[CSE537] Classification

REPORT for TEAM PROJECT #4

by Oleksii Starov, Hyungjoon Koo

Nov. 19, 2013

A. General Information

(1) Team-up (Group 41): Oleksii Starov (ostarov@cs.stonybrook.edu), Hyungjoon Koo (hykoo@cs.stonybrook.edu)

(2) Project information: Classification (PJT#4), http://www.cs.sunysb.edu/~ram/cse537/project04-2013.html (Due date: Nov. 19, 2013)

(3) Implementation: ID3 Decision Tree, Naïve Bayes, Odd Ratios (Extra)

B. Implementation Note

(1) **Manual Analysis with Small Data**:

Observation	Feature1 (0,0)	Feature2 (0,1)	Feature3 (0,2)	Results
X_1	0	0	0	0
X_2	1	0	0	1
X_3	0	1	1	2
X_4	1	1	0	2
X_5	0	0	1	1

Let the result be the number of 1s in observations. Each feature holds either TRUE(1) or FALSE(0) (Y:Features, X:Results)

a. To build a decision tree, we computed the initial entropy. The result set is {0, 1, 2}. Each result value holds the probability of 1/5, 2/5 and 2/5. The initial entropy $H(X) = -1/5 \log(1/5) - 2/5 \log(2/5) - (2/5) \log(2/5) = 1.05492$. Let us find the feature (or attribute) to maximize the information gain from possible hypothesis space. The feature which has the greatest value of information gain would be the first root in a decision tree.

Y=F1 → $H(X|F1=True) = -0/2 \log(0/2) - 1/2 \log(1/2) - 1/2 \log(1/2) = 0.69315$

→ $H(X|F1=False) = -1/3 \log(1/3) - 1/3 \log(1/3) - 1/3 \log(1/3) = 1.09862$

→ $IG(X|F1) = 1.05492 - (2/5 * 0.69315 + 3/5 * 1.09862) = 0.11849$

Y=F2 → $H(X|F2=True) = -0/2 \log(0/2) - 0/2 \log(0/2) - 2/2 \log(2/2) = 0$

→ $H(X|F2=False) = -1/3 \log(1/3) - 2/3 \log(2/3) - 0/3 \log(0/3) = 0.63651$

→ $IG(X|F2) = 1.05492 - (2/5 * 0 + 3/5 * 0.63651) = 0.673014$

Y=F3 → $H(X|F3=True) = -0/2 \log(0/2) - 1/2 \log(1/2) - 1/2 \log(1/2) = 0.69315$

→ $H(X|F3=False) = -1/3 \log(1/3) - 1/3 \log(1/3) - 1/3 \log(1/3) = 1.09862$

→ $IG(X|F3) = 1.05492 - (2/5 * 0.69315 + 3/5 * 1.09862) = 0.11849$

```
    (0,1)
   1↙  ↘0
  2    (0,0)
      1↙  ↘0
     1    (0,2)
          1↙  ↘0
          1     0
```

Finally, we should select F2 as a root node in a decision tree. The figure on the right side is the tree after all process is done.

b. In a naïve Bayes, we can get the prior probability and the conditional probability as following:

Prior Probability: P(X=0)=0.2, P(X=1)=0.4, P(X=2)=0.4

Conditional Probability:

[F1] P(F1=0|Y=0) = 1, P(F1=1|Y=0) = 0

P(F1=0|Y=1) = 0.5, P(F1=1|Y=1) = 0.5

P(F1=0|Y=2) = 0.5, P(F1=1|Y=2) = 0.5

[F2] P(F2=0|Y=0) = 1, P(F2=1|Y=0) = 0

P(F2=0|Y=1) = 1, P(F2=1|Y=1) = 0

P(F2=0|Y=2) = 0, P(F2=1|Y=2) = 1

[F3] P(F3=0|Y=0) = 1, P(F3=1|Y=0) = 0

P(F3=0|Y=1) = 0.5, P(F3=1|Y=1) = 0.5

P(F3=0|Y=2) = 0.5, P(F3=1|Y=2) = 0.5

(2) **ID3**

ID3.py has a class of *DecisionTreeClassifier*. It consists of a couple of important functions: *train, chooseTheBestInformationGain, buildDecisionTree, searchTheTree* and *classify*.

```
class DecisionTreeClassifier(classificationMethod.ClassificationMethod):

    def __init__(self, legalLabels):
        self.guess = None
        self.type = "id3"
        self.tree = {}

    def train(self, data, labels, validationData, validationLabels):
        allTrainingFeatures = data[0].keys()
        t = self.buildDecisionTree(data, labels, allTrainingFeatures, [0, 1, 2, 3, 4, 5, 6, 7, 8, 9])
        self.tree = t

    def chooseTheBestInformationGain(self, allData, allLabels, legalFeatures, legalLabels, priorCounts):
        priorH = 0
        for l in legalLabels:
            if l in priorCounts:
                p = priorCounts[l] / float(len(allData))
                if p > 0:
                    priorH += -p * math.log(p)
        maxIG = -1
        bestF = -1
        for f in legalFeatures:
            labelCountsTrue = {}
            labelCountsFalse = {}
```

```
        totalNumberTrue = 0
        totalNumberFalse = 0
        cnt = 0
        for row in allData:
            # Value of this feature in a one evidence
            v = row[f]
            if v == 1:
                totalNumberTrue += 1
                if allLabels[cnt] in labelCountsTrue:
                    labelCountsTrue[allLabels[cnt]] += 1
                else:
                    labelCountsTrue[allLabels[cnt]] = 1
            else:
                totalNumberFalse += 1
                if allLabels[cnt] in labelCountsFalse:
                    labelCountsFalse[allLabels[cnt]] += 1
                else:
                    labelCountsFalse[allLabels[cnt]] = 1
            cnt += 1
        hForTrue = 0
        for l in legalLabels:
            if l in labelCountsTrue:
                p = labelCountsTrue[l] / float(totalNumberTrue)
                if p > 0:
                    hForTrue += -p * math.log(p)
        hForFalse = 0
        for l in legalLabels:
            if l in labelCountsFalse:
                p = labelCountsFalse[l] / float(totalNumberFalse)
                if p > 0:
                    hForFalse += -p * math.log(p)
        IG = priorH - (totalNumberTrue / float(len(allData))) * hForTrue - (totalNumberFalse / float(len(allData))) * hForFalse
        if IG > maxIG:
            maxIG = IG
            bestF = f
    return bestF

def buildDecisionTree(self, allData, allLabels, legalFeatures, legalLabels):
    # Preliminary, calculating of occurrences
    priorCounts = {}
    popularLabel = 0 # default
    maxPopularity = -1
    for l in allLabels:
        if l in priorCounts:
            priorCounts[l] += 1
        else:
            priorCounts[l] = 1
        if maxPopularity < priorCounts[l]:
            maxPopularity = priorCounts[l]
            popularLabel = l

    # First, base cases with leaf nodes
```

```
        if len(legalFeatures) == 0:
            # All decisions made, select what is most likely
            return {'val': popularLabel, 'children':[]}
        if len(allData) == 0:
            # No other train evidences
            return {'val': legalLabels[0], 'children':[]}
        if priorCounts[popularLabel] == len(allLabels):
            # Only one choice is left
            return {'val': popularLabel, 'children':[]}

        # Otherwise, choose next attribute-feature with the best IG
        nextFeature = self.chooseTheBestInformationGain(allData, allLabels, legalFeatures, legalLabels, priorCounts)
        currentNode = {'attr': nextFeature, 'children':[]}

        # A node can have only two children
        listFisTrue = []
        labelsFisTrue = []
        listFisFalse = []
        labelsFisFalse = []

        cnt = 0;
        for row in allData:
            # Value of this feature in a one evidence
            v = row[nextFeature]
            if v == 1:
                listFisTrue.append(row)
                labelsFisTrue.append(allLabels[cnt])
            else:
                listFisFalse.append(row)
                labelsFisFalse.append(allLabels[cnt])
            cnt += 1

        # First child is by '1' value
        currentNode['children'].append(
            self.buildDecisionTree(listFisTrue, labelsFisTrue, [f for f in legalFeatures if f != nextFeature], legalLabels))

        # Second child is by '0' value
        currentNode['children'].append(
            self.buildDecisionTree(listFisFalse, labelsFisFalse, [f for f in legalFeatures if f != nextFeature], legalLabels))
        return currentNode

    def searchTheTree(self, oneSample):
        dict = self.tree
        while len(dict['children']) > 0:
            f = dict['attr']
            if oneSample[f] == 1:
                dict = dict['children'][0]
            else:
                dict = dict['children'][1]

        return dict['val']
```

```
    def classify(self, testData):

        result = []
        for test in testData:
            result.append(self.searchTheTree(test))
        return result
```

(3) **NaiveBayes**

The NaiveBayes consists of *trainAndTune* and *calculateLogJointProbabilities* with two helper functions.

```
def trainAndTune(self, trainingData, trainingLabels, validationData, validationLabels, kgrid):

    allLabels = self.legalLabels
    allTrainingFeatures = [ f for datum in trainingData for f in datum.values() ];
    numOfAllLabels = len(allLabels)
    numOfTrainingData = len(trainingLabels)
    squareLength = int(math.sqrt(len(allTrainingFeatures)/len(trainingLabels)))

    # c(y): the number of training instances with label y
    # Set Prior Distribution from Training Data
    self.priorDistribution = {}
    self.setPriorDistribution(allLabels, trainingLabels)
    print "Prior Distribution: " + str(self.priorDistribution)

    # Choose the best k value from kgrid when setting "autotune"
    maxCorrect = -1
    correctK = -1
    self.condProbOfFeaturesList = {}
    for k in kgrid:
        self.condProbOfFeatures = {}
        self.setCondProbOfFeatures(numOfAllLabels, squareLength,numOfTrainingData, trainingData, trainingLabels, k)
        guesses = self.classify(validationData)
        correct = [guesses[i] == validationLabels[i] for i in range(len(validationLabels))].count(True)
        self.condProbOfFeaturesList[k] = dict(self.condProbOfFeatures)
        if maxCorrect < correct:
            maxCorrect = correct
            correctK = k
        print "Current k: {}, Rate of Correctness: {}%".format(k, correct)

    print "Final Selected k: {}, Maximum Rate of Correctness: {}".format(correctK, maxCorrect)
    self.condProbOfFeatures = dict(self.condProbOfFeaturesList[correctK])

  # Helper method to set the prior distributions for Y
  # Using training data,count all features

  def setPriorDistribution (self, allLabels, trainingLabels):
```

```
    for label in allLabels :
        dataCtr = 0
        for tLabel in trainingLabels :
            if tLabel == label :
                dataCtr = dataCtr + 1
        estimateProb = float(dataCtr) / float(len(trainingLabels))
        self.priorDistribution[label] = estimateProb

  # Helper method to compute the conditional probability of all features
  def setCondProbOfFeatures(self, numOfAllLabels, squareLength, numOfTrainingData, trainingData, trainingLabels, k):

    for y in range(0, numOfAllLabels) :
        for i in range(0, squareLength):
            for j in range(0, squareLength):
                dataCtr = 0
                for index in range(0,numOfTrainingData):
                    if y==trainingLabels[index] and trainingData[index][(i,j)]==1 :
                        dataCtr = dataCtr + 1
                        # Given y, the number of times pixel Fi, took value fi = 1
                    self.condProbOfFeatures[(y,i,j)] = float(format((dataCtr + k)/(self.priorDistribution[y]*numOfTrainingData +
                                                                                                          \k),'.8f'))
def calculateLogJointProbabilities(self, datum):

    for label in self.legalLabels:
        logProbabilities = 0
        py = self.priorDistribution[label]
        for key in datum.keys():
            (i,j) = key
            if datum[(i,j)] == 1:
                value = self.condProbOfFeatures[(label,i,j)]
            else:
                value = 1 - self.condProbOfFeatures[(label,i,j)]
            if value > 0 :
               logProbabilities = logProbabilities + math.log(value)
        logProbabilities = math.log(py) + logProbabilities
        logJoint[label] = logProbabilities

    return logJoint
```

(4) **findHighOddsFeatures**

```
def findHighOddsFeatures(self, label1, label2):

    featuresOdds = []

    allOdds = []
    for (i,j) in self.features:
        odds = self.condProbOfFeatures[(label1, i, j)] / self.condProbOfFeatures[(label2, i, j)]
        # Will be sorted by odds first
```

```
        allOdds.append((odds, i, j))

    allOdds.sort()

    for x in range(len(allOdds)-1, max(0, len(allOdds)-100), -1):
        (odds, i, j) = allOdds[x]
        featuresOdds.append((i,j))

    return featuresOdds
```

C. The results

Here is the table we have gotten from our implementation. The more training set the better. Also the *autotune* option allows us to find the optimal *k* from *kgrid*, which improves the correctness a lot. The number within parenthesis in red means the ratio of successful guesses in the Validation and Testing sets respectively. We used the "**reduced-error pruning**" technique to consider each of the decision nodes in the tree to be candidates for pruning. The book says, "*Pruning a decision node consists of removing the subtree rooted at that node, making it a leaf node and assigning it the most common classification of the training examples affiliated with that node. Nodes are only removed if the resulting pruned tree performs no worse than original over the validation set.*"

After pruning implementation, we found that this does not improve a lot when the dataset is small (100). However, as dataset grows, we could see there was an improvement in ID3 (decision tree). We also discovered that validation dataset is better fitting to the decision tree than testing dataset. **Figure 1** shows the position in the decision tree where pruning happens. **Table 1 and Table 2** summarize the entire results we get with each parameter respectively.

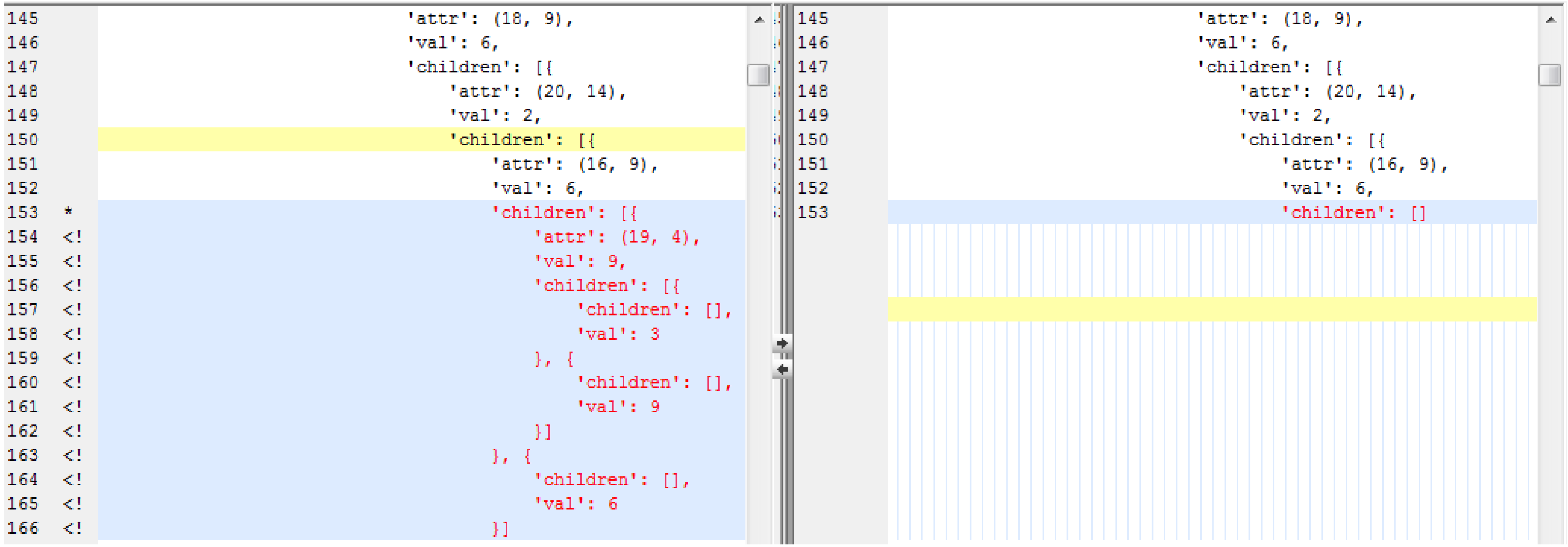

Figure1. Pruning result

Table1. Test Result for *ID3*

ID3 / Dataset Size		100	1,000	3,000
Validation data	With Pruning	54%	63%	69%
	Without Pruning	46%	67%	76%
Testing data	With Pruning	55%	68%	75%
	Without Pruning	43%	67%	77%

Table2. Test Result for *naiveBayes*

ID3 / Dataset Size		K value	100	1,000
Validation data	Default	2	66%	75%
	Autotunes	0.05	53%	64%
Testing data	Default	2	81%	82%
	Autotunes	0.001	77%	78%

ID3 with training set of 100 without Pruning (54/46)

```
data:          digits
classifier:             id3
using enhanced features?:     False
training set size:    100
Extracting features...
Training...
```

{'attr': (15, 6), 'children': [{'attr': (20, 9), 'children': [{'attr': (16, 14), 'children': [{'attr': (8, 23), 'children': [{'attr': (18, 19), 'children': [{'children': [], 'val': 3}, {'children': [], 'val': 8}]}, {'children': [], 'val': 2}]}, {'children': [], 'val': 0}]}, {'attr': (9, 18), 'children': [{'attr': (16, 9), 'children': [{'attr': (18, 19), 'children': [{'children': [], 'val': 8}, {'children': [], 'val': 2}]}, {'children': [], 'val': 6}]}, {'attr': (13, 18), 'children': [{'attr': (10, 24), 'children': [{'children': [], 'val': 9}, {'children': [], 'val': 1}]}, {'attr': (20, 7), 'children': [{'children': [], 'val': 5}, {'children': [], 'val': 3}]}]}]}]}, {'attr': (14, 13), 'children': [{'attr': (11, 9), 'children': [{'attr': (22, 19), 'children': [{'children': [], 'val': 3}, {'children': [], 'val': 9}]}, {'attr': (17, 15), 'children': [{'attr': (17, 10), 'children': [{'children': [], 'val': 4}, {'attr': (16, 9), 'children': [{'children': [], 'val': 5}, {'children': [], 'val': 3}]}]}, {'children': [], 'val': 1}]}]}, {'attr': (11, 15), 'children': [{'attr': (17, 17), 'children': [{'children': [], 'val': 4}, {'attr': (21, 8), 'children': [{'children': [], 'val': 5}, {'attr': (16, 9), 'children': [{'children': [], 'val': 9}, {'children': [], 'val': 6}]}]}]}, {'attr': (21, 6), 'children': [{'children': [], 'val': 4}, {'children': [], 'val': 7}]}]}]}]}

```
Validating...
```

[7, 2, 1, 8, 7, 1, 3, 9, 6, 9, 0, 1, 4, 0, 1, 7, 2, 7, 1, 4, 2, 9, 1, 3, 4, 0, 7, 4, 0, 1, 3, 1, 3, 7, 4, 0, 7, 1, 3, 1, 1, 7, 4, 2, 3, 6, 9, 6, 9, 4, 6, 2, 4, 1, 1, 0, 4, 1, 4, 5, 7, 6, 1, 3, 4, 9, 1, 4, 3, 0, 7, 0, 7, 8, 1, 9, 3, 7, 9, 7, 4, 6, 2, 7, 0, 4, 9, 3, 6, 1, 5, 1, 2, 3, 9, 9, 9, 1, 6, 3]

```
54 correct out of 100 (54.0%).
Testing...
```

[4, 6, 2, 2, 1, 3, 7, 9, 1, 0, 3, 1, 4, 3, 4, 9, 7, 7, 2, 1, 3, 7, 2, 8, 5, 1, 1, 1, 3, 2, 1, 2, 6, 8, 0, 7, 1, 1, 1, 1, 1, 5, 9, 4, 9, 3, 3, 4, 7, 6, 4, 4, 2, 0, 1, 3, 8, 4, 3, 0, 4, 0, 9, 9, 6, 2, 3, 3, 1, 8, 5, 7, 3, 9, 3, 1, 2, 9, 7, 6, 4, 3, 5, 1, 0, 6, 3, 9, 9, 6, 1, 2, 4, 6, 0, 3, 7, 1, 4, 5]

```
46 correct out of 100 (46.0%).
```

ID3 with training set of 100 with Pruning (55/43)

```
data:          digits
classifier:             id3
using enhanced features?:     False
training set size:    100
Extracting features...
Training...
Prune:  54  to  55
```

{'attr': (15, 6), 'val': 1, 'children': [{'attr': (20, 9), 'val': 0, 'children': [{'attr': (16, 14), 'val': 0, 'children': [{'attr': (8, 23), 'val': 8, 'children': [{'attr': (18, 19), 'val': 8, 'children': [{'children': [], 'val': 3}, {'children': [], 'val': 8}]}, {'children': [], 'val': 2}]}, {'children': [], 'val': 0}]}, {'attr': (9, 18), 'val': 6, 'children': [{'attr': (16, 9), 'val': 6, 'children': [{'attr': (18, 19), 'val': 2, 'children': [{'children': [], 'val': 8}, {'children': [], 'val': 2}]}, {'children': [], 'val': 6}]}, {'attr': (13, 18), 'val': 1, 'children': [{'attr': (10, 24), 'val': 1, 'children': [{'children': [], 'val': 9}, {'children': [], 'val': 1}]}, {'attr': (20, 7), 'val': 3, 'children': []}]}]}]}, {'attr': (14, 13), 'val': 4, 'children': [{'attr': (11, 9), 'val': 9, 'children': [{'attr': (22, 19), 'val': 9, 'children': [{'children': [], 'val': 3}, {'children': [], 'val': 9}]}, {'attr': (17, 15), 'val': 1, 'children': [{'attr': (17, 10), 'val': 4, 'children': [{'children': [], 'val': 4}, {'attr': (16, 9), 'val': 5, 'children': [{'children': [], 'val': 5}, {'children': [], 'val': 3}]}]}, {'children': [], 'val': 1}]}]}, {'attr': (11, 15), 'val': 7, 'children': [{'attr': (17, 17), 'val': 4, 'children': [{'children': [], 'val': 4}, {'attr': (21, 8), 'val': 5, 'children': [{'children': [], 'val': 5}, {'attr': (16, 9), 'val': 6, 'children': [{'children': [], 'val': 9}, {'children': [], 'val': 6}]}]}]}, {'attr': (21, 6), 'val': 7, 'children': [{'children': [], 'val': 4}, {'children': [], 'val': 7}]}]}]}]}

```
Validating...
55 correct out of 100 (55.0%).
Testing...
43 correct out of 100 (43.0%).
```

ID3 with training set of 1,000 without Pruning (63/67)	**ID3 with training set of 1,000 with Pruning (68/67)**
data: digits classifier: id3 using enhanced features?: False training set size: 1000 Extracting features... Training... {'attr': (14, 13), 'children': [{'attr': (13, 17), 'children': [{'attr': (11, 12), 'children': [{'attr': (20, 15), 'children': [{'attr': (17, 9), 'children': [{'attr': (17, 20), 'children': [{'attr': (22, 19), 'children': [{'children': [], 'val': 3}, {'children': [], 'val': 8}]}, {'children': [], 'val': 4}]}, {'attr': (9, 15), 'children': [{'children': [], 'val': 6}, {'attr': (21, 6), 'children': [{'children': [], 'val': 8}, {'children': [], 'val': 5}]}]}]}, {'attr': (16, 5), 'children': [{'attr': (18, 3), 'children': [{'children': [], 'val': 6}, {'attr': (14, 14), 'children': [{'children': [], 'val': 8}, {'children': [], 'val': 0}]}]}, {'attr': (16, 7), 'children': [{'attr': (9, 19), 'children': [{'attr': (19, 4), 'children': [{'children': [], 'val': 2}, {'children': [], 'val': 8}]}, {'attr': (19, 20), 'children': [{'children': [], 'val': 8}, {'children': [], 'val': 9}]}]}, {'attr': (12, 16), 'children': [{'attr': (10, 14), 'children': [{'children': [], 'val': 4}, {'children': [], 'val': 1}]}, {'attr': (20, 7), 'children': [{'attr': (16, 9), 'children': [{'children': [], 'val': 7}, {'children': [], 'val': 8}]}, {'children': [], 'val': 5}]}]}]}]}]}]}, {'attr': (20, 18), [OMITTED], {'children': [], 'val': 4}]}, {'attr': (19, 4), 'children': [{'children': [], 'val': 8}, {'attr': (18, 19), 'children': [{'children': [], 'val': 3}, {'children': [], 'val': 7}]}]}]}]}]}]}]}]}]}]} Validating... [7, 8, 1, 0, 3, 1, 4, 4, 2, 9, 0, 6, 9, 0, 1, 5, 7, 7, 6, 4, 9, 4, 6, 5, 9, 0, 7, 4, 0, 1, 3, 8, 3, 6, 7, 9, 7, 1, 5, 1, 1, 7, 9, 1, 4, 8, 1, 2, 4, 4, 6, 3, 7, 5, 9, 0, 4, 1, 4, 9, 9, 4, 9, 5, 5, 4, 8, 7, 3, 0, 7, 0, 6, 8, 1, 1, 3, 7, 9, 7, 9, 6, 2, 7, 4, 5, 7, 9, 4, 1, 3, 6, 9, 3, 1, 6, 1, 2, 6, 9] 63 correct out of 100 (63.0%). Testing... [9, 6, 2, 3, 8, 1, 4, 9, 1, 0, 4, 1, 7, 9, 4, 4, 6, 4, 8, 1, 3, 7, 9, 4, 4, 1, 5, 1, 3, 8, 1, 2, 4, 8, 0, 6, 2, 1, 1, 1, 1, 6, 3, 4, 4, 5, 9, 0, 7, 2, 4, 4, 8, 2, 1, 7, 7, 4, 9, 4, 5, 0, 9, 9, 2, 2, 3, 3, 4, 3, 5, 7, 5, 3, 2, 1, 2, 4, 9, 6, 4, 4, 5, 8, 0, 6, 9, 7, 9, 9, 9, 7, 0, 8, 0, 3, 7, 1, 3, 2] 67 correct out of 100 (67.0%).	data: digits classifier: id3 using enhanced features?: False training set size: 1000 Extracting features... Training... Prune: 63 to 64 Prune: 64 to 65 Prune: 65 to 66 Prune: 66 to 67 Prune: 67 to 68 {'attr': (14, 13), 'val': 7, 'children': [{'attr': (13, 17), 'val': 1, 'children': [{'attr': (11, 12), 'val': 1, 'children': [{'attr': (20, 15), 'val': 8, 'children': [{'attr': (17, 9), 'val': 6, 'children': [{'attr': (17, 20), 'val': 4, 'children': [{'attr': (22, 19), 'val': 3, 'children': [{'children': [], 'val': 3}, {'children': [], 'val': 8}]}, {'children': [], 'val': 4}]}, {'attr': (9, 15), 'val': 6, 'children': [{'children': [], 'val': 6}, {'attr': (21, 6), 'val': 8, 'children': [{'children': [], 'val': 8}, …, {'attr': (7, 16), 'val': 9, 'children': [{'attr': (17, 6), 'val': 4, 'children': [{'attr': (16, 11), 'val': 4, 'children': [{'children': [], 'val': 4}, {'attr': (19, 4), 'val': 9, 'children': [{'children': [], 'val': 0}, [OMITTED] {'children': [], 'val': 7}]},{'children': [], 'val': 3}]}, {'attr': (19, 10), 'val': 7, 'children': [{'attr': (19, 4), 'val': 7, 'children': [{'children': [], 'val': 1}, {'children': [], 'val': 7}]}, {'attr': (17, 20), 'val': 7, 'children': []}]}]}, {'attr': (16, 7), 'val': 6, 'children': [{'attr': (20, 10), 'val': 5, 'children': [{'attr': (12, 17), 'val': 7, 'children': [{'children': [], 'val': 9}, {'children': [], 'val': 7}]}, {'attr': (17, 20), 'val': 5, 'children': [{'children': [], 'val': 5}, {'children': [], 'val': 1}]}]}, {'attr': (12, 20), 'val': 6, 'children': [{'attr': (23, 6), 'val': 6, 'children': [{'children': [], 'val': 4}, {'children': [], 'val': 6}]}, {'attr': (8, 18), 'val': 4, 'children': []}]}]}]}]}]}]}]} Validating... 68 correct out of 100 (68.0%). Testing... 67 correct out of 100 (67.0%).

ID3 with training set of 3,000 without Pruning (69/76)	**ID3 with training set of 3,000 with Pruning (75/77)**
data: digits classifier: id3 using enhanced features?: False training set size: 3000 Extracting features... Training... {'attr': (13, 16), 'val': 1, 'children': [{'attr': (9, 13), 'val': 1, 'children': [{'attr': (10, 19), 'val': 4, 'children': [{'attr': (11, 23), 'val': 6, 'children': [{'attr': (13, 6), 'val': 8, 'children': [{'attr': (19, 10), 'val': 8, 'children': [{'children': [], 'val': 8}, {'attr': (13, 9), 'val': 8, 'children': [{'attr': (20, 7), 'val': 5, 'children': [{'children': [], 'val': 5}, {'children': [], 'val': 7}]}, {'attr': (17, 6), 'val': 8, 'children': [{'children': [], 'val': 8}, {'children': [], 'val': 2}]}]}]}, {'attr': (16, 17), 'val': 8, 'children': [{'attr': (12, 17), 'val': 5, 'children': [{'attr': (21, 11), 'val': 4, 'children': [{'attr': (10, 7), 'val': 7, 'children': [{'children': [], 'val': 7}, {'children': [], 'val': 9}]}, {'attr': (17, 20), 'val': 4, 'children': [{'children': [], 'val': 0}, {'children': [], 'val': 4}]}]}, {'children': [], 'val': 5}]}, {'attr': (19, 13), 'val': 8, 'children': [{'children': [], 'val': 8}, {'attr': (17, 20), 'val': 6, 'children': [{'children': [], 'val': 6}, {'children': [], 'val': 1}]}]}]}]}, {'attr': (14, 20), 'val': 6, 'children': [{'attr': (20, 8), 'val': 6, 'children': [{'attr': (13, 10), 'val': 2, 'children': [{'attr': (17, 6), 'val': 5, 'children': [{'attr': (11, 22), 'val': 8, 'children': [{'attr': (16, 9), 'val': 0, 'children': [{'children': [], 'val': 0}, {'children': [], 'val': 9}]}, {'children': [], 'val': 8}]}, ..., {'attr': (20, 25), 'val': 7, 'children': [{'attr': (17, 20), 'val': 2, 'children': [{'children': [], 'val': 2}, {'children': [], 'val': 7}]}, {'children': [], 'val': 7}]}]}, {'attr': (14, 11), 'val': 7, 'children': [{'attr': (10, 7), 'val': 9, 'children': [{'attr': (17, 20), 'val': 4, 'children': [{'children': [], 'val': 4}, {'children': [], 'val': 7}]}, {'attr': (18, 19), 'val': 9, 'children': [{'children': [], 'val': 3}, {'children': [], 'val': 9}]}]}, {'attr': (12, 18), 'val': 7, 'children': [{'attr': (20, 25), 'val': 3, 'children': [{'children': [], 'val': 3}, {'children': [], 'val': 1}]}, {'children': [], 'val': 7}]}]}]}]}]}]}]}]}]}]}]}]} Validating... 69 correct out of 100 (69.0%). Testing... 76 correct out of 100 (76.0%).	data: digits classifier: id3 using enhanced features?: False training set size: 3000 Extracting features... Training... Prune: 69 to 70 Prune: 70 to 71 Prune: 71 to 72 Prune: 72 to 73 Prune: 73 to 74 Prune: 74 to 75 {'attr': (13, 16), 'val': 1, 'children': [{'attr': (9, 13), 'val': 1, 'children': [{'attr': (10, 19), 'val': 4, 'children': [{'attr': (11, 23), 'val': 6, 'children': [{'attr': (13, 6), 'val': 8, 'children': [{'attr': (19, 10), 'val': 8, 'children': [{'children': [], 'val': 8}, {'attr': (13, 9), 'val': 8, 'children': [{'attr': (20, 7), 'val': 5, 'children': [{'children': [], 'val': 5}, {'children': [], 'val': 7}]}, {'attr': (17, 6), 'val': 8, 'children': [{'children': [], 'val': 8}, {'children': [], 'val': 2}]}]}]}, {'attr': (16, 17), 'val': 8, 'children': [{'attr': (12, 17), 'val': 5, 'children': [{'attr': (21, 11), 'val': 4, 'children': [{'attr': (10, 7), 'val': 7, 'children': [{'children': [], 'val': 7}, {'children': [], 'val': 9}]}, {'attr': (17, 20), 'val': 4, 'children': [{'children': [], 'val': 0}, {'children': [], 'val': 4}]}]}, {'children': [], 'val': 5}]}, {'attr': (19, 13), 'val': 8, 'children': [{'children': [], 'val': 8}, {'attr': (17, 20), 'val': 6, 'children': [{'children': [], 'val': 6}, {'children': [], 'val': 1}]}]}]}]}, {'attr': (14, 20), 'val': 6, 'children': [{'attr': (20, 8), 'val': 6, **[OMITTED]** , {'children': [], 'val': 0}]}, {'attr': (17, 20), 'val': 4, 'children': [{'children': [], 'val': 4}, {'children': [], 'val': 3}]}]}, {'children': [], 'val': 5}]}]}, {'attr': (13, 5), 'val': 7, 'children': [{'attr': (11, 20), 'val': 2, 'children': [{'children': [], 'val': 2}, {'attr': (17, 20), 'val': 7, 'children': [{'attr': (16, 9), 'val': 3, 'children': [{'children': [], 'val': 3}, {'children': [], 'val': 5}]}, {'children': [], 'val': 7}]}]}, {'attr': (19, 10), 'val': 7, 'children': [{'children': [], 'val': 9}, {'attr': (20, 25), 'val': 7, 'children': [{'attr': (17, 20), 'val': 2, 'children': [{'children': [], 'val': 2}, {'children': [], 'val': 7}]}, {'children': [], 'val': 7}]}]}, {'attr': (14, 11), 'val': 7, 'children': []}]}]}]}]}]}]}]}]} Validating... 75 correct out of 100 (75.0%). Testing... 77 correct out of 100 (77.0%).

naiveBayes with training set of 100 and k=2 (66/53)	**naiveBayes with training set of 100 and autoTunes option (75/64)**
data: digits classifier: naiveBayes using enhanced features?: False training set size: 100 using smoothing parameter k=2.000000 for naivebayes Extracting features... Training... Prior Distribution: {0: 0.13, 1: 0.14, 2: 0.06, 3: 0.11, 4: 0.11, 5: 0.05, 6: 0.11, 7: 0.1, 8: 0.08, 9: 0.11} Current k: 2.0, Rate of Correctness: 66% Final Selected k: 2.0, Maximum Rate of Correctness: 66% Validating... 66 correct out of 100 (66.0%). Testing... 53 correct out of 100 (53.0%).	data: digits classifier: naiveBayes using enhanced features?: False training set size: 100 Extracting features... Training... Prior Distribution: {0: 0.13, 1: 0.14, 2: 0.06, 3: 0.11, 4: 0.11, 5: 0.05, 6: 0.11, 7: 0.1, 8: 0.08, 9: 0.11} Current k: 0.001, Rate of Correctness: 72% Current k: 0.01, Rate of Correctness: 74% Current k: 0.05, Rate of Correctness: 75% Current k: 0.1, Rate of Correctness: 72% Current k: 0.5, Rate of Correctness: 72% Current k: 1, Rate of Correctness: 72% Current k: 5, Rate of Correctness: 43% Current k: 10, Rate of Correctness: 23% Current k: 20, Rate of Correctness: 17% Current k: 50, Rate of Correctness: 14% Final Selected k: 0.05, Maximum Rate of Correctness: 75% Validating... 75 correct out of 100 (75.0%). Testing... 64 correct out of 100 (64.0%).
naiveBayes with training set of 1,000 and k=2 (81/77)	**naiveBayes with training set of 1,000 and autoTunes option (82/78)**
data: digits classifier: naiveBayes using enhanced features?: False training set size: 1000 using smoothing parameter k=2.000000 for naivebayes Extracting features... Training... Prior Distribution: {0: 0.097, 1: 0.116, 2: 0.099, 3: 0.093, 4: 0.105, 5: 0.092, 6: 0.094, 7: 0.117, 8: 0.087, 9: 0.1} Current k: 2.0, Rate of Correctness: 81% Final Selected k: 2.0, Maximum Rate of Correctness: 81% Validating... 81 correct out of 100 (81.0%). Testing... 77 correct out of 100 (77.0%).	data: digits classifier: naiveBayes using enhanced features?: False training set size: 1000 using automatic tuning for naivebayes Extracting features... Training... Prior Distribution: {0: 0.097, 1: 0.116, 2: 0.099, 3: 0.093, 4: 0.105, 5: 0.092, 6: 0.094, 7: 0.117, 8: 0.087, 9: 0.1} Current k: 0.001, Rate of Correctness: 82% Current k: 0.01, Rate of Correctness: 81% Current k: 0.05, Rate of Correctness: 80% Current k: 0.1, Rate of Correctness: 79% Current k: 0.5, Rate of Correctness: 79% Current k: 1, Rate of Correctness: 79% Current k: 5, Rate of Correctness: 78% Current k: 10, Rate of Correctness: 76% Current k: 20, Rate of Correctness: 73% Current k: 50, Rate of Correctness: 57% Final Selected k: 0.001, Maximum Rate of Correctness: 82% Validating... 82 correct out of 100 (82.0%). Testing... 78 correct out of 100 (78.0%).

D. References

1. [Book] Machine_Learning by Tom_Mitchell
2. http://decisiontrees.net/decision-trees-tutorial/tutorial-14-pruning/
3. http://decisiontrees.net/decision-trees-tutorial/tutorial-15-exercise-6/
4. http://jsbeautifier.org/