CS158 - Assignment 7

Naive Bayes or Intelligent Bayes?

Part A – Due: Sunday, October 19th by 11:59pm
Part B – Due: Sunday, October 26th by 11:59pm

https://www.smbc-comics.com/comic/bayesian

For this assignment we're going to be adding another multi-class classifier to our collection, the Naive Bayes classifier. In addition, we're going to play with generating ROC curves. You may (and I would strongly encourage you to) work with a partner on this assignment.

Starter

Follow the same process you have with the previous assignments:

`https://classroom.github.com/a/0vXIX7tl`

(There isn't anything new in the starter code for this time, though there is a small dataset we'll use to check our calculations.)

Part A: Getting your hands dirty

I've put together a simple test data set that's useful for doing *some* checking if you're on the right path. To get you started, I'm asking you to manually calculate the probabilities for this data set. This will make sure you understand what you're doing and will also be useful for testing the correctness of your code.

0. Read through the rest of this document (in particular, the "Training" portion of the Specifications section) to make sure you understand how to calculate probabilities, etc.

1. Probabilities by hand

   (a) In the starter, there is a data directory with a file call `simple.data`. There are 6 documents (3 negative and 3 positive). Manually calculate the Naive Bayes classifier probabilities listed below without smoothing and put these in a text file (note these are *all* of the probabilities your model would calculate):

   ```
   p(positive) =
   p(negative) =
   p( I | positive ) =
   p( hated | positive ) =
   p( that | positive ) =
   p( movie | positive ) =
   p( loved | positive ) =
   p( it | positive ) =
   p( I | negative ) =
   p( hated | negative ) =
   p( that | negative ) =
   p( movie | negative ) =
   p( loved | negative ) =
   p( it | negative ) =
   ```

   (b) Calculate manually what the probability "score" (i.e. $p(\texttt{features}|\texttt{label})p(\texttt{label})$) of the following sentence would be for both the positive class and the negative class using a) the "correct" approach that includes all features and b) the positive features-only approach:

   ```
   I loved it
   ```

   (You should have four probabilities, i.e., $p(features, label)$ for the two labels for the two approaches.)

2. Probabilities with smoothing

   Repeat all of the calculations for Problem 1 with $\lambda = 1$.

Put your answers in a pdf file and submit via gradescope under **Assignment 7a**.

I will post the correct probabilities on Monday/Tuesday so that you can use them to check your answers. Once you get training and classifying working, you can train your system on this "simple" data, print out the counts/probabilities and then also test the probability of your sentence and make sure that works as well.

I know this may feel a bit tedious but I promise you having an example that you understand and can use for testing will help save you time next week.

Part B: Ready, set, implement!

Specifications

Implement a class called `NBClassifier` that implements the `Classifier` interface. Your NB classifier should treat each feature as a binary feature, that is your probability distributions $p(x_i|y)$ are only over two values, *positive* and *negative*. We will treat any feature that is both available AND non-zero as *positive* and any feature that is either not available or zero as *negative*.

Your class should:

- have a zero parameter constructor
- include a `setLambda` function that allows you to set the λ regularization/smoothing parameter
- support two different ways of calculating the probability of examples. The mathematically correct way of implementing Naive Bayes is to calculate $p(x_1, x_2, ..., x_m, y)$ as:

$$p(x_1, x_2, ..., x_m, y) = p(y) \prod_{i=1}^{m} p(x_i|y)$$

  Specifically, if a features is present, then you multiply by $p(x_i = \text{positive } |y)$, and, if it is not present, then you multiply by $p(x_i = \text{negative } |y) = 1 - p(x_i = \text{positive } |y)$.

  In some domains, however, if the examples are very sparse (e.g. text) this can be very expensive and, in some cases, non-intuitive. Another variant that is often considered in these domains is calculating $p(x_1, x_2, ..., x_m, y)$ using only the features that are present/positive:

$$p(x_1, x_2, ..., x_m, y) = p(y) \prod_{i=1}^{m} \begin{cases} p(x_i|y) & \text{if } x_i > 0 \\ 1 & \text{otherwise} \end{cases}$$

  i.e., we include the probability of the feature occurring for each feature that occurs in the example.

Your implementation should support both ways of classifying an example. A few things to note:

1) the training method and method for calculating individual feature probabilities is exactly the same between these two approaches and
2) when only considering positive features, for efficiency reasons, you should only iterate over the features available for the example and NOT over all features. Notice that the `DataSet` class has a function that tells you all the features that are available and the `Example` class has a function that tells you all features available for a given example.

- include a `setUseOnlyPositiveFeatures` function that takes a `boolean` and chooses between the different classification variants just described. If it is set to `true` then is should only use the positive features. By default, this should be set to `false`, that is to use all features.

- include the following functions:

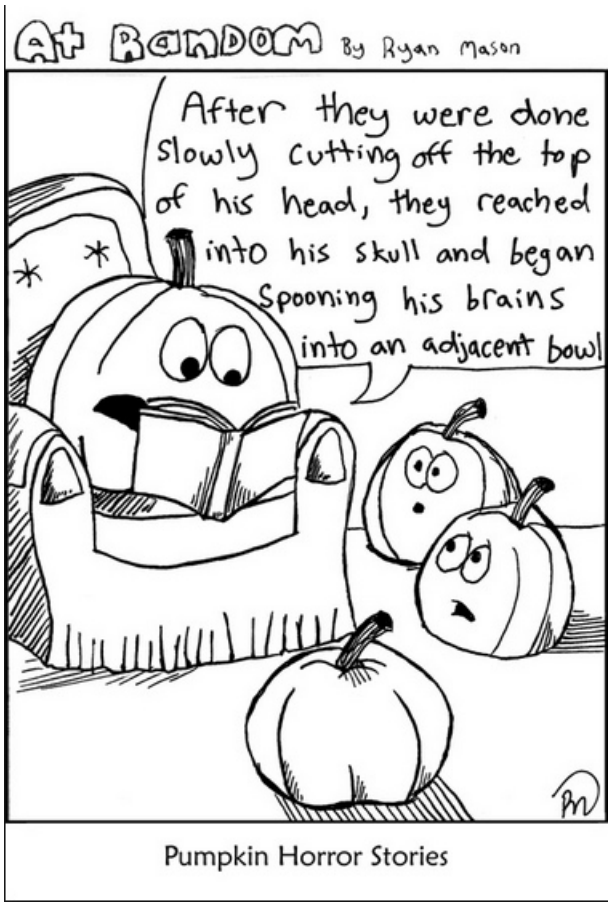
```
public double getLogProb(Example ex, double label)
public double getFeatureProb(int featureIndex, double label)
```

The first should return the log (*base 10*) probability of the example with the label under the current trained model, i.e. $p(x_1, x_2, ..., x_m, y)$, and the second should give $p(x_i|y)$ for the model. These will help me test your model's correctness.

- the `confidence` method should return the log probability of the most likely label (i.e. the label chosen for that example).

- the `train` method should retrain the model from scratch each time, so make sure that all of your data gets wiped clean. This will allow you to train the same classifier multiple times on different data.

- all logs should be base 10



http://www.atrandomcomics.com/october-2015/

Hints/Advice

- When dealing with multiplying probabilities, you should almost always use log probabilities and not probabilities since underflow often becomes a problem. Specifically, instead of classifying as:

$$label = argmax_{y \in labels}\ p(x_1, x_2, ..., x_m, y)$$

you should use:

$$label = argmax_{y \in labels}\ \log(p(x_1, x_2, ..., x_m, y))$$

which for the all features setting then becomes:

$$label = argmax_{y \in labels}\ log(p(y)) + \sum_{i=1}^{m} log(p(x_i|y))$$

- You should NOT regularize/smooth the label probabilities $p(y)$
- Make sure you understand that for this NB model we are learning *many* different probability distributions (one for each feature, plus one for the labels). Each of these distributions should be proper probability distributions that sum to one. Make sure when you smooth that you make this happen correctly, i.e. that you understand the denominator of:

$$\hat{p}(x_i|y) = \frac{\text{count}(x_i, y) + \lambda}{\text{count}(y) + \text{possible\_values\_of\_}x_i * \lambda}$$

- In the `ml.utils` package there are a couple of classes that might be useful for this assignment. In particular, take a look at the `HashMapCounter` class.
- During training you can either just store the raw counts or you can collect the counts and then calculate and store the probabilities. There are advantages to either approach. Most often people choose the first (storing raw counts) for two reasons: 1) you can change the λ value without having to retrain and 2) the model can be stored as integers instead of doubles, creating some memory savings. That said, storing the probabilities does save us some computation during classification. Either implementation is fine for this assignment.
- You should be able to train the model with a *single pass* through the training data, possibly with a second loop over your model to calculate the probabilities if you decide not to calculate them on the fly. If you do it right, the train method should be very fast.

Experiments

In a file called `experiments` (pick some reasonable file type) include the analysis below (explicitly number each part in your writeup). For all the experiments use the *wine data set*.

1. What is the optimal lambda for the NB classifier using all features? Show your experimental data for how you found this.
2. What is the optimal lambda for the NB classifier using only positive features? Show your experimental data for how you found this.
3. Which version of the NB classifier is better (all features vs. positive features). Include one or two sentences that describe how you came to this conclusion. If you use a performance argument, make sure to use statistical tests to justify your answer.
4. One of the uses of classifier confidence is to allow you to improve the accuracy of the classifier by only labeling those examples above a given confidence threshold. If your confidence measure is a good predictor, you can get an increase in accuracy with only a small number of examples not getting labeled.

   To explore this, create a graph that plots the accuracies on the y-axis versus the confidence thresholds on the x-axis based on the first split of the data using a 10-fold cv split. To do this you'll need to:

(a) train the NB classifier

(b) for each example, get the label prediction and the confidence for that prediction

(c) sort these pairs in decreasing order by confidence

(d) working from the most confident to the least, calculate the accuracy if you placed the confidence threshold at the current example. If your data is sorted by confidence, this should be easy to do in a single pass over the data by keeping track of how many you've seen already that are correct. This will give you a (accuracy,confidence cutoff) point for each test example which you can then plot.

1 When You're Done

Make sure that your code compiles, that your files are named as specified and that you have followed the specifications exactly (i.e. method names, number of parameters, etc.).

Your `experiments` file with the answers, etc. should also be added to your repo.

Submit your assignment on gradescope by providing the url for your github repo. **Make sure that you have pushed the latest version of your code, etc. to the repo before you submit.** Note that you may submit as many times as you want up until the assignment is due.

Commenting and code style

Your code should be commented appropriately (though you don't need to go overboard). The most important things:

- Your name (or names) and the assignment number should be at the top of each file
- Each class and method should have an appropriate doctsring
- If anything is complicated, it should include some comments.

There are many possible ways to approach this problem, which makes code style and comments very important here so that I can understand what you did. For this reason, you will lose points for poorly commented or poorly organized code.