Programming Assignment 1

Samuel Cuthbertson SAMUEL.CUTHBERTSON@COLORADO.EDU

1. Introduction

In this assignment, I wrote a Naive Bayes Classifier for the **Iris** dataset. This dataset contains information about 3 different species of Iris, encoded in 150 data points formatted as shown in Table 1. This classifier takes as input some number of known feature vectors x_i, $i \in (1, 2, 3, 4)$, with known associated classes c_n, $n \in (1, 2, 3)$, and builds a model in order to take in vectors y_i and assign the most probable class c_n to them. In order to show that I achieved a reasonable accuracy, **Vowpal Wabbit** was used as a benchmark algorithm on the same data. The end results were that my classifier achieved 97% accuracy while training and testing on the same data, compared to Vowpal Wabbit's 98%. While training on 120 data points from the Iris dataset and testing on the other 30, my classifier achieved 86% accuracy, the exact same as Vowpal Wabbit's 86%.

SL	SW	PL	PW	Class
5.1	3.5	1.4	0.2	Iris-setosa
4.9	3.0	1.4	0.2	Iris-setosa
4.7	3.2	1.3	0.2	Iris-setosa
4.6	3.1	1.5	0.2	Iris-setosa

Table 1. Shows how the Iris dataset is formatted, where S stands for Sepal, P for Petal, L for length and W for width.

2. Methods

Note: This next section references specific lines from the file `classifier.py`, as well as other named files which should have arrived as part of the zip file containing this report.

2.1. Data Formatting

The first step in processing the Iris dataset was reformatting the csv it came in. First, I used the unix command `shuf` to shuffle the dataset and get rid of any bias that having a sorted dataset would impart. Then, I moved the the classes to the first column and replaced the class names with integers (1, 2 and 3). These decisions were made to both improve the code used to process the files, and to simplify the conversion into Vowpal Wabbit's format.

2.1.1. File Splitting - Test and Train Sets

After the reformat into the one shown in Table 2, the file `iris.full` (150 lines) was split into `iris.train` (120 lines) and `iris.test` (30 lines). All three of these files were subsequently copied and converted to Vowpal Wabbit's format, found in `vwtest/iris.vw`, `vwtest/iris.train.vw` and `vwtest/iris.test.vw`, respectively. Furthermore, the files `vwtest/iris.correct` and `vwtest/iris.test.correct` are composed of simply the first column of `iris.vm` and `iris.train.vm`, in order to test Vowpal Wabbit's results.

Class	x_1	x_2	x_3	x_4
3	5.8	2.7	5.1	1.9
2	6.9	3.1	4.9	1.5
2	5.6	2.7	4.2	1.3
2	6.7	3.1	4.7	1.5
1	4.5	2.3	1.3	0.3

Table 2. Shows how the input data for my algorithm is formatted

2.2. Data Proccessing

2.2.1. Loading Model

In the first section of `classifier.py` (lines 22 through 30), the training data is loaded into a three dimensional matrix `data_matrix` of the format shown in Table 3. The second section (lines 34 onward) deals much more with actual calculations.

1	2	3
$\{x_1, x_2, x_3, x_4\}$	$\{x_1, ..., x_4\}$	{...}
{...}	{...}	{...}
{...}	{...}	{...}

Table 3. Shows how the data_matrix is formatted, where the column labels are the classes.

2.2.2. Probability Calculations

In the second section, the test file is loading in and has its rows iterated over. Each iteration calculates the most probable class using Naive Bayes with Maximum Likelihood, as well as using Laplace smoothing to eliminate the false

negatives from data points not in the training data.

In simpler terms, the model calculates

$$argmax_c(p(c|x))$$

in order to find the class which is most probably the class of the input vector, x. To do this, we use the fact that

$$p(c|x) = \frac{p(x|c)p(c)}{p(x)}$$

Also assuming maximum likelihood and naive bayes, we can state that

$$p(x) = \sum_{n=1}^{3} p(x|c_n)p(c_n) = \sum_{n=1}^{3} \left(\left(\sum_{i=1}^{4} p(x_i|c_n) \right) p(c_n) \right)$$

$$p(x) = \sum_{n=1}^{3} \left(\left(\sum_{i=1}^{4} p(x_i|c) \right) \frac{1}{3} \right)$$

We are allowed to set the limits for n and i, as well as defining $p(c_n)$ as $\frac{1}{3}$, because we know those to be true about our dataset. This all leads to our $argmax_c(p(c|x))$ working to maximize

$$p(c|x) = \frac{\sum_{i=1}^{4} \left(p(x_i|c) * \frac{1}{3} \right)}{\sum_{n=1}^{3} \left(\left(\sum_{i=1}^{4} p(x_i|c_n) \right) \frac{1}{3} \right)}$$

The way I computed $p(x_i|c)$ is best explained through example. Let's say that we have x where $x_1 = 6.7$. I would count all the times that feature 1 would equal 6.7 for the current class under consideration in the data_matrix, and then divide that by all the times that feature 1 appeared in the current class under consideration. I also added a Laplace smoothing element by starting the first count, the count of all the times that feature 1 would equal 6.7 for the current class under consideration in the data_matrix, at 1. If feature 1 never equaled 6.7, then the $p(x_1|c)$ would be the inverse of how many times x_1 appeared in c.

All this put together leads simply to several for loops, which are somewhat unrolled in my code to speed things along. Computationally speaking, it's very simple when you begin to use log probabilities.

3. Results

My results were much better than expected. When testing and training on the full 150 data points, my classifier was able to correctly identify the classes of 97% of the points, as opposed to Vowpal Wabbit's 98%. This is understandably high, given that every input already exists in the model's data. The less expected result was the 86% accuracy when training on 80% of the data and only testing on the remaining 20%. This tied Vowpal Wabbit's results for the same split, and leads me to conclude that my classifier works very similarly to the way Vowpal Wabbit runs when using the commands provided by the instructor. It most likely handles continuous values similarly to the way I did (as discussed in 2.2.2), and it almost certainly makes many of the same assumptions I was most reluctant to make, but made anyways because it greatly simplified the math (Laplace Smoothing, Maximum Likelihood being the same from full dataset to reduced testing data, log probabilities). Overall, my classifier stood up very well when compared to the provided Vowpal Wabbit setup, and was robust in handling novel data.