10-601 Machine Learning: Homework 3

Due 5 p.m. Wednesday, February 4, 2015

Instructions

- **Late homework policy:** Homework is worth full credit if submitted before the due date, half credit during the next 48 hours, and zero credit after that. You *must* turn in at least $n-1$ of the n homeworks to pass the class, even if for zero credit.

- **Collaboration policy:** Homeworks must be done individually, except where otherwise noted in the assignments. "Individually" means each student must hand in their own answers, and each student must write and use their own code in the programming parts of the assignment. It is acceptable for students to collaborate in figuring out answers and to help each other solve the problems, though you must in the end write up your own solutions individually, and you must list the names of students you discussed this with. We will be assuming that, as participants in a graduate course, you will be taking the responsibility to make sure you personally understand the solution to any work arising from such collaboration.

- **Online submission:** You must submit your solutions online on autolab. We recommend that you use LaTeX to type your solutions to the written questions, but we will accept scanned solutions as well. On the Homework 3 autolab page, you can download the template, which is a tar archive containing a blank placeholder pdf for the written question and one Octave source file for each of the programming questions. Replace the pdf file with one that contains your solutions to the written questions (g), (h), and (i) (include all parts in the same pdf) and fill in each of the Octave source files. When you are ready to submit, create a new tar archive of the top-level directory and submit your archived solutions online by clicking the "Submit File" button. You should submit a single tar archive identical to the template, except with each of the Octave source files filled in and with the blank pdf replaced by your solutions for the written questions. You are free to submit as many times as you like (which is useful since you can see the autograder feedback immediately).

  DO NOT change the name of any of the files or folders in the submission template. In other words, your submitted files should have exactly the same names as those in the submission template. Do not modify the directory structure.

Problem 1: Implementing Naive Bayes

In this question you will implement a Naive Bayes classifier for a text classification problem. You will be given a collection of text articles, each coming from either the serious European magazine *The Economist*, or from the not-so-serious American magazine *The Onion*. The goal is to learn a classifier that can distinguish between articles from each magazine.

We have pre-processed the articles so that they are easier to use in your experiments. We extracted the set of all words that occur in any of the articles. This set is called the *vocabulary* and we let V be the number of words in the vocabulary. For each article, we produced a feature vector $X = \langle X_1, \dots, X_V \rangle$, where X_i is equal to 1 if the i^{th} word appears in the article and 0 otherwise. Each article is also accompanied by a class label of either 1 for The Economist or 2 for The Onion. Later in the question we give instructions for loading this data into Octave.

When we apply the Naive Bayes classification algorithm, we make two assumptions about the data: first, we assume that our data is drawn iid from a joint probability distribution over the possible feature vectors

X and the corresponding class labels Y; second, we assume for each pair of features X_i and X_j with $i \neq j$ that X_i is conditionally independent of X_j given the class label Y (this is the Naive Bayes assumption). Under these assumptions, a natural classification rule is as follows: Given a new input X, predict the most probable class label $\hat{Y}$ given X. Formally,

$$\hat{Y} = \underset{y}{\operatorname{argmax}}\, P(Y = y|X).$$

Using Bayes Rule and the Naive Bayes assumption, we can rewrite this classification rule as follows:

$$\begin{aligned}
\hat{Y} &= \underset{y}{\operatorname{argmax}}\, \frac{P(X|Y = y)P(Y = y)}{P(X)} && \text{(Bayes Rule)} \\
&= \underset{y}{\operatorname{argmax}}\, P(X|Y = y)P(Y = y) && \text{(Denominator does not depend on } y\text{)} \\
&= \underset{y}{\operatorname{argmax}}\, P(X_1, \ldots, X_V|Y = y)P(Y = y) \\
&= \underset{y}{\operatorname{argmax}} \left(\prod_{w=1}^{V} P(X_w|Y = y) \right) P(Y = y) && \text{(Conditional independence).}
\end{aligned}$$

The advantage of the Naive Bayes assumption is that it allows us to represent the distribution $P(X|Y = y)$ using many fewer parameters than would otherwise be possible. Specifically, since all the random variables are binary, we only need one parameter to represent the distribution of X_w given Y for each $w \in \{1, \ldots, V\}$ and $y \in \{1, 2\}$. This gives a total of $2V$ parameters. On the other hand, without the Naive Bayes assumption, it is not possible to factor the probability as above, and therefore we need one parameter for all but one of the 2^V possible feature vectors X and each class label $y \in \{1, 2\}$. This gives a total of $2(2^V - 1)$ parameters. The vocabulary for our data has $V \approx 26,000$ words. Under the Naive Bayes assumption, we require on the order of $52,000$ parameters, while without it we need more than 10^{7000}!

Of course, since we don't know the true joint distribution over feature vectors X and class labels Y, we need to estimate the probabilities $P(X|Y = y)$ and $P(Y = y)$ from the training data. For each word index $w \in \{1, \ldots, V\}$ and class label $y \in \{1, 2\}$, the distribution of X_w given $Y = y$ is a Bernoulli distribution with parameter θ_{yw}. In other words, there is some unknown number θ_{yw} such that

$$P(X_w = 1|Y = y) = \theta_{yw} \quad \text{and} \quad P(X_w = 0|Y = y) = 1 - \theta_{yw}.$$

We believe that there is a non-zero (but maybe very small) probability that any word in the vocabulary can appear in an article from either The Onion or The Economist. To make sure that our estimated probabilities are always non-zero, we will impose a Beta(2,1) prior on θ_{yw} and compute the MAP estimate from the training data.

Similarly, the distribution of Y (when we consider it alone) is a Bernoulli distribution (except taking values 1 and 2 instead of 0 and 1) with parameter ρ. In other words, there is some unknown number ρ such that

$$P(Y = 1) = \rho \quad \text{and} \quad P(Y = 2) = 1 - \rho.$$

In this case, since we have many examples of articles from both The Economist and The Onion, there is no risk of having zero-probability estimates, so we will instead use the MLE.

Programming Instructions

Parts (a) through (e) of this question each ask you to implement one function related to the Naive Bayes classifier. You will submit your code online through the CMU autolab system, which will execute it remotely against a suite of tests. Your grade will be automatically determined from the testing results. Since you get immediate feedback after submitting your code and you are allowed to submit as many different versions as you like (without any penalty), it is easy for you to check your code as you go.

Our autograder requires that you write your code in Octave. Octave is a free scientific programming language with syntax identical to that of MATLAB. Installation instructions can be found on the Octave website (http://www.gnu.org/software/octave/), and we have posted links to several Octave and MATLAB tutorials on Piazza.

To get started, you can log into the autolab website (https://autolab.cs.cmu.edu). From there you should see 10-601B in your list of courses. Download the template for Homework 3 and extract the contents (i.e., by executing `tar xvf hw3.tar` at the command line). In the archive you will find one `.m` file for each of the functions that you are asked to implement and a file that contains the data for this problem, `HW3Data.mat`. To finish each programming part of this problem, open the corresponding `.m` file and complete the function defined in that file. When you are ready to submit your solutions, you will create a new tar archive of the top-level directory (i.e., by executing `tar cvf hw3.tar hw3`) and upload that through the Autolab website.

The file `HW3Data.mat` contains the data that you will use in this problem. You can load it from Octave by executing `load("HW3Data.mat")` in the Octave interpreter. After loading the data, you will see that there are 7 variables: `Vocabulary`, `XTrain`, `yTrain`, `XTest`, `yTest`, `XTrainSmall`, and `yTrainSmall`.

- `Vocabulary` is a $V \times 1$ dimensional cell array that that contains every word appearing in the documents. When we refer to the j^{th} word, we mean `Vocabulary(j,1)`.
- `XTrain` is a $n \times V$ dimensional matrix describing the n documents used for training your Naive Bayes classifier. The entry `XTrain(i,j)` is 1 if word j appears in the i^{th} training document and 0 otherwise.
- `yTrain` is a $n \times 1$ dimensional matrix containing the class labels for the training documents. `yTrain(i,1)` is 1 if the i^{th} document belongs to The Economist and 2 if it belongs to The Onion.
- `XTest` and `yTest` are the same as `XTrain` and `yTrain`, except instead of having n rows, they have m rows. This is the data you will test your classifier on and it should not be used for training.
- Finally, `XTrainSmall` and `yTrainSmall` are subsets of `XTrain` and `yTrain` which are used in the final question.

Logspace Arithmetic

When working with very large or very small numbers (such as probabilities), it is useful to work in *logspace* to avoid numerical precision issues. In logspace, we keep track of the logs of numbers, instead of the numbers themselves. For example, if $p(x)$ and $p(y)$ are probability values, instead of storing $p(x)$ and $p(y)$ and computing $p(x) * p(y)$, we work in log space by storing $\log(p(x))$, $\log(p(y))$, and we can compute the log of the product, $\log(p(x) * p(y))$ by taking the sum: $\log(p(x) * p(y)) = log(p(x)) + log(p(y))$.

(a) [**1 Point**] Complete the function `logProd(x)` which takes as input a vector of numbers in logspace (i.e., $x_i = \log p_i$) and returns the product of those numbers in logspace—i.e., `logProd(x)` $= \log\big(\prod_i p_i\big)$.

Training Naive Bayes

(b) [**4 Points**] Complete the function `[D] = NB_XGivenY(XTrain, yTrain)`. The output `D` is a $2 \times V$ matrix, where for any word index $w \in \{1, \dots, V\}$ and class index $y \in \{1, 2\}$, the entry `D(y,w)` is the MAP estimate of $\theta_{yw} = P(X_w = 1|Y = y)$ with a Beta(2,1) prior distribution.

(c) [**4 Points**] Complete the function `[p] = NB_YPrior(yTrain)`. The output `p` is the MLE for $\rho = P(Y = 1)$.

(d) [**8 Points**] Complete the function `[yHat] = NB_Classify(D, p, X)`. The input `X` is an $m \times V$ matrix containing m feature vectors (stored as its rows). The output `yHat` is a $m \times 1$ vector of predicted class labels, where `yHat(i)` is the predicted label for the i^{th} row of `X`. [Hint: In this function, you will want to use the `logProd` function to avoid numerical problems.]

(e) [**1 Point**] Complete the function `[error] = ClassificationError(yHat, yTruth)`, which takes two vectors of equal length and returns the proportion of entries that they disagree on.

Questions

(g) [**4 Points**] Train your classifier on the data contained in `XTrain` and `yTrain` by running

```
D = NB_XGivenY(XTrain, yTrain);
p = NB_YPrior(yTrain);
```

Use the learned classifier to predict the labels for the article feature vectors in `XTrain` and `XTest` by running

```
yHatTrain = NB_Classify(D, p, XTrain);
yHatTest = NB_Classify(D, p, XTest);
```

Use the function `ClassificationError` to measure and report the training and testing error by running

```
trainError = ClassificationError(yHatTrain, yTrain);
testError = ClassificationError(yHatTest, yTest);
```

How do the train and test errors compare? Explain any significant differences.

Solution

I get a train error of 0 and a test error of 0.0207. These errors are quite close. The discrepency is because we calculated the MAP estimates for the feature probabilities over the train set, and applied them to the test set.

(h) [**4 Points**] Repeat the steps from part (g), but this time use the smaller training set `XTrainSmall` and `yTrainSmall`. Explain any difference between the train and test error in this question and in part (g). [Hint: When we have less training data, does the prior have more or less impact on our classifier?]

Solution

When using the small training set, I get a train error of 0.0966 and a test error of 0.2759. Error has increased in both cases compared to using the full train set. The test error is worse here because the training set is smaller, so the estimated probabilities are more inaccurate. More specifically, because we have less training data, it is more likely that we encounter novel words in the test set. Our Beta prior gives novel (unseen) words a probability of 1, which causes the error.

(i) [**4 Points**] Finally, we will try to interpret the learned parameters. Train your classifier on the data contained in `XTrain` and `yTrain`. For each class label $y \in \{1, 2\}$, list the five words that the model says are most likely to occur in a document from class y. Also for each class label $y \in \{1, 2\}$, list the five words w that maximize the following quantity:

$$\frac{P(X_w = 1|Y = y)}{P(X_w = 1|Y \neq y)}.$$

Which list of words describes the two classes better? Briefly explain your reasoning. (Note that some of the words may look a little strange because we have run them through a stemming algorithm that tries to make words with common roots look the same. For example, "stemming" and "stemmed" would both become "stem".)

Solution

The most common words for The Economist are ’the’, ’to’, ’of’, ’in’, ’a’. The most common words for the Onion are ’a’, ’and’, ’the’, ’to’, ’of’. We do not see much difference in the most common words. It is impossible to differentiate the two classes based on these words.

However, if we look at the words which maximize the defined quantity, we can see a difference. The top words for The Economist are ’organis’, ’reckon’, ’favour’, ’centr’, labour’. The top words for The Onion are ’4enlarg’, ’5enlarg’, ’monday’, ’percent’, and ’realiz’. There is a clear difference between these sets of words.

The second metric does better than the most frequent words because this new metric captures the frequency difference of words. The intuition is that there are words that are common in one class, but rare in the other. These words are a stronger signal for classification.