 Naïve Bayes

Now where you are familiar with the data, it's time to get serious and implement the Naive Bayes classifier from scratch. But first things first.

1. Ensure you are familiar with Naive Bayes. Consult the readings, available on canvas. Schutt & O'Neill is an easy and accessible (and long) introduction, Whitten & Frank is a lot shorter but still accessible introduction. The Lecture notes contains examples how to create baf-of-words (BOW), and how to compute Naive Bayes classifier using BOW-s.

2. Convert your data (quotes) into bag-of-words. Your code should look something like this:

```
from sklearn.feature_extraction.text import CountVectorizer
vectorizer = CountVectorizer(binary=True)
# define vectorizer
X = vectorizer.fit_transform(cwork.quote.values)
# vectorize your data.  Note: this creates a sparce matrix,
# use .toarray() if you want a dense matrix.
words = vectorizer.get_feature_names()
# in case you want to see what are the actual words
```

   `binary=True` specifies that we don't want BOW-s that contains counts of words but just 1/0 for the presence/non-presence of the words.

3. Split your work data and target (i.e. the variable *fresh*) into training and validation chunks (80/20 or so).

Good. Now you are ready with the preparatory work and it's time to dive into the real thing. Let's implement Naive Bayes. Use only training data when doing the fitting below.

4. Compute the unconditional (log) probability that the tomato is fresh/rotten, $\log \Pr(\mathsf{F})$, and $\log \Pr(\mathsf{R})$. These probabilities are based on the values of *fresh* alone, not on the words the quotes contain.

5. For each word w, compute $\log \Pr(w|\mathsf{F})$ and $\log \Pr(w|\mathsf{R})$, the (log) probability that the word is present in a fresh/rotten review. These probabilities can easily be calculated from counts of how many times these words are present for each class.

   Hint: these computations are based on your BOW-s X. Look at ways to sum along columns in this matrix.

Now we are done with the estimator. Your fitted model is completely described by these four probability vectors: $\log \Pr(\mathsf{F}), \log \Pr(\mathsf{R}), \log \Pr(w|\mathsf{F}), \log \Pr(w|\mathsf{R})$. Let's now turn to prediction, and pull out your validation data (not the test data!).

6. For both destination classes, F and R, compute the log-likelihood that the quote belongs to this class. *log-likelihood* is what is given inside the brackets in equation (1) on slide 28, and the equations on Schutt "Doing Data Science", page 102. In lecture notes it is explained before the email classification example (and in the example too). On the slides we have the log-likelihood essentially as (although we do not write it out):

$$\ell_i(c) = \log \Pr(c) + \sum_j \log \Pr(w_{ij}|c)$$

where $c \in \{\mathsf{F}, \mathsf{R}\}$ is the class, i is the review, j indexes words, and w_{ij} is the j-th word of the review i.

Computing these likelihoods involves sums of the previously computed probabilities, $\log \Pr(w|\mathsf{F})$, and BOW elements x_{ij}. Check out `np.apply_along_axis` that can be used to apply a function on matrix columns/rows so you can create a fairly good one-liner to compute log-likelihood. Loops are fine too if *apply* seems too complex, just slower and less compact.

Based on the log-likelihoods, predict the class F or R for each quote in the validation set.

7. Print the resulting confusion matrix and accuracy (feel free to use existing libraries).

3 Interpretation

Now it is time to look at your fitted model a little bit closer. NB model probabilities are rather easy to understand and interpret. The task here is to find the best words to predict a fresh, and a rotten review. And we only want to look at words that are reasonably frequent, say more frequent than 30 times in the data.

1. Extract from your conditional probability vectors $\log \Pr(w|\mathsf{F})$ and $\log \Pr(w|\mathsf{R})$ the probabilities that correspond to frequent words only.

2. Find 10 best words to predict F and 10 best words to predict R. Hint: imagine we have a review that contains just a single word. Which word will give the highest weight to the probability the review is fresh? Which one to the likelihood it is rotten?

   Comment your results.

3. Print out a few missclassified quotes. Can you understand why these are misclassified?

4 NB with smoothing

So, now you have your brand-new NB algorithm up and running. As a next step, we add smoothing to it. As our task is to find the best smoothing parameter below, your first task is to mold what you did above into two funcions: one for fitting and another one for predicting.

1. Create two functions: one for fitting NB model, and another to predict outcome based on the fitted model.

   As mentioned above, the model is fully described with 4 probabilities, so your fitting function may return such a list as the model; and the prediction function may take it as an input.

2. Add smoothing to the model. See Schutt p 103 and 109. Smoothing amounts to assuming that we have "seen" every possible word $\alpha \geqslant 0$ times already, for both classes. (If you wish, you can also assume you have seen the words α times for F and β times for R). Note that α does not have to be an integer, and typically the best $\alpha < 1$.

3. Cross-validate the accuracy (on the validation data) on a number of α values and find the α that gives you the best result. You can use your own CV algorithm you created for PS4, or an existing library.