CS440/ECE448

Fall 2016

Assignment 3: Naive Bayes Classification

Section R3

Patrick Wang (3 credit)

Armin Mohammadi (3 credit)

Ryan Chien (3 Credit)

Pwang39, Amohmmd2, Rchien2

Part 1: Digit Classification

1.1: Single Pixels as Features

We utilized the Bernoulli Naive Bayes Classifier to train a set of 5000 images of written numerical digits (0-9), then attempt to classify a set of test images, varying in clearness and quality of handwriting.

Implementation:
To perform the classifications, we first parsed through the training set of images to record the counts of training images (i.e. how many of each label) and the features at each pixel position for each label (i.e. counts of foreground and background pixels for each label).

With this training data, we performed the Bernoulli Naive Bayes calculation to classify each test image, by calculating the likelihoods of every pixel, for all possible labels, and the priors for all labels. We multiply all of these values together, and the label that has the highest probability will be our classification (in practice, we actually sum base-2 logs of these values because they tend to be very small). The following is the formula for the Bernoulli likelihood of a pixel with Laplace smoothing:

$$P(pixel(i,j) = value|Label) = \frac{\text{number of times pixel has this value in the training for this label} + k}{\text{total number of training images of this label} + kv}$$

For smoothing, we used v = 2 (pixel is either foreground or background), and ended up using k = .1 for the best overall accuracy (accuracy would lower with larger k value).

Our files for part 1, *readDigitFiles.py* and *countPixels.py*, were used for reading and parsing the training and test data, while *classifyTests.py* performes the Naive Bayes Classification, produces the confusion matrix to the terminal (values copied to tables in this report), and calculates the OddsRatio for two labels. *array2D.py* and *plot.py* print data matrices and heatmaps respectively.

We kept count of the predicted labels for all tests and their true labels to obtain the classification rates for all classifications (both accurate and inaccurate). On the next page are the results of our classifications for the test data.

Analysis of Test Images:
Overall Accuracy: 77.3%
Confusion Matrix (rates of classification for each label):

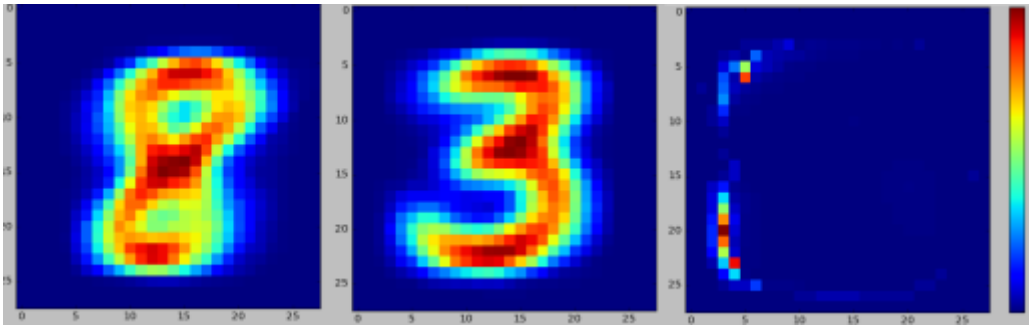
	Predicted										
Actual		0	1	2	3	4	5	6	7	8	9
	0	**84.4**	0.0	1.1	0.0	1.1	5.6	3.3	0.0	4.4	0.0
	1	0.0	**96.3**	0.9	0.0	0.0	1.9	0.9	0.0	0.0	0.0
	2	1.0	2.9	**78.6**	3.9	1.9	0.0	5.8	1.0	4.9	0.0
	3	0.0	1.0	0.0	**80.0**	0.0	3.0	2.0	7.0	1.0	6.0
	4	0.0	0.0	0.9	0.0	**74.8**	0.9	3.7	0.9	1.9	16.8
	5	2.2	1.1	1.1	13.0	3.3	**68.5**	1.1	1.1	2.2	6.5
	6	1.1	4.4	4.4	0.0	4.4	6.6	**76.9**	0.0	2.2	0.0
	7	0.0	4.7	3.8	0.0	2.8	0.0	0.0	**72.6**	2.8	13.2
	8	1.0	1.0	2.9	13.6	2.9	7.8	0.0	1.0	**60.2**	9.7
	9	1.0	1.0	0.0	3.0	10.0	2.0	0.0	2.0	1.0	**80.8**

From the above confusion matrix, the classification rates of each label are in **bold** across the diagonal, and the top four "confused" classification pairs are highlighted in red, which we used for our odds ratios analysis. The following page features heatmaps to represent these pairs.

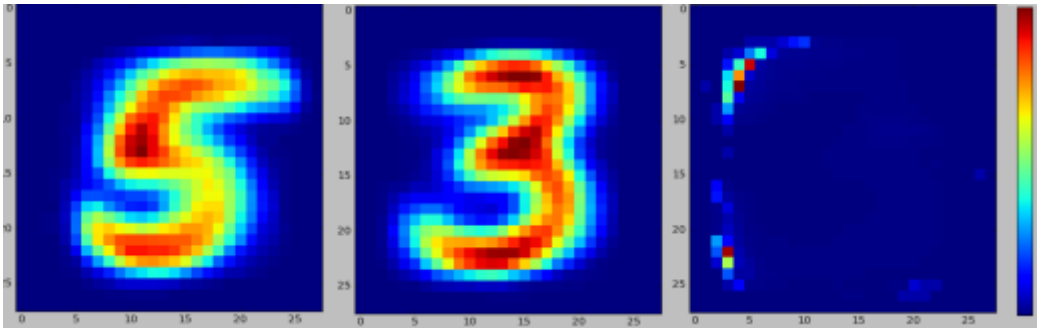
Top Four Odds Ratios Pairs (with Likelihoods):

**** Note: high log values are more red, low log values are more blue***

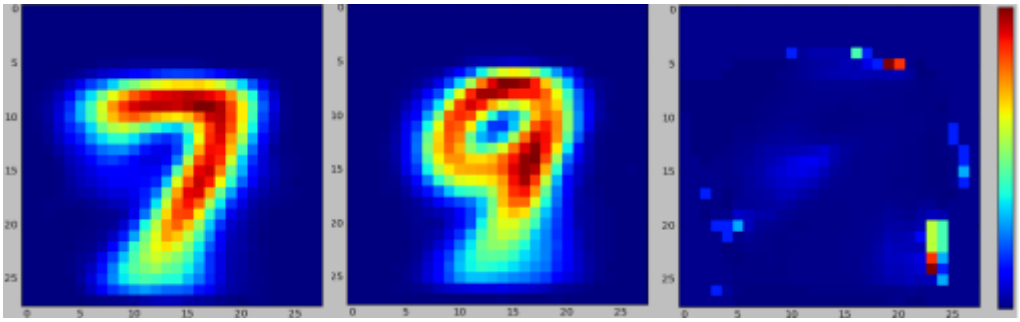
OddsRatio(8,3):



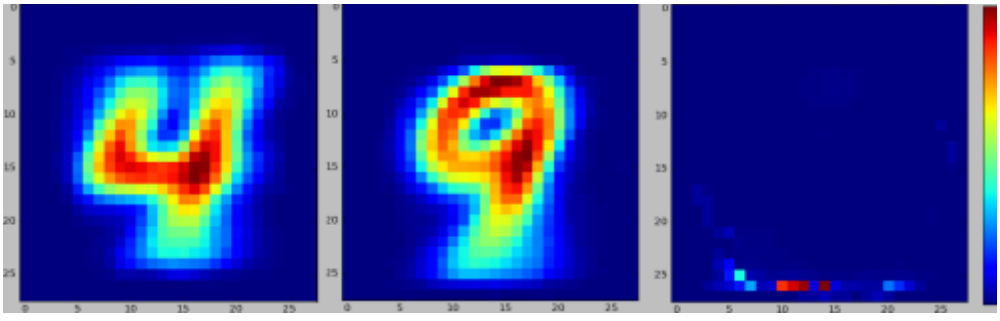
OddsRatio(5,3):



OddsRatio(7,9):



OddsRatio(4,9):



Best/Worst Tests:

The following are the best and worst test images (with their indices) in terms of classification for each possible label (0-9):

Extra Credit:
In an attempt to improve our classification accuracy, we decided to examine each image and its pixels not with just binary values, but instead with ternary values (i.e. distinct between a space, "#", and "+"). Now, we examine the foreground, background, *and* middleground pixels to try and better represent more specific intricacies in the images. We used the same Bernoulli Naive Bayes Classification process as before, with the same Laplacian smoothing constants (as they were equally effective in improving the overall accuracy), and the following are the results of the change to ternary pixel values:

Analysis of Test Images (Ternary):
Overall Accuracy: 77.8% (increase from binary)
Confusion Matrix (rates of classification for each label):

	Predicted										
Actual		0	1	2	3	4	5	6	7	8	9
	0	**83.3**	0.0	1.1	0.0	0.0	6.7	3.3	0.0	5.6	0.0
	1	0.0	**95.4**	0.0	0.0	0.0	1.9	0.9	0.0	1.9	0.0
	2	1.0	2.9	**76.9**	3.9	1.0	1.0	6.8	1.9	4.9	0.0
	3	0.0	1.0	0.0	**81.0**	0.0	3.0	2.0	6.0	2.0	5.0
	4	0.0	0.0	0.0	0.0	**76.6**	0.9	2.8	0.9	1.9	16.8
	5	2.2	1.1	1.1	13.0	3.3	**68.5**	1.1	1.1	2.2	6.5
	6	0.0	3.3	3.3	0.0	4.4	5.5	**81.3**	0.0	2.2	0.0
	7	0.0	5.7	2.8	0.0	2.8	0.0	0.0	**73.6**	2.8	12.3
	8	1.0	1.0	2.9	11.7	2.9	8.7	0.0	1.0	**60.2**	10.7
	9	1.0	0.0	0.0	2.0	10.0	2.0	0.0	2.0	2.0	**81.0**

Interestingly, while some individual accuracies decreased (ex: correct classification rate for 0), the overall accuracy increased by about 0.6%. Also, the highest confusion pairs remain the same,

though for the most part, the actual rates of confusion decreased. Overall, considering ternary values for pixels proves to be a more effective factor for classification.

Part 2: Text Document Classification

2.1: Sentiment analysis of movie reviews and binary conversation topic labeling

The second part of this MP required us to apply Naive Bayes classifiers to solve basic binary classification problems of sentiment analysis of movie reviews and conversation topic identification.

Implementation:

To store the vocabulary, we populated a dictionary of tuples (number of times word is used in positive reviews, number of times word is used in negative reviews) with the words as the keys:

1. Read training document
2. For each line of the text file:
    a. Look at first character to determine whether review is positive or negative
    b. Separate elements of the line by whitespaces to create a list of (word:count) pairs
    c. For each element of the list:
        i. If the word is not found in the list of keys:
            1. Add new entry into dictionary
        ii. Else:
            1. Replace tuple corresponding to found key with updated values
3. Return dictionary
4. Iterate over vocabulary dictionary

With the vocabulary dictionary from the training data, we performed the Multinomial and Bernoulli Naive calculations to calculate the likelihoods for each word for each class as well as the priors. We then added the logs of the likelihoods to calculate the likelihood of a word being part of a positive or negative review. The following is an example formula used for the Multinomial likelihood of a word for the sentiment analysis of movie reviews:

$$P(word = w \mid review = r) = \frac{number\ of\ times\ w\ in\ review\ type\ r + k}{total\ number\ of\ words\ in\ all\ type\ r\ reviews + kV}$$

As for smoothing, V corresponds to the total number of unique words in the training vocabulary and we used a value of 1 for k, similar to the example we did in lecture. We tried adjusting the k value for a higher overall accuracy but were unable to do so even by decreasing or increasing the k value.

We then kept count of predicted labels and actual labels to calculate the classification rates for each class, shown in the confusion matrices below. We also sorted the dictionaries containing likelihoods to obtain the top 10 words with the highest likelihoods and odds ratio, which are also listed below.

Sentiment analysis of movie reviews:

Top 10 words in descending order of highest odds ratio: *epic, thanks, quiet, among, warm, hilarious, examination, absorbing, thoughtful, witty*
Top 10 words for positive reviews in descending order of highest likelihood: *film, movie, --, one, like, story, **good**, **comedy**, **way**, even*
Top 10 words for negative reviews in descending order of highest likelihood: *movie, film, like, one, --, **bad,** story, **much**, **time**, even*
Observations: top 10 words are same for both models; top 10 for positive and negative classes share over half the words, the most significant difference being in good vs. bad.

(Extra Credit) Bag-of-words representation of movie sentiment document using word cloud maps with odds ratio: **See note at bottom about word cloud**

Multinomial Naive Bayes Model: Overall accuracy: **68.35%**
Confusion Matrix (classification rates for each label):

	Predicted Positive Review	Predicted Negative Review
Actual Positive Review	**68.95**	31.05
Actual Negative Review	32.27	**67.73**

Bernoulli Naive Bayes: Overall accuracy: **66.94%**
Confusion Matrix (classification rates for each label):

	Predicted Positive Review	Predicted Negative Review
Actual Positive Review	**68.41**	31.59
Actual Negative Review	34.58	**65.42**

Binary conversation topic classification:

Top 10 words with highest odds ratio: *welfare, wages, wage, minimum, salary, tax, poverty, mcdonald, employers, increase*
Top 10 words for positive reviews in descending order of highest likelihood: *know, yeah, like, uh, um, right, don, think, just, oh*
Top 10 words for negative reviews in descending order of highest likelihood: *know, yeah, uh, like, um, right, just, think, oh, don*
Observations: top 10 words shared in both models; top 10 words for positive and negative are all the same words in a different order, maybe because those words are generally tone neutral

(Extra Credit) Bag-of-words representation of binary conversation topics document using word cloud maps with odds ratio: **See note at bottom about word cloud**

Multinomial Naive Bayes Model: Overall accuracy: **92.48%**
Confusion Matrix (classification rates for each label):

	Predicted Minimum Wage Topic	Predicted Life Partners Topic
Actual Minimum Wage Topic	**96.40**	3.60
Actual Life Partners Topic	11.58	**88.42**

Bernoulli Naive Bayes: Overall accuracy: 92.47%
Confusion Matrix (classification rates for each label):

	Predicted Minimum Wage Topic	Predicted Life Partners Topic
Actual Minimum Wage Topic	**96.44**	3.56
Actual Life Partners Topic	11.64	**88.36**

Note: Word clouds generated using online word cloud generator (https://www.jasondavies.com/wordcloud/); the generator made the size of the words based on how many instances of the word in the input text. In this case, the number of occurrences corresponds to the odds ratio of the word. We also added a smoothing factor of 1 to account for words with ratios of 0. For example, if an odds ratio was 7, we would have printed the word 8 times.

Group Member Contribution:

Patrick: 33.3333333333333%
Armin: 33.3333333333333%
Ryan: 33.3333333333333%