Final Project - Comprehensive Classifier Creation

Sam Neyhart
Jon Clark Freeman
Kevin Sayarath
ECE 471 - Professor Qi

April 23, 2017

Abstract

The objective of this final project was to integrate various classification techniques to achieve the best performance on the chosen dataset. This project was a group effort where we explored both supervised and unsupervised classificaiton techniques including MPP (all 3 cases), kNN (with different k's), BPNN (developed), decision tree, and k-means, winner-take-all, and kohonen maps. We also explored two classifier fusion techniques including majority vote fusion and naive Bayes.

Introduction

This project performed for ECE 471 is an exercise using all the classificaiton methods we have used in past projects with the additional exploration of two new fusion methods which we previously had no experience with. The dataset we are using is titled "Poker Hand Dataset," publised by Robert Cattrel et. al., released in January 2007 *link to dataset*. By experimenting with various classification techniques, we learn the best specific methodology to classify this particular dataset.

Overall the project served its purpose and was an opportunity to review and become experts with the classification methods learned during this course. We are proud to announce we did not use any supporting libraries that do heavy-lifting of core computations, i.e. all classification methods were re-written by team members for this project.

Overall the project served its purpose and was an opportunity to learn much about the concepts of this course. For this lab the numpy library was very helpful.

We used a github repository to collaborate on this project:
https://github.com/sneyhart/swagteam6

Team Member Contributions

Clark

Mpp
    Knn
    BPNN
    Decision Tree
    Report Writing

Sam

Kevin

Normalization
    Evaluation
    Team organization
    Report writing

Dataset

The dataset we chose consists of all possible permutations of five-card poker hands dealt from a standard deck of 52 cards. Each sample is one hand and each card is represented by two features (suit and rank), for a total of 10 predictive attributes plus one class attribute. As previously mentioned, the order of cards is mportant, which is why there are 480 possible Royal Flush hands as compared to only 4 in the combination set.

From poker-hand.names

Attribute Information:
1) S1 Suit of card 1
    Ordinal (1-4) representing Hearts, Spades, Diamonds, Clubs
2) C1 Rank of card 1
    Numerical (1-13) representing (Ace, 2, 3, ... , Queen, King)
3) S2 Suit of card 2
    Ordinal (1-4) representing Hearts, Spades, Diamonds, Clubs
4) C2 Rank of card 2
    Numerical (1-13) representing (Ace, 2, 3, ... , Queen, King)
5) S3 Suit of card 3
    Ordinal (1-4) representing Hearts, Spades, Diamonds, Clubs
6) C3 Rank of card 3
    Numerical (1-13) representing (Ace, 2, 3, ... , Queen, King)
7) S4 Suit of card 4
    Ordinal (1-4) representing Hearts, Spades, Diamonds, Clubs
8) C4 Rank of card 4
    Numerical (1-13) representing (Ace, 2, 3, ... , Queen, King)
9) S5 Suit of card 5
    Ordinal (1-4) representing Hearts, Spades, Diamonds, Clubs
10) C5 Rank of card 5
    Numerical (1-13) representing (Ace, 2, 3, ... , Queen, King)
11) CLASS Poker Hand
    Ordinal (0-9)

Class Information:
0: Nothing in hand; not a recognized poker hand
1: One pair; one pair of equal ranks within five cards
2: Two pairs; two pairs of equal ranks within five cards
3: Three of a kind; three equal ranks within five cards
4: Straight; five cards, sequentially ranked with no gaps
5: Flush; five cards with the same suit
6: Full house; pair + different rank three of a kind
7: Four of a kind; four equal ranks within five cards
8: Straight flush; straight + flush
9: Royal flush; Ace, King, Queen, Jack, Ten + flush

N: 25,010 training, 1,000,000 testing

Technical Approach

MPP   The mpp.py Python script implements the MPP algorithm using basic Python, the numpy library, and the matplotlib library. It performs MPP parametric-based classification by first calculating the mean and covariance matricies from the dataset. *Bayes/Discriminant Func*

Case 1   The features are statistically independent, and have the same variance. Geometrically, the samples fall in equal-size hyperspherical clusters. Decision boundary: hyperplane of d-1 dimension. This classification technique employs the linear discriminant function and linear machine. Additionally, when prior probabilities are the same, the discriminant function is actually measuring the minimum distance from each feature to each of the c mean vectors.

Case 2   The covariance matrices for all the classes are identical but not a scalar of identity matrix. Geometrically, the samples fall in hyperellisoidal clusters. Decision boundary: hyperplane of d-1 dimension.

Case 3   No assumption: the covariance matrices are different for each class. Quadratic classifier. Decision boundary: hyperquadratic for 2-D Gaussian.

kNN   The knn.py Python script implements the kNN algorithm usign basic Python, the numpy library, and the scipy library. It performs kNN classification, or majority voting, using Euclidean distance to assign a random sample according to the majority representation of classes within the enclosing hypersphere of k nearest neighbors. We experiment with different k's and evaluate performance.

BPNN   The backprop.py script implements the backpropogation algorithm using basic Python, the numpy library, and the random library. Backpropogation is a type of multilayer feedforward network that calculates the difference between unit output and expected output, "back-propogating" this delta from the output layers back toward the feeding layers. This technique causes the network to "learn" correct classifictions in a supervised architecture. The script uses a network with 2 hidden layes. Each hidden layer has 15 neurons in each layer. The input layers change based on if the PCA preprocessing has been done on the input data. The output for the network is one neuron for the classification.

Decision Tree   The dtree.py script implements the decision tree architecture using basic Python and the numpy library. The decision tree is structured to decide the various classes in an increasing faction. The highest valued class that is decided is the resulting classification of that hand. This is very efficient because the features of the data set are such that each class is made up of a

specific set of cards. Thus each node will test the hand and classify it into that hand.

K-means The *kmeans.py* file impliments the k-means algorithm using basic python and the numpy library. It will perform k-means clustering on the data and then subsequently output the mean values for the requested number of clusters. This implementation is incredibly slow however so in order to speed the process of obtaining the various versions of the *flowers.ppm* image required, the Project4.py file was created.

**Winner-Take-All** This project makes use of the file *winner.py* which contains instructions which implement winner-takes-all clustering. This algorithm is much faster than the k-means algorithm. The winner-take-all clustering algorithm is initialized with random cluster centers which can result in differing final values.

Kohonen Maps SOMETHING

Experiments and Results

MPP

kNN

Average BPNN Accuracy
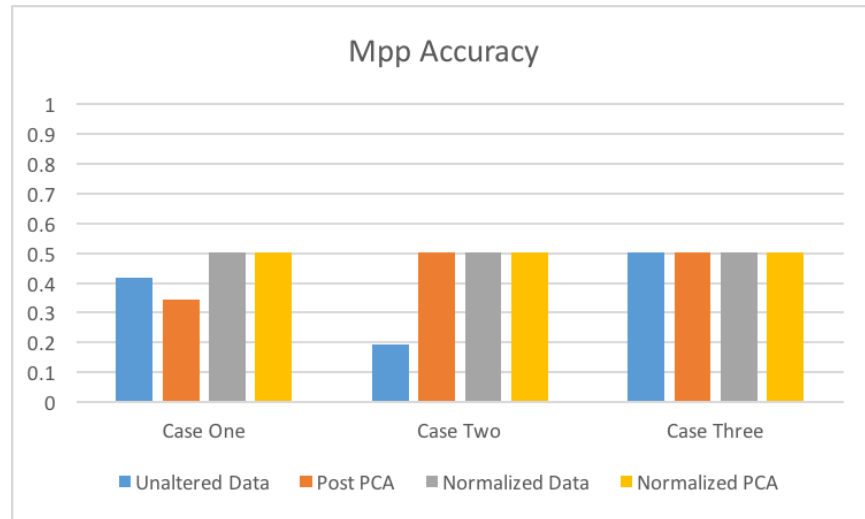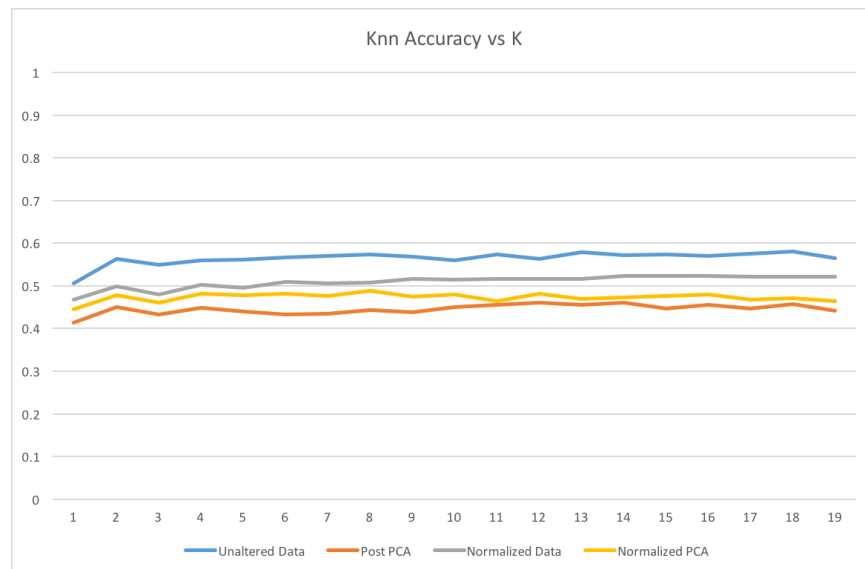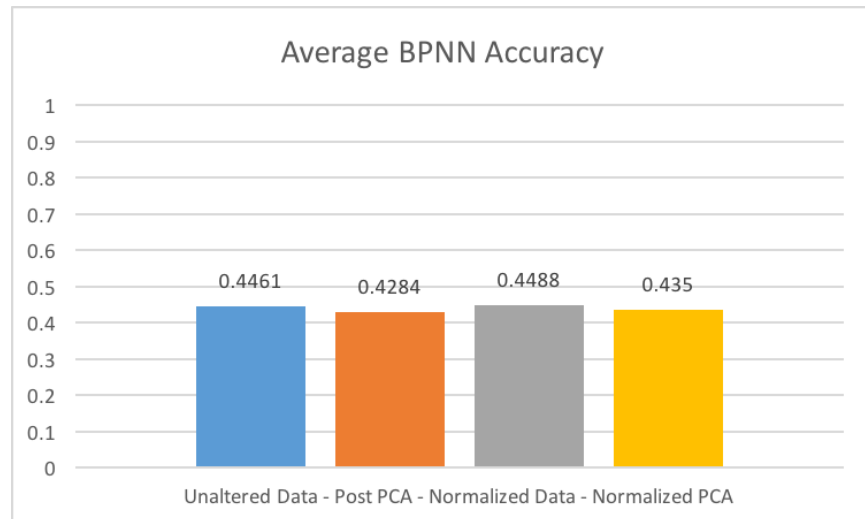
BPNN

Decision Tree The Decision tree algorithm was actually able to classify with an accuracy of one hundred percent when run on the testing set. The algorithm simply ran through each of the hands and had a node that decided for each of the groups since each group is defined in such a way that is known before hand and has to be a certain combination of the cards. This lead the algorithm to be very fast as well since it only goes through the data set one time to classify each hand.

K-means

Winner-Take-All

Kohonen Maps The original image *flowers.ppm* is displayed above. All experiments are performed on this image. The first expiriment is the kmeans clustering of colors in *flowers.png*. The resulting images can be seen on the following page. To generate these images k-means clustering was performed with [1, 2, 4, 8, 16 , 32, 64, 128, 256] clusters respectively.

The k-means images are seen above. The visual differences on these images are easy to see for the first couple images but by k=16 it becomes harder to find difference between the original image and the processed image. A list of the colors included in the each of these images is found in the apendix.

Evaluation

Evalution was performed using 10-fold cross-validation for the kNN classifier. Evaluation is performed in evaluation.py using standard Python language, with the support of the math, random, numpy, and subprocess libraries. This script performes the m-fold cross-validation algorithm by dividing the dataset into m disjoint sets and iterates over them using the 'leave-one-out' methodology, where one set is preserved as the testing (or validaiton) set and the remaining m-1 sets are combined to become the new training set. Normalization is performed anew on each of the m iterations for the training and testing sets separately.
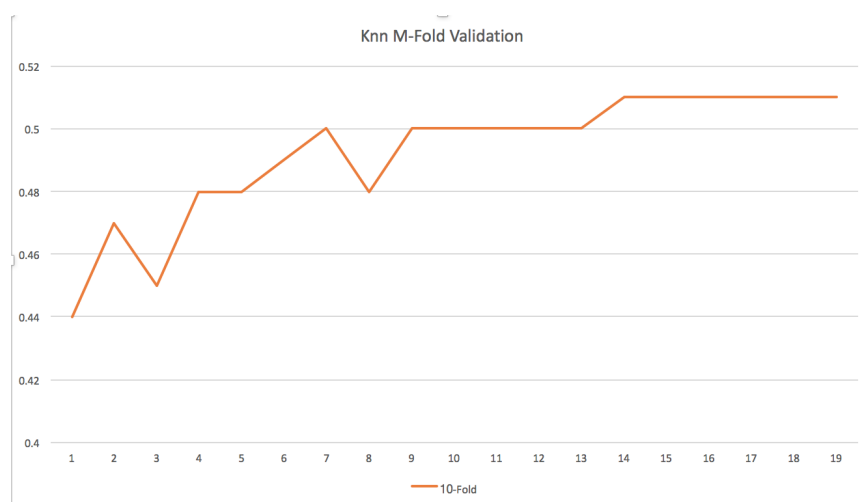
kNN with 10-fold validation has a similar profile to the previous kNN trials performed with generally lower accuracy due to near complete mitigation of training/testing cycle bias.

Discussion

This project was very interesting to perform. Employing all of the classifiers that the course has covered proved to be a difficult yet enjoyable exercise. Using python was a good choice for most of the programs because of the simplicity of programming in Python as well as the ever useful libraries provided in the language. The clustering programs were written in c because of the size of the data set. C performed much faster than Python would have thus the group was able to run more tesst to glean a reasonable understanding of how well the programs performed. This was a very rewarding project to work on for the team.

Appendix

normalize.py

```
 1  # Normalizaiton
 2  # Input/Output file names defined in main funciton
 3  # Running simply normalizes data and writes to output file
 4  # Handy data separation techniques used
 5
 6  import math
 7  import numpy as np
 8  from numpy import linalg as LA
 9
10  DEBUG = 0
11
12  def getInputSeparated(fname):
13          #fname = 'poker-hand-training-true.data'
14          file = open(fname, 'r')                              # read file
15          #with open(fname) as f:
16                  #lines = f.read().splitlines
17          lines = file.readlines()
18          file.close()
19          classData = []
20          class0Data = []                                      # can probably do this n
21          class1Data = []
22          class2Data = []
23          class3Data = []
24          class4Data = []
25          class5Data = []
26          class6Data = []
27          class7Data = []
28          class8Data = []
29          class9Data = []
30          for line in lines:                                   # read elements
31                  line = line.strip()
32                  splitLine = line.split(',')
33                  classIndex = len(splitLine)-1   # max index of splitLine
34                  #print splitLine
35                  #print 'Class Index: {}'.format(classIndex)
36                  #print splitLine[classIndex]
37                  sampleData = []
38                  #sampleData.append(int(splitLine[:lenIndexSplitLine]))
39                  for j in range(classIndex):
40                          #print j
41                          sampleData.append(float(splitLine[j]))
42                  # Separate by classes
```

14

```python
43                            if (splitLine[classIndex] == '0'):
44                                    #print 'appending 0'
45                                    class0Data.append(sampleData)
46                            elif(splitLine[classIndex] == '1'):
47                                    class1Data.append(sampleData)
48                            elif(splitLine[classIndex] == '2'):
49                                    class2Data.append(sampleData)
50                            elif(splitLine[classIndex] == '3'):
51                                    class3Data.append(sampleData)
52                            elif(splitLine[classIndex] == '4'):
53                                    class4Data.append(sampleData)
54                            elif(splitLine[classIndex] == '5'):
55                                    class5Data.append(sampleData)
56                            elif(splitLine[classIndex] == '6'):
57                                    class6Data.append(sampleData)
58                            elif(splitLine[classIndex] == '7'):
59                                    class7Data.append(sampleData)
60                            elif(splitLine[classIndex] == '8'):
61                                    class8Data.append(sampleData)
62                            elif(splitLine[classIndex] == '9'):
63                                    class9Data.append(sampleData)
64              classData.append(np.asarray(class0Data))
65              classData.append(np.asarray(class1Data))
66              classData.append(np.asarray(class2Data))
67              classData.append(np.asarray(class3Data))
68              classData.append(np.asarray(class4Data))
69              classData.append(np.asarray(class5Data))
70              classData.append(np.asarray(class6Data))
71              classData.append(np.asarray(class7Data))
72              classData.append(np.asarray(class8Data))
73              classData.append(np.asarray(class9Data))
74              return np.asarray(classData)
75
76   def test_getInputSeparated():
77           print 'Entered test_getInputSeparated()'
78           classData = getInputSeparated('poker-hand-training-true.txt')
79           i = -1
80           for classSamples in classData:
81                   i += 1
82                   print 'Class {} data:'.format(i)
83                   for classSample in classSamples:
84                           print classSample
85
86   def test_getInputSeparated2():
87           print 'Entered test_getInputSeparated2()'
88           classData = getInputSeparated('poker-hand-training-true.txt')
```

```
 89                print classData
 90
 91  def weave(classData):
 92          weaved = []
 93          largestClassSampCount = 0
 94          # find the length of the longest list (the number of samples in the larg
 95          for classSamples in classData:
 96                  numClassSamples = len(classSamples)
 97                  if numClassSamples > largestClassSampCount:
 98                          largestClassSampCount = numClassSamples
 99          # weave samples across all classes
100          # Kevin − I can explain this, just ask me
101          for i in range(largestClassSampCount):
102                  for classSamples in classData:
103                          if (i < len(classSamples)):
104                                  weaved.append(classSamples[i])
105          return weaved
106
107  def test_weave():
108          fName_tr = 'poker−hand−training−true.data'
109          # Get training data
110          X0 = getInputSeparated(fName_tr)
111          X = weave(X0)
112          for samp in X:
113                  print samp
114
115  def getColumnMeans(X):
116          return np.mean(X, axis=0)
117
118  def getColumnStddevs(X):
119          return np.std(X, axis=0)
120
121  def getColMeansSeparated(_XS):
122          colMeansSep = []
123          for classSamples in _XS:
124                  colMeansSep.append(getColumnMeans(classSamples))
125          return np.asarray(colMeansSep)
126
127  def getColumnStddevsSeparated(_XS):
128          colMeansSep = []
129          for classSamples in _XS:
130                  colMeansSep.append(getColumnStddevs(classSamples))
131          return np.asarray(colMeansSep)
132
133  def normalize(X, colMeans, colStddevs):
134          nX = []
```

```
135                for sample in X:
136                        nRow = sample
137                        for i in range(len(nRow)):
138                                nRow[i] = (sample[i] - colMeans[i])/colStddevs[i]
139                        nX.append(nRow)
140                return np.asarray(nX)
141
142    def normalizeSep(XS, colMeans_all_tr, colStddevs_all_tr):
143            nXS = []
144            for classSamples in XS:
145                    nXClass = normalize(classSamples, colMeans_all_tr, colStddevs_al
146                    nXS.append(nXClass)
147            return np.asarray(nXS)
148
149    def rebuild(nXS, fname):
150            nOriginalData = open(fname, 'w')
151            i = -1
152            for nClassSamples in nXS:
153                    i += 1
154                    for nSamp in nClassSamples:
155                            nSampString = ''
156                            for feature in nSamp:
157                                    nSampString += str(feature) + ',␣'
158                            nSampString += str(i) + '\n'
159                            nOriginalData.write(nSampString)
160                            #print nSampString
161            nOriginalData.close()
162
163    def main():
164            if DEBUG: print 'Entered␣main()'
165            #fName_IN = 'poker-hand-training-true.data'
166            fName_IN = 'poker-hand-testing.data'
167            #fName_OUT = 'poker-hand-training-true-normalized.data'
168            fName_OUT = 'poker-hand-testing-normalized.data'
169
170            # 'X' denotes original dataset
171            # 'XS' denotes original dataset separated by class (hand type)
172            XS = getInputSeparated(fName_IN)
173            #if DEBUG: print 'XS: \n{}'.format(XS)
174            X = weave(XS)
175
176            if DEBUG: print 'X:\n{}\n\n\n'.format(X)
177            if DEBUG: print 'XS:\n{}\n\n\n'.format(XS)
178
179            # Compute un-normalized training mu & sigma for normalization
180            colMeans_all_tr = getColumnMeans(X)
```

```
181            colStddevs_all_tr = getColumnStddevs(X)
182
183        # 'nX' denotes original dataset - normalized
184        # 'nXS' denotes original dataset, separated - normalized
185        nXS = normalizeSep(XS, colMeans_all_tr, colStddevs_all_tr)
186        nX = weave(nXS)                                    # nX is both classes com
187
188        if DEBUG: print 'nX:\n{}\n\n\n'.format(nX)
189        #if DEBUG:
190        print 'nXS:\n{}'.format(nXS)
191
192        rebuild(nXS, fName_OUT)
193
194
195    main()
```

mpp.py

```
1   ###################
2   # Jon Clark Freeman, Sam Neyhart, Kevin Sayarath
3   # mpp.py classifies using the three cases of mpp
4   ###################
5
6   import matplotlib.pyplot as plt
7   import numpy as np
8
9   #function that returns the likelihood for a given [x, y], mean, and covariance n
10  def cmv(x, mean, cov, prior):
11      pre_top = 1;
12      pre_bottom = (2 * np.pi) * np.sqrt(np.linalg.det(cov))
13      post_matrix_mult = np.dot(np.dot(np.transpose(np.subtract(np.array(x), mean)
14      post_scalar = -1.0/2.0
15      return (pre_top/pre_bottom) * np.exp(post_scalar * post_matrix_mult) * prior
16
17  #function that returns Case 1 Discriminant
18  def case_1(x, mean, covariance, prior):
19      return -np.dot(np.subtract(np.array(x), mean), np.transpose(np.subtract(np.ar
20
21  #function that returns Case 2 Discriminant
22  def case_2(x, mean, cov, prior):
23      return -np.dot(np.dot(np.transpose(np.subtract(np.array(x), mean)), np.linal
24
25  #initialization of priors and glasses
26  priors = [0.501177,0.422569,0.047539,0.021128,0.003925,0.001965,0.001441,0.00024
27  classes = [[] for i in range(0,10)]
28
29  # Handle file manipulation
30  lines = [np.array((file_line.rstrip('\n').split(','))).astype(float) for file_lin
31  lines = np.array(lines)
32
33  #handle getting covariance matrix and means
34  cov_lines = np.delete(lines.T, lines.T.shape[0] - 1,0)
35  for line in lines:
36      classes[int(line[-1])].append(np.delete(line,(line.shape[0]-1),0))
37  classes = np.array([np.array(classer) for classer in classes])
38  means = [np.mean(row) for row in classes]
39  covariance = np.cov(cov_lines)
40
41  testing_lines = [np.array((file_line.rstrip('\n').split(','))).astype(float) for
42  testing_lines = np.array(testing_lines)
43  tests = np.delete(testing_lines.T, testing_lines.T.shape[0] - 1,0).T
44  actual_classes = testing_lines.T[-1]
```

```python
45
46   choices = []
47
48   for t, test in enumerate(tests):
49       max_disc = -1
50       max_val = -1000000
51       for i in range(0, len(means)):
52           res = cmv(test, means[i], covariance, priors[i])
53           if res > max_val:
54               max_disc = i
55               max_val = res
56       choices.append(max_disc)
57
58   corrects = 0
59   total = 0
60   for i in range(0, len(choices)):
61       if choices[i] == int(actual_classes[i]):
62           corrects = corrects + 1
63       total = total + 1
64   fusout = open("fusion-files-majority/mpp.data", 'w')
65   for choice in choices:
66       fusout.write("{}\n".format(int(choice)))
```

knn.py

```python
1  import numpy as np
2  from scipy import spatial as sp
3
4  #Preprocessing
5  lines = []
6  tlines = []
7  data = []
8  groups = []
9  classes = []
10 dtp = []
11 knn_res = []
12 corrects = [0] * 20
13 #file IO
14
15 lines = [np.array((file_line.rstrip('\n').split(','))).astype(float) for file_lin
16 lines = np.array(lines)
17 training = np.delete(lines.T, lines.T.shape[0] - 1,0).T
18 res = lines.T[-1]
19
20 testing_lines = [np.array((file_line.rstrip('\n').split(','))).astype(float) for
21 testing_lines = np.array(testing_lines)
22 tests = np.delete(testing_lines.T, testing_lines.T.shape[0] - 1,0).T
23 actual_classes = testing_lines.T[-1]
24
25
26 #testing each unit in the testing set
27 fusout = open("fusion-files-majority/knn_un.data",'w')
28 for u, unit in enumerate(tests):
29     if u % 150 == 0:
30         print "{}%".format(u/150*10)
31     dis = []
32     cords = unit
33     actual = actual_classes[u]
34     for point in training:
35         dis.append(np.linalg.norm(point - cords))
36     idx = np.argsort(dis, axis=0)
37     dis = np.array(dis)[idx]
38     results = np.array(res)[idx]
39     counts = [0,0,0,0,0,0,0,0,0,0]
40
41 #going through each k value after each distance is calculated
42     for k in range(1,20):
43         mmax = -1
44         mindex = -1
```

```
45                counts[int(results[k-1])] = counts[int(results[k-1])] + 1
46
47            for ind,count in enumerate(counts):
48                if mmax < count:
49                    mmax = count
50                    mindex = ind
51            if k == 17:
52                fusout.write("{}\n".format(mindex))
53            if(actual == mindex):
54                corrects[k] += 1
55  # counting the correct guesses per each k value
56  corrects = [(correct * 1.0)/len(tests) for correct in corrects]
57  knn_res.append(corrects)
58  knn_res_tp = np.transpose(knn_res)
59  knn_res_tp = [np.mean(col) for col in knn_res_tp]
60  knn_res = np.transpose(knn_res_tp)
61  #print out the results for each k value up to the max k value
62  for i, k in enumerate(knn_res):
63      print "{},{}".format(i, k)
```

backprop.py

```
1   ### backprop.py
2   ### Jon Clark Freeman
3   ### learns and tests XOR logic
4   import numpy as np
5   import numpy.random as rando
6   import random
7
8   class BPNN:
9       #initialize the network
10      def __init__(self, network):
11          self.weights = []
12
13          #weights for hidden and input
14          for i in range(1, len(network) - 1):
15              self.weights.append(2*rando.random((network[i-1] + 1, network[i] + 1
16
17          self.weights.append(2*rando.random((network[i] + 1, network[i+1])) - 1)
18
19      def activation(self, x):
20          return np.tanh(x)
21
22      def activation_deriv(self, x):
23          return 1.0 - x**2
24
25      def train(self, inp, target, learning_rate=0.01, trials=10000):
26          #Add bias units as a col of ones on the inputs
27          ones = np.atleast_2d(np.ones(inp.shape[0]))
28          x = np.concatenate((ones.T, inp), axis=1)
29          for j in range(trials):
30              #choose a random input to train with
31              i = np.random.randint(inp.shape[0])
32              activations = [x[i]]
33
34              #calculating the activations for each of the layers
35              for k in range(len(self.weights)):
36                  dot_value = np.dot(activations[k], self.weights[k])
37                  act = self.activation(dot_value)
38                  activations.append(act)
39
40              # calculating error from outputs and targets
41              error = target[i] - activations[-1]
42              deltas = [error * self.activation_deriv(activations[-1])]
43
44              # Calculating deltas starting at the hidden layer
```

```
45                      for k in range(len(activations) − 2, 0, −1):
46                          deltas.append(deltas[−1].dot(self.weights[k].T)*self.activation_
47                      deltas.reverse()
48
49                      # updating the weights using the deltas, activations
50                      for i in range(len(self.weights)):
51                          layer = np.atleast_2d(activations[i])
52                          delta = np.atleast_2d(deltas[i])
53                          self.weights[i] += learning_rate * layer.T.dot(delta)
54
55          #test on all of the inputs
56          def test(self, inp):
57              retval = np.concatenate((np.ones(1).T, np.array(inp)), axis=1)
58              for i in range(0, len(self.weights)):
59                  retval = self.activation(np.dot(retval, self.weights[i]))
60              return retval
61
62  #code to test and train using the class above
63  nn = BPNN([10,15,15,1])
64
65  lines = [np.array((file_line.rstrip('\n').split(','))).astype(float) for file_lir
66  lines = np.array(lines)
67  training = np.delete(lines.T, lines.T.shape[0] − 1,0).T
68  res = lines.T[−1]
69
70  testing_lines = [np.array((file_line.rstrip('\n').split(','))).astype(float) for
71  testing_lines = np.array(testing_lines)
72  tests = np.delete(testing_lines.T, testing_lines.T.shape[0] − 1,0).T
73  actual_classes = testing_lines.T[−1]
74
75  nn.train(training, res)
76  corrects = 0.0
77  total = 0.0
78  fusout = open("fusion−files−majority/bpnn.data",'w')
79  for t,inpu in enumerate(tests):
80      total = total + 1
81      fusout.write("{}\n".format(int(round(nn.test(inpu)[0]))))
82      if int(round(nn.test(inpu)[0])) == actual_classes[t]:
83          corrects = corrects + 1
84  print "Accuracy −− {}".format(corrects/total)
```

dtree.py

```
'''
dtree.py
decision tree classifier
has 9 nodes to check for each possible hand
'''
#file IO
import numpy as np
testing_lines = [np.array((file_line.rstrip('\n').split(','))).astype(int) for file_line in
testing_lines = np.array(testing_lines)
tests = np.delete(testing_lines.T, testing_lines.T.shape[0] - 1,0).T
actual_classes = testing_lines.T[-1]

#initialize
corrects = 0.0
total = 0.0

#each testing
for t,test in enumerate(tests):
    #get info sorted into suits and ranks
    total = total + 1
    ranks = []
    suits = []
    for i in range(0,10,2):
        suits.append(test[i])
        ranks.append(test[i+1])
    matches = [0]
    idx = np.argsort(ranks, axis=0)
    ranks = np.array(ranks)[idx]
    suits = np.array(suits)[idx]
    ranks = ranks.tolist()
    suits = suits.tolist()

    #is pair?
    ph = 0
    for val in ranks:
        if ranks.count(val) == 2:
            ph = 1
    if ph == 1:
        matches.append(1)

    #is two pair?
    ph = 0
    first = 0
    for val in ranks:
```

```
            if ranks.count(val) == 2:
                if first == 0:
                    first = val
                elif first != val:
                    ph = 1
    if ph == 1:
        matches.append(2)

    #is three of a kind?
    ph = 0
    for val in ranks:
        if ranks.count(val) == 3:
            ph = 1
    if ph == 1:
        matches.append(3)

    #is straight?
    ph = 1
    if ranks.count(1) == 1 and ranks[-1] == 13:
        for i,val in enumerate(ranks):
            if i != 0 and i != 1:
                if ranks[i] - ranks[i - 1] != 1:
                    ph = 0
    else:
        for i,val in enumerate(ranks):
            if i != 0:
                if ranks[i] - ranks[(i - 1)] != 1:
                    ph = 0
    if ph == 1:
        matches.append(4)


    #is flush?
    if suits.count(suits[0]) == 5:
        matches.append(5)

    #is full house?
    if 1 in matches and 3 in matches:
        matches.append(6)

    #is four of a kind?
    ph = 0
    for val in ranks:
        if ranks.count(val) == 4:
            ph = 1
    if ph == 1:
```

```python
            matches.append(7)

        #is straight flush?
        if 4 in matches and 5 in matches:
            matches.append(8)

        #is royal flush?
        if 8 in matches and ranks[0] == 1 and ranks[-1] == 13:
            matches.append(9)

        #checking result
        result = matches[-1]
        if result == actual_classes[t]:
            corrects = corrects + 1
print "Accuracy {}".format(corrects/total)
```

kmeans.c

```
/*
 * kmeans.c
 * Sam Neyhart
 * This file contains the kmeans program used in the final project of
 * ECE 471.
 * USAGE: kmeans filename.data nclusters
 */

#include<string.h>
#include<time.h>
#include<stdio.h>
#include<math.h>
#include<stdlib.h>
#include"hand.h"


static const int ITERATIONS = 1; //How many times to go through data
static const int NCLASSES = 10;
static const int NCARDS = 10;
static int NCLUSTERS;
static int NLINES = 0;

int main(int argc, char ** argv)
{
//----------File IO---------
FILE * in;
in = fopen(argv[1],"r");
if (argc != 3) return 0;
NCLUSTERS = atoi(argv[2]);
//Get numLines
char buf[255];
while(fgets(buf,255,in) != NULL){
NLINES++;
}
rewind(in);
//create arrays
hand * data = malloc(NLINES*sizeof(hand));
hand * mu = malloc(NCLUSTERS*sizeof(hand));
int * clas = malloc(NLINES*sizeof(int));
//fill data
for(int i = 0; i < NLINES; i++){
char * tmp;
fgets(buf,255,in);
tmp = strtok(buf,",");
```

```c
for(int j = 0; j < NCARDS; j++){
data[i].cards[j] = atof(tmp);
tmp = strtok(NULL,",");
}
clas[i]=atoi(tmp);
}
//randomly fill mu
time_t t;
srand((unsigned)time(&t));
for(int i = 0; i < NCLUSTERS; i++){
int tmp = rand() % NLINES;
for(int j = 0; j < NCARDS; j++){
mu[i].cards[j] = data[tmp].cards[j];
}
}

//---------Perform Kmeans----------
int *clus1 = malloc(NLINES*sizeof(int));
int *clus2 = malloc(NLINES*sizeof(int));
int flip = 0;
int done = 0;
int it = 0;
while (done==0){
//printf("iteration %d\n",it);
if (flip==0){
cluster(data,mu,NLINES,NCLUSTERS,clus1);
avg(mu, data, clus1,NLINES,NCLUSTERS);
flip++;
}else{
cluster(data,mu,NLINES,NCLUSTERS,clus2);
avg(mu, data, clus2,NLINES,NCLUSTERS);
flip--;
}
if (clus_equal(clus1,clus2,NLINES))
done++;
it++;
}

//-----Deterimine Accuracy-----
hand *avgclas = calloc(NCLASSES,sizeof(hand));
hand *avgclus = calloc(NCLUSTERS,sizeof(hand));
avg(avgclas, data, clas, NLINES, NCLASSES);
avg(avgclus, data, clus1, NLINES, NCLUSTERS);
printf(
"---------------------------For Number of clusters %d"
"---------------------------\n",NCLUSTERS);
```

```c
printf("Number of Iterations for Kmeans: %d\n",it);
printf("Avg Clusters:\n");
for(int i = 0; i < NCLUSTERS; i++){
for(int j = 0; j < NCARDS; j++){
printf("%lf ",avgclus[i].cards[j]);
}
printf("\n");
}
printf("\nAvg Classes:\n");
for(int i = 0; i < NCLASSES; i++){
for(int j = 0; j < NCARDS; j++){
printf("%lf ",avgclas[i].cards[j]);
}
printf("\n");
}
double acc;
acc = accuracy(NLINES,NCLUSTERS,NCLASSES,clus1,clas,avgclus,avgclas);
printf("\nOverall Accuracy is %lf\n\n",acc);

//---------Cleanup----------
free(data);
free(mu);
free(clas);
free(avgclas);
free(avgclus);
free(clus1);
free(clus2);
fclose(in);
}
```

wta.c

```
1   /*
2    * wta.c
3    * Sam Neyhart
4    * This file contains the wta program used in the final project of
5    * ECE 471.
6    * USAGE: wta filename.data nclusters
7    */
8
9   #include<string.h>
10  #include<time.h>
11  #include<stdio.h>
12  #include<math.h>
13  #include<stdlib.h>
14  #include"hand.h"
15
16
17  static const int ITERATIONS = 20; //How many times to go through data
18  static const int NCLASSES = 10;
19  static const int NCARDS = 10;
20  static int NCLUSTERS;
21  static int NLINES = 0;
22
23  int main(int argc, char ** argv)
24  {
25          //—————————File IO—————————
26          FILE * in;
27          in = fopen(argv[1],"r");
28          if (argc != 3) return 0;
29          NCLUSTERS = atoi(argv[2]);
30          //Get numLines
31          char buf[255];
32          while(fgets(buf,255,in) != NULL){
33                  NLINES++;
34          }
35          rewind(in);
36          //create arrays
37          hand * data = malloc(NLINES*sizeof(hand));
38          hand * mu = malloc(NCLUSTERS*sizeof(hand));
39          int * clas = malloc(NLINES*sizeof(int));
40          //fill data
41          for(int i = 0; i < NLINES; i++){
42                  char * tmp;
43                  fgets(buf,255,in);
44                  tmp = strtok(buf,",");
```

```
45                  for(int j = 0; j < NCARDS; j++){
46                          data[i].cards[j] = atof(tmp);
47                          tmp = strtok(NULL,",");
48                  }
49                  clas[i]=atoi(tmp);
50          }
51          //randomly fill mu
52          time_t t;
53          srand((unsigned)time(&t));
54          for(int i = 0; i < NCLUSTERS; i++){
55                  int tmp = rand() % NLINES;
56                  for(int j = 0; j < NCARDS; j++){
57                          mu[i].cards[j] = data[tmp].cards[j];
58                  }
59          }
60
61          //---------Perform WTA---------
62          for(int i = 0; i < ITERATIONS; i++){
63                  for(int j=0; j<NLINES; j++){
64                          update_mu(&data[j],mu,NCLUSTERS);
65                  }
66          }
67
68          //-----Deterimine Accuracy-----
69          int *clus = malloc(NLINES*sizeof(int));
70          cluster(data,mu,NLINES,NCLUSTERS,clus);
71          hand *avgclas = calloc(NCLASSES,sizeof(hand));
72          hand *avgclus = calloc(NCLUSTERS,sizeof(hand));
73          avg(avgclas, data, clas, NLINES, NCLASSES);
74          avg(avgclus, data, clus, NLINES, NCLUSTERS);
75          printf(
76                          "-----------------------------------For_Number_of_clusters_%d"
77                          "-----------------------------------\n",NCLUSTERS);
78          printf("Avg_Clusters:\n");
79          for(int i = 0; i < NCLUSTERS; i++){
80                  for(int j = 0; j < NCARDS; j++){
81                          printf("%lf_",avgclus[i].cards[j]);
82                  }
83                  printf("\n");
84          }
85          printf("\nAvg_Classes:\n");
86          for(int i = 0; i < NCLASSES; i++){
87                  for(int j = 0; j < NCARDS; j++){
88                          printf("%lf_",avgclas[i].cards[j]);
89                  }
90                  printf("\n");
```

```
 91              }
 92              double  acc ;
 93              acc = accuracy (NLINES,NCLUSTERS,NCLASSES, clus , clas , avgclus , avgclas );
 94              printf (" \ nOverall ␣ Accuracy ␣ is ␣%lf \ n \ n" , acc );
 95
 96              //−−−−−−−−−Cleanup−−−−−−−−−−
 97              free ( data );
 98              free (mu);
 99              free ( clas );
100              free ( clus );
101              fclose ( in );
102      }
```

kohonen.c

```c
1  /*
2   * kohonen.c
3   * Sam Neyhart
4   * This file contains the kohonen program used in the final project of
5   * ECE 471.
6   * USAGE: kohonen filename.data nclusters
7   */
8
9  #include<string.h>
10 #include<time.h>
11 #include<stdio.h>
12 #include<math.h>
13 #include<stdlib.h>
14 #include"hand.h"
15
16
17 static const int ITERATIONS = 6000000; //How many times to go through data
18 static const int NCLASSES = 10;
19 static const int NCARDS = 10;
20 static int NCLUSTERS;
21 static int NLINES = 0;
22
23 int main(int argc, char ** argv)
24 {
25         //—————————File IO—————————
26         FILE * in;
27         in = fopen(argv[1],"r");
28         if (argc != 3) return 0;
29         NCLUSTERS = atoi(argv[2]);
30         //Get numLines
31         char buf[255];
32         while(fgets(buf,255,in) != NULL){
33                 NLINES++;
34         }
35         rewind(in);
36         //create arrays
37         hand * data = malloc(NLINES*sizeof(hand));
38         hand * mu = malloc(NCLUSTERS*sizeof(hand));
39         int * clas = malloc(NLINES*sizeof(int));
40         //fill data
41         for(int i = 0; i < NLINES; i++){
42                 char * tmp;
43                 fgets(buf,255,in);
44                 tmp = strtok(buf,",");
```

```
45              for(int j = 0; j < NCARDS; j++){
46                      data[i].cards[j] = atof(tmp);
47                      tmp = strtok(NULL,",");
48              }
49              clas[i]=atoi(tmp);
50      }
51      //randomly fill mu
52      time_t t;
53      srand((unsigned)time(&t));
54      for(int i = 0; i < NCLUSTERS; i++){
55              int tmp = rand() % NLINES;
56              for(int j = 0; j < NCARDS; j++){
57                      mu[i].cards[j] = data[tmp].cards[j];
58              }
59      }
60
61      //——————Perform Kohonen——————
62      for(int i = 0; i < ITERATIONS; i++){
63              kohonen_mu(&data[i%NLINES],mu,NCLUSTERS,i,ITERATIONS);
64      }
65
66      //—————Deterimine Accuracy———
67      int *clus = malloc(NLINES*sizeof(int));
68      cluster(data,mu,NLINES,NCLUSTERS,clus);
69      hand *avgclas = calloc(NCLASSES,sizeof(hand));
70      hand *avgclus = calloc(NCLUSTERS,sizeof(hand));
71      avg(avgclas, data, clas, NLINES, NCLASSES);
72      avg(avgclus, data, clus, NLINES, NCLUSTERS);
73      printf(
74                      "————————————————————For_Number_of_clusters_%d"
75                      "————————————————————\n",NCLUSTERS);
76      printf("Avg_Clusters:\n");
77      for(int i = 0; i < NCLUSTERS; i++){
78              for(int j = 0; j < NCARDS; j++){
79                      printf("%lf_",avgclus[i].cards[j]);
80              }
81              printf("\n");
82      }
83      printf("\nAvg_Classes:\n");
84      for(int i = 0; i < NCLASSES; i++){
85              for(int j = 0; j < NCARDS; j++){
86                      printf("%lf_",avgclas[i].cards[j]);
87              }
88              printf("\n");
89      }
90      double acc;
```

```
 91            acc = accuracy(NLINES,NCLUSTERS,NCLASSES, clus , clas , avgclus , avgclas );
 92            printf(")\nOverall_Accuracy_is_%lf\n\n",acc );
 93
 94            //————————Cleanup—————————
 95            free ( data );
 96            free (mu);
 97            free ( clas );
 98            free ( clus );
 99            fclose ( in );
100    }
```

evaluation.py

```
1   # Normalizaiton
2   # Input/Output file names defined in main funciton
3   # Running simply normalizes data and writes to output file
4   # Handy data separation techniques used
5
6   import math
7   import random
8   import numpy as np
9   from numpy import linalg as LA
10  from subprocess import call
11
12  DEBUG = 0
13  CLASSINDEX = 10                        # zero-indexed
14
15  def getInputSeparated(fname):
16          #fname = 'poker-hand-training-true.data'
17          file = open(fname, 'r')                        # read file
18          #with open(fname) as f:
19                  #lines = f.read().splitlines
20          lines = file.readlines()
21          file.close()
22          classData = []
23          class0Data = []                                    # can probably d
24          class1Data = []
25          class2Data = []
26          class3Data = []
27          class4Data = []
28          class5Data = []
29          class6Data = []
30          class7Data = []
31          class8Data = []
32          class9Data = []
33          for line in lines:                                # read elements
34                  line = line.strip()
35                  splitLine = line.split(',')
36                  classIndex = len(splitLine)-1   # max index of splitLine
37                  #print splitLine
38                  #print 'Class Index: {}'.format(classIndex)
39                  #print splitLine[classIndex]
40                  sampleData = []
41                  #sampleData.append(int(splitLine[:lenIndexSplitLine]))
42                  for j in range(classIndex):
43                          #print j
44                          sampleData.append(float(splitLine[j]))
```

37

```python
45                        # Separate by classes
46                        if (splitLine[classIndex] == '0'):
47                                #print 'appending 0'
48                                class0Data.append(sampleData)
49                        elif(splitLine[classIndex] == '1'):
50                                class1Data.append(sampleData)
51                        elif(splitLine[classIndex] == '2'):
52                                class2Data.append(sampleData)
53                        elif(splitLine[classIndex] == '3'):
54                                class3Data.append(sampleData)
55                        elif(splitLine[classIndex] == '4'):
56                                class4Data.append(sampleData)
57                        elif(splitLine[classIndex] == '5'):
58                                class5Data.append(sampleData)
59                        elif(splitLine[classIndex] == '6'):
60                                class6Data.append(sampleData)
61                        elif(splitLine[classIndex] == '7'):
62                                class7Data.append(sampleData)
63                        elif(splitLine[classIndex] == '8'):
64                                class8Data.append(sampleData)
65                        elif(splitLine[classIndex] == '9'):
66                                class9Data.append(sampleData)
67            classData.append(np.asarray(class0Data))
68            classData.append(np.asarray(class1Data))
69            classData.append(np.asarray(class2Data))
70            classData.append(np.asarray(class3Data))
71            classData.append(np.asarray(class4Data))
72            classData.append(np.asarray(class5Data))
73            classData.append(np.asarray(class6Data))
74            classData.append(np.asarray(class7Data))
75            classData.append(np.asarray(class8Data))
76            classData.append(np.asarray(class9Data))
77            return np.asarray(classData)
78
79  def test_getInputSeparated():
80            print 'Entered test_getInputSeparated()'
81            classData = getInputSeparated('poker-hand-training-true.txt')
82            nTotal = 0
83            i = -1
84            for classSamples in classData:
85                    i += 1
86                    print 'Class_{}_data:'.format(i)
87                    for classSample in classSamples:
88                            print classSample
89
90  def test_getInputSeparated2():
```

```python
91                    print 'Entered_test_getInputSeparated2()'
92                    classData = getInputSeparated('poker-hand-training-true.txt')
93                    print classData
94
95      # aggregates a 2-D array into a single numpy array
96      def weave(classData, classification=0):
97                    weaved = []
98                    largestClassSampCount = 0
99                    # find the length of the longest list (the number of samples in the larg
100                   for classSamples in classData:
101                           numClassSamples = len(classSamples)
102                           if numClassSamples > largestClassSampCount:
103                                   largestClassSampCount = numClassSamples
104                   # weave samples across all classes
105                   # Kevin - I can explain this, just ask me
106                   c = 0
107                   samp = []
108                   for i in range(largestClassSampCount):
109                           c = -1
110                           for j in range(len(classData)):
111                                   c += 1
112                                   classSamples = classData[j]
113                                   if (i < len(classSamples)):
114                                           samp = []
115                                           for feature in classSamples[i]:
116                                                   samp.append(feature)
117                                           if (classification): samp.append(c)
118                                           weaved.append(samp)
119                   return np.asarray(weaved)
120
121     def test_weave():
122                   fName_tr = 'poker-hand-training-true.data'
123                   # Get training data
124                   X0 = getInputSeparated(fName_tr)
125                   X = weave(X0, 1)
126                   for samp in X:
127                           print samp
128
129     def getColumnMeans(X):
130                   return np.mean(X, axis=0)
131
132     def getColumnStddevs(X):
133                   return np.std(X, axis=0)
134
135     def getColMeansSeparated(_XS):
136                   colMeansSep = []
```

```
137              for classSamples in _XS:
138                      colMeansSep.append(getColumnMeans(classSamples))
139              return np.asarray(colMeansSep)
140
141   def getColumnStddevsSeparated(_XS):
142              colMeansSep = []
143              for classSamples in _XS:
144                      colMeansSep.append(getColumnStddevs(classSamples))
145              return np.asarray(colMeansSep)
146
147   # Normalize the given dataset
148   def normalize(X, colMeans, colStddevs):
149              nX = []
150              for sample in X:
151                      nRow = sample
152                      for i in range(len(nRow)):
153                              nRow[i] = (sample[i] - colMeans[i])/colStddevs[i]
154                      nX.append(nRow)
155              return np.asarray(nX)
156
157   # Normalize each separate class
158   def normalizeSep(XS, colMeans_all_tr, colStddevs_all_tr):
159              nXS = []
160              for classSamples in XS:
161                      nXClass = normalize(classSamples, colMeans_all_tr, colStddevs_al
162                      nXS.append(nXClass)
163              return np.asarray(nXS)
164
165   # Rebuild original data file with normalized samples (classes are not normalized
166   def rebuild(nXS, fname):
167              nOriginalData = open(fname, 'w')
168              i = -1
169              for nClassSamples in nXS:
170                      i += 1
171                      for nSamp in nClassSamples:
172                              nSampString = ''
173                              for feature in nSamp:
174                                      nSampString += str(feature) + ',_'
175                              nSampString += str(i) + '\n'
176                              nOriginalData.write(nSampString)
177                              #print nSampString
178              nOriginalData.close()
179
180   # Verify correct sample classificaiton
181   def verify(sample, classification):
182              if (sample[CLASSINDEX] == classification):
```

```
183                        return 1
184            else :
185                        return 0
186         #return  sample [CLASSINDEX] == classification
187
188  # Leave  specified  set  out  (stored  as  an  empty  list)
189  def combineSetsExcept(sets , exclude ):
190         combined = []
191         i = −1
192         for  st  in  sets :
193                   i += 1
194                   if  i == exclude :
195                           combined . append ([ ])
196                           continue
197                   combined . append ( st )
198         return np. asarray (combined)
199
200  def combineSetsExcept2(sets , exclude ):
201         combined = []
202         sampWithClass = []
203         i = −1
204         for  st  in  sets :
205                   i += 1
206                   if  i == exclude : continue
207                   for  sample  in  st :
208                           sampWithClass = sample
209                           sampWithClass . append ( i )
210                           combined . append (sampWithClass)
211         return np. asarray (combined)
212
213  # Get m randomly  assigned  sets
214  def getMSets(XS, m):
215         mSets = []
216         for  mth  in  range(m):                                          # init n
217                   mSets . append ([ ])
218         #print  'len (XS):\ n{ } '. format (len (XS))
219         X = weave(XS, 1)
220         n = len (X)
221
222         # Randomly  divide  dataset  into  m sets
223         randIs = random. sample (range(n), n)                           # get random ind
224         numRIs = len ( randIs )
225         mMaxSize = (n / m) + n % m                                      # max m
226         ri = −1
227         for  _ in  range(mMaxSize):                                     # iterat
228                   for  mthSet  in  range(m):                            # iterat
```

41

```
229                              if ((ri + 1) >= numRIs):
230                                      break
231                              else:
232                                      ri += 1
233                              mSets[mthSet].append(X[ri])                          # assign
234              return mSets
235
236    def test_getMSets():
237              fName_tr = 'poker-hand-training-true.data'
238              # Get training data
239              XS = getInputSeparated(fName_tr)
240              mSets = getMSets(XS, 10)
241
242              mSetSize = 0
243              mSetSizeCombined = 0
244              i = -1
245              for mthSet in mSets:
246                      i += 1
247                      mSetSize = len(mthSet)
248                      mSetSizeCombined += mSetSize
249                      print '{}th_set:\nSize:_{}'.format(i, mSetSize)
250                      for samp in mthSet:
251                              print samp
252                      print
253              print 'mSetSizeCombined:_{}'.format(mSetSizeCombined)
254
255    # Placeholder for WIP code
256    def mFoldValid_devSup():
257              print '{}th_set:'.format(mthSet)
258              for samp in trSet:
259                      print samp
260              print '\n\n'
261
262    # m = n?  See project requirements
263    def mFoldValid(XS, m):
264              print 'Entered_mFoldValid()'
265              fName_OUT = 'm-foldValid.txt'
266              mSets = getMSets(XS, 10)
267
268              # go through sets
269              for mthSet in range(len(mSets)):
270                      te = mSets[mthSet]
271                      trS = combineSetsExcept(mSets, mthSet)                    # trS is
272                      tr = weave(trS)
273                      # Normalize tr set
274                      trColMeans = getColumnMeans(tr)
```

42

```python
275                    trColStddevs = getColumnStddevs(tr)
276                    nTr = normalize(tr, trColMeans, trColStddevs)
277                    nTrS = normalizeSep(trS, trColMeans, trColStddevs)
278                    # Normalize te set
279                    teColMeans = getColumnMeans(te)
280                    teColStddevs = getColumnStddevs(te)
281                    nTe = normalize(trSet, teColMeans, teColStddevs)
282                    # CLASSIFY using tr
283                    #rebuild(trS, fName_OUT)
284                    #call(['knn.py'])
285                    # TEST using te
286                    #call(['knn.py'])
287
288  def test_mFoldValid():
289          print 'Entered_test_mFoldValid()'
290          fName_tr = 'poker-hand-training-true.data'
291          XS = getInputSeparated(fName_tr)
292          mFoldValid(XS, 10)
293
294  def main():
295          if DEBUG: print 'Entered_main()'
296          fName_IN = 'poker-hand-training-true.data'
297          #fName_IN = 'poker-hand-testing.data'
298          fName_OUT = 'poker-hand-training-true-normalized.data'
299          #fName_OUT = 'poker-hand-testing-normalized.data'
300
301          # 'X' denotes original dataset
302          # 'XS' denotes original dataset separated by class (hand type)
303          XS = getInputSeparated(fName_IN)
304          #if DEBUG: print 'XS: \n{}'.format(XS)
305          X = weave(XS)
306
307          if DEBUG: print 'X:\n{}\n\n\n'.format(X)
308          if DEBUG: print 'XS:\n{}\n\n\n'.format(XS)
309
310          # Compute un-normalized training mu & sigma for normalization
311          colMeans_all_tr = getColumnMeans(X)
312          colStddevs_all_tr = getColumnStddevs(X)
313
314          # 'nX' denotes original dataset - normalized
315          # 'nXS' denotes original dataset, separated - normalized
316          nXS = normalizeSep(XS, colMeans_all_tr, colStddevs_all_tr)
317          nX = weave(nXS)                                    # nX is both classes con
318
319          if DEBUG: print 'nX:\n{}\n\n\n'.format(nX)
320          #if DEBUG:
```

```
321            print 'nXS:\n{}'.format(nXS)
322
323            # Create normalized file
324            #rebuild(nXS, fName_OUT)
325
326            # M-fold validation
327            mfoldValid(XS, 10)
328
329
330
331    main()
332    #test_weave()
333    #test_getMSets()
334    #test_mFoldValid()
```