CSE 537 Assignment 5 Report: ML Classifiers

Remy Oukaour
SBU ID: 107122849
remy.oukaour@gmail.com

Jian Yang
SBU ID: 110168771
jian.yang.1@stonybrook.edu

Wednesday, December 11, 2015

1 Introduction

This report describes our submission for assignment 5 in the CSE 537 course on artificial intelligence. Assignment 5 requires us to implement two kinds of machine learning classifiers: a decision tree classifier for predicting credit worthiness of applicants, and a naïve Bayes classifier for classifying handwritten digits. In this report, we discuss the implementation details and performance of our solutions.

2 Decision tree classifier

Jian Yang developed the decision tree classifier for predicting credit worthiness of applicants.

3 Naïve Bayes classifier

Remy Oukaour developed the naïve Bayes classifier for classifying handwritten digits.

3.1 Instructions

To run the classifier, enter *python naive-bayes.py*. It will read from *trainingimages.txt*, *traininglabels.txt*, *testimages.txt*, and *testlabels.txt*, output to *predictedlabels.txt* and print a confusion matrix to standard output.

3.2 Implementation

The classifier is a straightforward implementation of naïve Bayes that supports arbitrary categorical (a.k.a. generalized Bernoulli or multinoulli) features, using the formula "log posterior ∝ log prior + log likelihood":

$$\log P(label|features) \propto \log P(label) + \sum_{feature} P(feature|label)$$

The prior probability $P(label)$ is estimated to be the fraction of training instances with a given label (0 to 9). The likelihood $P(feature|label)$ of a feature having a certain value for a test instance is estimated to be the fraction of training instances (limited to that label) with that value for that feature. (We use Laplace smoothing to handle novel feature values in the test data, with a smoothing value of 0.001.) We then pick the label of each test instance using a maximum likelihood estimator:

$$classification(features) = \underset{labels}{\operatorname{argmax}} \ \log P(label|features)$$

3.3 Feature selection

We tested many different kinds of features to maximize performance. First, we decided how to use the images' pixel values: whether to treat gray as black, and whether to use groups of pixels as individual features. We counted the number of correctly classified test instances for each alternative.

- Individual pixels: 772 correct
- 2×2 blocks: 849 correct
- 2×2 overlapping blocks: 861 correct
- 3×3 blocks, 1-overlapping: 819 correct
- 3×3 blocks, 2-overlapping: 821 correct
- 4×4 blocks: 689 correct
- 4×4 blocks, 1-overlapping: 725 correct
- 4×4 blocks, 2-overlapping: 742 correct
- 4×4 blocks, 3-overlapping: 758 correct

We chose to use 2×2 overlapping blocks of pixels as features.

We also tried treating gray pixels ("+") as black ones ("#"), but this lowered accuracy from 861 to 857 correct.

The Laplace smoothing value of 0.001 was likewise chosen via testing. We expected 1 to work well, but it did not improve the accuracy from 861. Higher values actually lowered the accuracy, so we tried lower values until they no longer provided more benefit. With smoothing of 0.001, we reached 893 correctly classified test instances.

At this point we added an minimum entropy threshold for the features, on the grounds that a low-entropy feature would just be adding noise. (Many pixel-block features around the edges of the digit images are completely white for all training and test images, thus having 0 entropy and providing no benefit.) Testing a range of threshold values from 0 to 0.3, we found a peak at 0.15, with 897 correct classifications.

To achieve at least 90% accuracy, we added holistic features:

- *num_regions*: a count of contiguous regions in the image, treating gray and black pixels as "foreground" and white as "background," counted using a flood-fill algorithm.
- *spread_ratio*: the ratio of vertical to horizontal foreground "spread." Spread is the total distance of all foreground pixels from a center line.
- *horizontal_bias*: the difference between the number of foreground pixels in the top and bottom halves of the image.
- *vertical_bias*: the difference between the number of foreground pixels in the left and right halves of the image.

By adding these features to the existing block-based ones, we were able to correctly classify 903 out of 1,000 test instances.

3.4 Performance results

The classifier's precision for all ten digits ranged from 79.5% (for 8) to 96.3% (for 1). The sensitivity ranged from 81.1% (for 7) to 95.6% (for 0). The overall accuracy was 90.3%.

The most common errors were to misclassify a 7 as a 9 (which happened 9 times), an 8 as a 3 (which happened 7 times), or a 5 as an 8 (which happened 6 times). These are all the kind of errors that a human judge could also make with sloppy handwriting. In addition, of the 97 misclassified digits, some are illegible even for humans.

pred\true	**0**	**1**	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**	**total**	**precision**
0	86	0	1	0	0	1	1	0	1	1	91	94.5%
1	0	103	0	0	0	0	1	3	0	0	107	96.3%
2	0	1	95	2	0	1	0	5	2	0	106	89.6%
3	0	0	1	91	0	2	0	0	7	2	103	88.3%
4	0	1	0	0	101	0	0	0	2	3	107	94.4%
5	0	0	0	3	0	81	4	1	1	1	91	89.0%
6	1	1	1	0	2	0	83	0	0	0	88	94.3%
7	1	0	1	1	1	0	0	86	1	1	92	93.5%
8	2	2	4	1	0	6	2	2	89	4	112	79.5%
9	0	0	0	2	3	1	0	9	0	88	103	85.4%
total	90	108	103	100	107	92	91	106	103	100	1000	
sensitivity	95.6%	95.4%	92.2%	91.0%	94.4%	88.0%	91.2%	81.1%	86.4%	88.0%		

Table 1: Confusion matrix with precision and sensitivity scores for all ten labels.

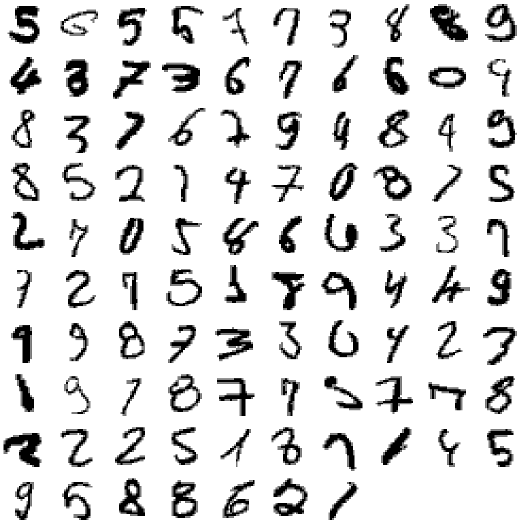

Figure 1: The 97 digits which the naïve Bayes classifier got wrong.