UNIVERSITY OF MARYLAND, BALTIMORE COUNTY

CMSC678 Homework 1

Yin Huang

February 5, 2015

1 PROBLEM ONE

In this part of the assignment you will gain familiarity with WEKA, the Waikato Environment for Knowledge Analysis. WEKA is widely used in the machine learning and data mining communities because, among other things, it provides both a nice user interface to a number of standard algorithms and a Java API.
First, you must download WEKA from the following URL: http://www.cs.waikato.ac.nz/ml/weka/. The "Getting Started" section of that page has links for information on system requirements, how to download the software, and documentation. WEKA is written in Java and should run on any platform with Java 1.5 or higher.
Read about the Adult Census Income dataset, and get it in the form of an ARFF file. Then do the following:

- Build a decision tree (J48 classifier) with the default parameters and report the (stratified cross-validation) accuracy.
- Now turn off pruning and report the accuracy. Inspect the output of the algorithm. Has it overfit? How can you tell?
- Build a decision stump (a decision tree with a single split; you can find it in the tree section of algorithms in Weka) and report the accuracy. Inspect the output of the algorithm. Has it underfit? How can you tell?

ANSWER 1 Below is the screen-shot of J48 classifier with the default parameters:

```
=== Stratified cross-validation ===
=== Summary ===

Correctly Classified Instances       41599               85.1705 %
Incorrectly Classified Instances      7243               14.8295 %
Kappa statistic                          0.5603
Mean absolute error                      0.2083
Root mean squared error                  0.3289
Relative absolute error                 57.2148 %
Root relative squared error             77.0846 %
Total Number of Instances            48842

=== Detailed Accuracy By Class ===

               TP Rate   FP Rate   Precision   Recall  F-Measure   ROC Area  Class
                 0.583     0.064      0.742     0.583     0.653      0.875    >50K
                 0.936     0.417      0.877     0.936     0.906      0.875    <=50K
Weighted Avg.    0.852     0.333      0.845     0.852     0.845      0.875

=== Confusion Matrix ===

     a     b   <-- classified as
  6811  4876 |     a = >50K
  2367 34788 |     b = <=50K
```

Number of Leaves : 689 Size of the tree : 834

ANSWER 2 Below is the screen-shot of J48 classifier without pruning:

```
=== Stratified cross-validation ===
=== Summary ===

Correctly Classified Instances       40758               83.4487 %
Incorrectly Classified Instances      8084               16.5513 %
Kappa statistic                          0.5282
Mean absolute error                      0.195
Root mean squared error                  0.3571
Relative absolute error                 53.5662 %
Root relative squared error             83.7107 %
Total Number of Instances            48842

=== Detailed Accuracy By Class ===

               TP Rate   FP Rate   Precision   Recall  F-Measure   ROC Area  Class
                 0.601     0.092      0.672     0.601     0.635      0.842    >50K
                 0.908     0.399      0.879     0.908     0.893      0.842    <=50K
Weighted Avg.    0.834     0.325      0.829     0.834     0.831      0.842

=== Confusion Matrix ===

     a     b   <-- classified as
  7026  4661 |     a = >50K
  3423 33732 |     b = <=50K
```

Number of Leaves : 13074 Size of the tree : 14871 Given the fact that the total size of the training set is 48842, and the output size of our decision tree is 14871 with 13074 leaves, we

can safely assume this is overfitting because almost one quoter of our training set is used to build the decision tree. In a word, our tree has a high variance but low bias. In order to verify our assumption, we need to test the accuracy using other test data with no overlapping datasets with our training set.

ANSWER 3 Below is the screen-shot of a decision dump with our dataset.

```
=== Classifier model (full training set) ===

Decision Stump

Classifications

marital-status = Married-civ-spouse : <=50K
marital-status != Married-civ-spouse : <=50K
marital-status is missing : <=50K

Class distributions

marital-status = Married-civ-spouse
>50K    <=50K
0.4461325349658162      0.5538674650341838
marital-status != Married-civ-spouse
>50K    <=50K
0.06435400370328383     0.9356459962967162
marital-status is missing
>50K    <=50K
0.23928176569346055     0.7607182343065395


Time taken to build model: 0.31 seconds

=== Stratified cross-validation ===
=== Summary ===

Correctly Classified Instances       37155               76.0718 %
Incorrectly Classified Instances     11687               23.9282 %
Kappa statistic                          0
Mean absolute error                      0.2917
Root mean squared error                  0.3819
Relative absolute error                 80.1222 %
Root relative squared error             89.5133 %
Total Number of Instances            48842

=== Detailed Accuracy By Class ===

               TP Rate   FP Rate   Precision   Recall  F-Measure   ROC Area  Class
                 0         0          0         0         0          0.756    >50K
                 1         1          0.761     1         0.864      0.756    <=50K
Weighted Avg.    0.761     0.761      0.579     0.761     0.657      0.756

=== Confusion Matrix ===

     a     b   <-- classified as
     0 11687 |     a = >50K
     0 37155 |     b = <=50K
```

As we can see, the tree is only testing on one feature: marital-status and then give us the income label. This is absolutely underfitting since we have 15 attributes. In a word, our tree has a high bias but low variance.

2 PROBLEM TWO

In this part of the homework you will implement k-means clustering and experiment with different ways of initializing the cluster centroids.
The MNIST dataset is a well-studied collection of handwritten digits. It is often used to test multi-class classification algorithms, where there is one class for each of the 10 digits (0 - 9). In this homework, you will use it for unsupervised clustering.
I've made two files available for you:

- The raw MNIST data, which is a text file containing 10,000 rows. Each row contains 28 * 28 = 784 integers in the range 0 to 255. Each integer is the pixel value from a 28 x 28 image of a handwritten digit. Every row corresponds to a vector in the dataset that is to be clustered.
- The labels for the raw data are in a file with 10,000 rows. The first row contains the correct digit label for the first row in the raw data. The second row is the label for the second instance, and so on.

Implement the k-means clustering algorithm. You will only use your algorithm for this dataset, so you can hard-wire in the number of instances and the size of each instance.
The goal is not to write a generic version of the algorithm (though you can if you wish). The goal is to understand how it works on real data. You will need to try different values of k so that must be a parameter.
After completing the implementation (and testing for correctness, of course), do the following:

- Randomly sample k = 10 instances, use them as the initial cluster centroids, and run the algorithm to covergence. For each cluster, find the most common digit in that cluster and count the number of instances in the cluster that are different from the most common one. Sum that count over all of the clusters.
- Repeat the above step 10 times in total and report the average number of iterations to convergence and the average number of instances that are in the wrong cluster.
- Run the algorithm with k = 5. Look at the clusters and see if there are digits that tend to get grouped together. What are they and explain why you think they are grouped into the same cluster.
- Finally, run the algorithm 10 times again with k = 10 and report the same information as above (iterations to convergence and number of wrongly clustered instances). But this time do not choose random instances for the cluster centroids. Randomly choose an instance that represents each of the digits and use them as the centroids. That is, one of the centroids will be a randomly chose 0, another will be a randomly chose 1, and so on. Do you observe any difference in the performance statistics? Why or why not?
- Turn in hard copy of your code.