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.

2.1 Introduction

To run the classifier, please run *python load.py*. It will load the data in *crx.data.txt*, and print the result to screen for both the training error and the test error.

2.2 Algorithm

To implement the decision tree to classify the data, we used ID3 algorithm which would pick features and the split values based on Information Gain defined by:

$$IG(X, Y) = H(Y) - H(Y|X)$$

$$H(Y|X) = \sum_j Pr(X = v_j) H(Y|X = v_j)$$

$$H(X) = \sum_1^m p_j \log_2 p_j$$

For each feature, we would calculate the Information Gain for each feature.

2.2.1 Continuous features

What is special is for the continuous features, it doesn't make sense to use each value as a single v_j. Instead, we use a binary splitting for continuous features, and enumerate all possible split values, from the smallest value for this feature to the largest value.
For example, if a feature has values $1, 2, 3, 4$, we would try using value one to split it to two sets of data with the values as 1 and $2, 3, 4$ and calculate the information gain and then split by $1, 2$ and $3, 4$ and then split by $1, 2, 3$ and 4. By picking the most information gain, we could decide the feature and the related split value. If in a subset of data the feature has only one value left, we would eliminate the feature in subset of the training data.

2.2.2 Discrete features

For discrete features with multiple possible values, we have two choices to build a branch and to calculate the information gain.
One is to consider each value has a single branch, which means when we calculate the $H(Y|X)$, we calculate multiple such entropy values for each value. Another solution is using binary branch, which means we could split the value set to two sets and calculate the entropy for the probability an example is in one set.
In the first condition, the branch for a discrete feature might have as many branches as the number of values, and for the second condition, there should always be binary branches for the discrete features.

2.3 Data set

The data contains unknown values as "?" for both the continuous features and the discrete features. We use a simple smoothing method, by choosing the most frequent features of discrete features, and the median of the continuous features.
We prepared all the data first to fill values for unknown values "?".
And then we shuffled the data and splitted it to training examples and the test examples by half and half. The reason we need to shuffle first is, the data might be biased for the feature values and the classification labels.
For training examples, since we need do post-pruning, we splitted the training data by half and half. And using the first-half of training data for training and the second-half of training data for testing.

2.4 Post-pruning

After the building of the tree, we used post-pruning to reduce overfitting. The reason of overfitting is there are errors and noises in data.
By post-pruning, we applied the second half training data as validation data to classify based on the tree. For every node on the tree, we predicted the results, and also calculate the majority of the training data on this node, if using majority of the training data had higher accuracy than the current node, we would prune this node and its subtrees by using the majority value directly as the classify rule on this node.

2.5 Test results

Since we randomly split data in every time the script was run, the accuracy is not a fixed number. But in general statistics by running it may times, we observed that the training error is around 88% and the test error is around 86% for multiple branching of discrete features. And the binary branching of discrete features has similar but a little smaller accuracy around 87% for the training error and 86% for test error.
And the trees with multiple branching of discrete features before post-pruning is showed in fig 1 and after post-pruning is showed in fig 2. By comparing them we could see several nodes have been pruned.

As showed in the textbook, we also tested by different sizes of training examples. We used training examples from 10 to 300 and draw the learning curve for test accuracy and the result is showed in fig 3.

The trees with binary branching of discrete features before post-pruning is showed in fig 4 and after post-pruning is showed in fig 5.

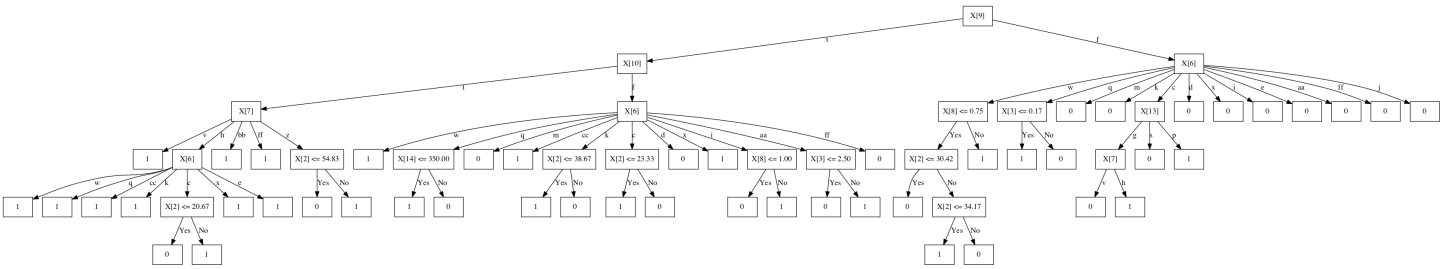

Figure 1: Decision Tree with multiple branching for discrete features before pruning. ($X[i]$ is feature A_i.)

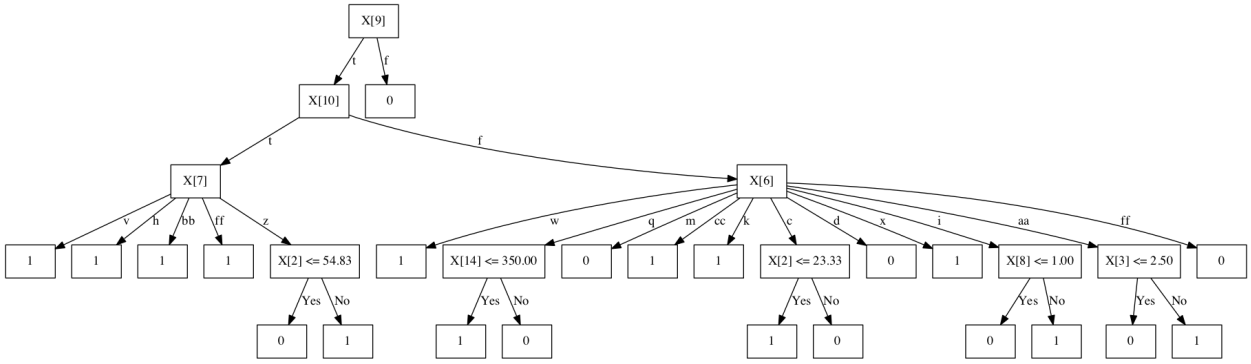

Figure 2: Decision Tree with multiple branching for discrete features after pruning.

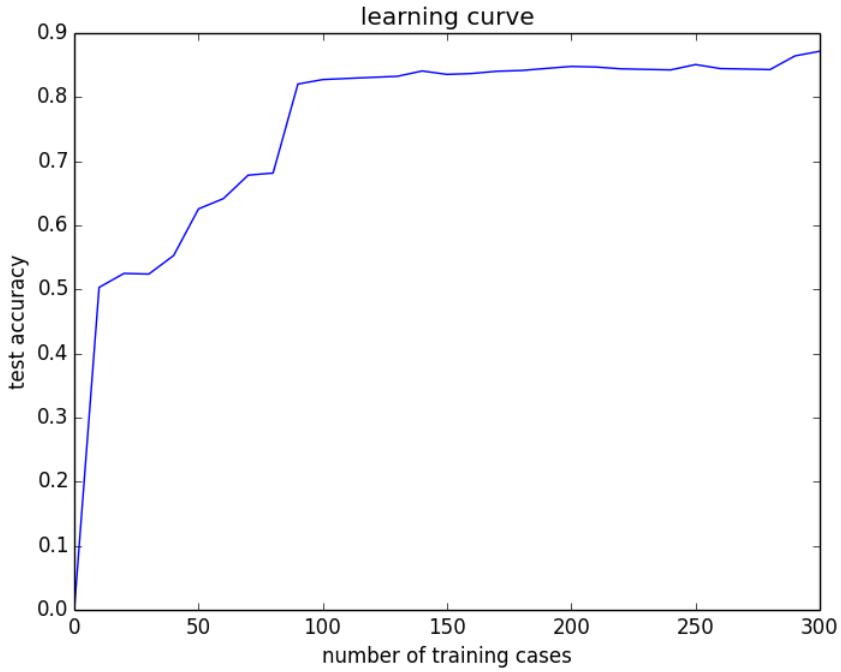

Figure 3: Learning curve for test accuracy.

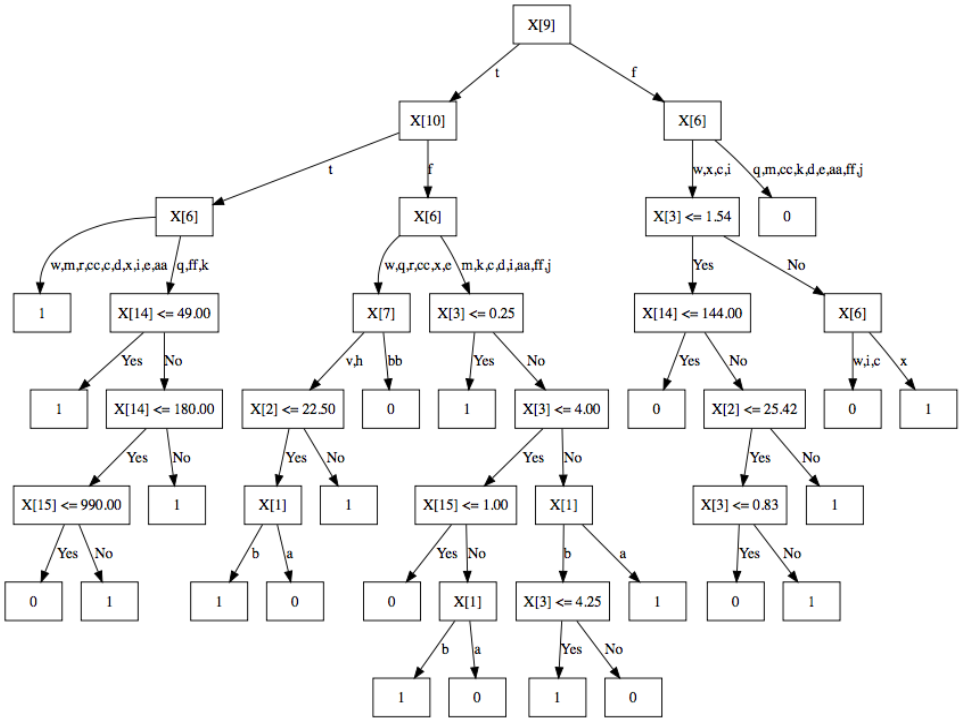

Figure 4: Decision Tree with binary branching for discrete features before pruning.

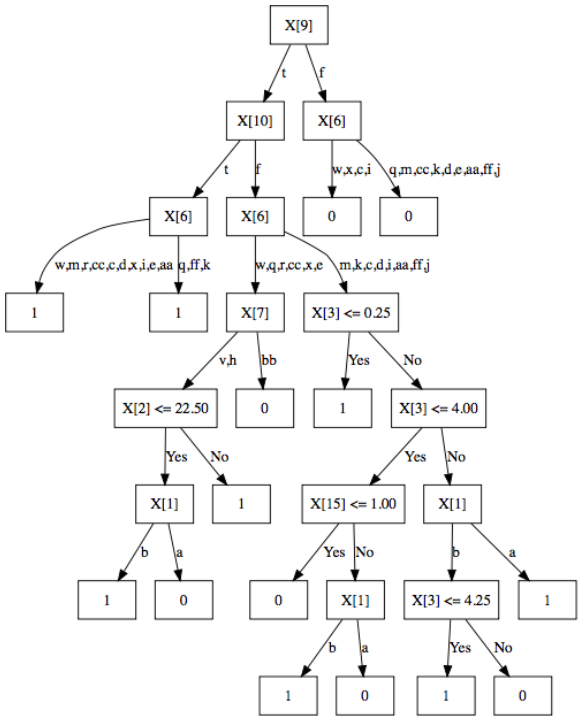

Figure 5: Decision Tree with binary branching for discrete features after pruning.

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 $\propto$ 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:

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

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%.

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.

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.

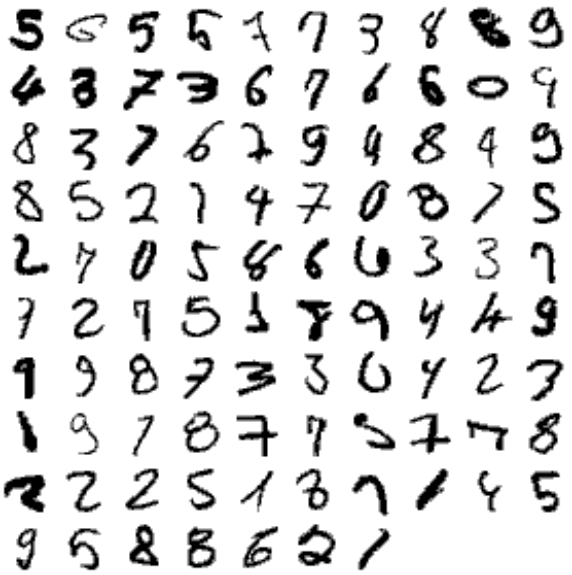

Figure 6: The 97 digits which the naïve Bayes classifier got wrong.