SM in AI

Assignment 1

Shubham Rathi
201356033

Data Sets Used: IRIS, Wine, Glass and Breast Cancer

Details of Each Data Set:

1. IRIS:
   Attributes**:**

   1.sepal length in cm

   2. sepal width in cm

   3. petal length in cm

   4. petal width in cm

   5. class:
   -- Iris Setosa
   -- Iris Versicolour
   -- Iris Virginica

2. Wine:

   These data are the results of a chemical analysis of wines grown in the same region in Italy but derived from three different cultivars

   The attributes are
   1) Alcohol
   2) Malic acid
   3) Ash
   4) Alcalinity of ash
   5) Magnesium
   6) Total phenols
   7) Flavanoids
   8) Nonflavanoid phenols
   9) Proanthocyanins
   10)Color intensity
   11)Hue
   12)OD280/OD315 of diluted wines
   13)Proline

3. Breast Cancer:

   Dataset describes characteristics of the cell nuclei present in the image.

   Attributes:

   a) radius (mean of distances from center to points on the perimeter)
   b) texture (standard deviation of gray-scale values)
   c) perimeter
   d) area
   e) smoothness (local variation in radius lengths)
   f) compactness (perimeter^2 / area - 1.0)
   g) concavity (severity of concave portions of the contour)

h) concave points (number of concave portions of the contour)
i) symmetry
j) fractal dimension ("coastline approximation" – 1)

4. Glass:

Total 10 attributes

```
Attribute Information:
   1. Id number: 1 to 214
   2. RI: refractive index
   3. Na: Sodium (unit measurement: weight percent in corresponding
oxide, as
                  are attributes 4-10)
   4. Mg: Magnesium
   5. Al: Aluminum
   6. Si: Silicon
   7. K: Potassium
   8. Ca: Calcium
   9. Ba: Barium
  10. Fe: Iron
  11. Type of glass: (class attribute)
      -- 1 building_windows_float_processed
      -- 2 building_windows_non_float_processed
      -- 3 vehicle_windows_float_processed
      -- 4 vehicle_windows_non_float_processed (none in this
database)
      -- 5 containers
      -- 6 tableware
      -- 7 headlamps
```

Distance Function Used: Euclidian

$$E(x,y) = \sqrt{\sum_{i=0}^{n}(x_i - y_i)^2}$$

Observations:
Incase of Tie Handling, Lower index is preferred.
IRIS: Euclidian Distance function serves as a better distance function over corellation.
Wine: High errors with Euclidian Function. No clear classification.
Breast Cancer: Best classification with Euclidian
Glass: Optimum classification with Euclidian.

Accuracies are very efficient for higher fold datasets especially 5 fold. Accuracies are relatively low for lower fold data sets. Also, With increase in N, accuracy increases.

Values:

Format

Mean=[KNN1, KNN2, KNN3, KNN4, KNN5]
Standard Deviation=[KNN1, KNN2, KNN3, KNN4, KNN5]

Glass:

Fold # 2

Mean:
[98.13084112 97.19626168 96.26168224 93.92523364 94.85981308]
Standard Deviations:
[0.93457943925233877, 0.93457943925233167, 1.8691588785046704, 2.3364485981308434, 2.3364485981308434]

Fold # 3

Mean:
[98.5915493 98.5915493 97.18309859 95.77464789 96.24413146]
Standard Deviations:
[0.0, 1.1499951844052478, 0.0, 1.9918500878494327, 1.756646660457253]

Fold # 4

Mean:
[98.58490566 98.58490566 97.64150943 96.69811321 96.69811321]
Standard Deviations:
[0.81700509790984954, 1.5644456558280166, 1.5644456558280158, 2.7906036712734039, 2.0560844073305051]

Fold # 5

Mean:
[99.04761905 98.57142857 98.57142857 97.61904762 96.66666667]
Standard Deviations:
[1.1664236870396081, 1.9047619047619089, 1.9047619047619089, 3.0116930096841714, 2.8571428571428585]

Breast Cancer:

Fold # 2

Mean:
[77.28873239 73.94366197 73.76760563 70.77464789 69.19014085]
Standard Deviations:
[1.5845070422535201, 1.408450704225352, 0.52816901408450434, 3.1690140845070474, 5.4577464788732399]

Fold # 3

Mean:
[81.48148148 78.13051146 78.30687831 75.13227513 74.07407407]
Standard Deviations:
[1.1429877774969768, 1.5171649500957034, 0.86401754595527025, 1.4965222882254987, 1.9797129030549998]

Fold # 4

Mean:
[82.21830986 77.81690141 79.92957746 76.05633803 75.88028169]
Standard Deviations:
[2.4584929654522774, 3.7759877798463428, 3.0493852245930935, 2.4898126098117852, 2.4075342132354449]

Fold # 5

Mean:
[82.30088496 78.76106195 80. 76.28318584 77.34513274]

Standard Deviations:
[3.6701666112084466, 4.1508104069233891, 3.09608065213025, 4.8600106965472527, 4.6693472418665225]

WINE:

Fold # 2

Mean:
[75.28089888 74.15730337 69.1011236 69.1011236 71.91011236]
Standard Deviations:
[5.6179775280898809, 4.4943820224719104, 5.0561797752809028, 3.9325842696629181, 1.1235955056179776]

Fold # 3

Mean:
[73.44632768 72.31638418 70.62146893 74.01129944 72.88135593]
Standard Deviations:
[3.4827197756886852, 2.113930726990926, 3.1959628528205517, 4.8600707723404701, 4.7939442792308276]

Fold # 4

Mean:
[77.27272727 73.29545455 74.43181818 74.43181818 72.72727273]
Standard Deviations:
[7.0050159124647466, 4.3642873567435307, 9.8247820828355579, 7.6017546365111652, 7.3644780663725706]

Fold # 5

Mean:
[77.71428571 72.57142857 73.14285714 74.28571429 74.28571429]
Standard Deviations:
[4.9156144383100688, 5.29921056885469, 5.8832172234211466, 5.1110125199995222, 10.690449676496975]

IRIS:

Fold # 2

Mean:
[94. 94. 94. 96. 94.66666667]
Standard Deviations:
[2.0, 2.0, 2.0, 0.0, 1.3333333333333357]

Fold # 3

Mean:
[95.33333333 94. 93.33333333 95.33333333 94.]
Standard Deviations:
[2.4944382578492941, 1.6329931618554521, 4.1096093353126504, 0.94280904158206336, 2.8284271247461903]

Fold # 4

Mean:
[95.94594595 95.27027027 95.27027027 96.62162162 96.62162162]
Standard Deviations:
[3.0217134831078214, 2.9452019888788303, 2.9452019888788303, 2.2409626961860809, 2.2409626961860809]

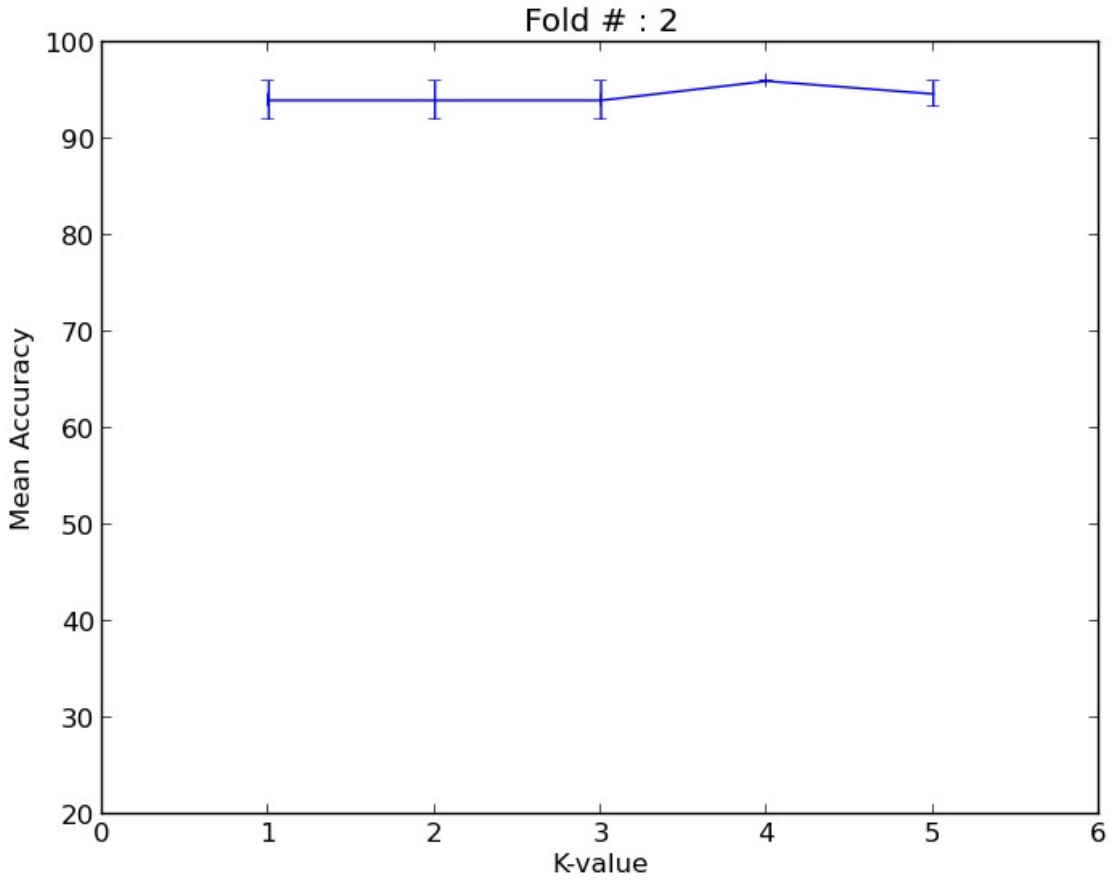
Fold # 5

Mean:
[96. 96. 96. 97.33333333 96.66666667]

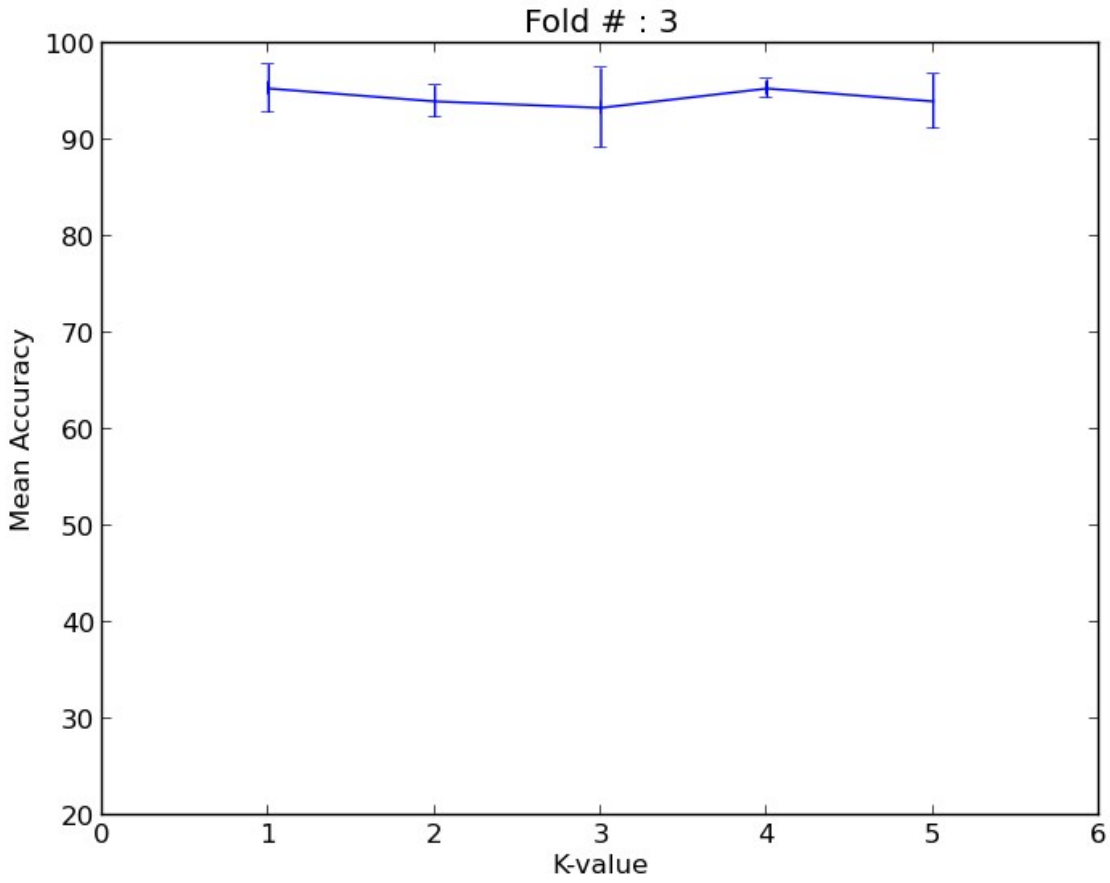
Standard Deviations:
[3.8873012632302011, 3.8873012632302011, 3.8873012632302011, 2.4944382578492954, 3.6514837167011072]

Plots:

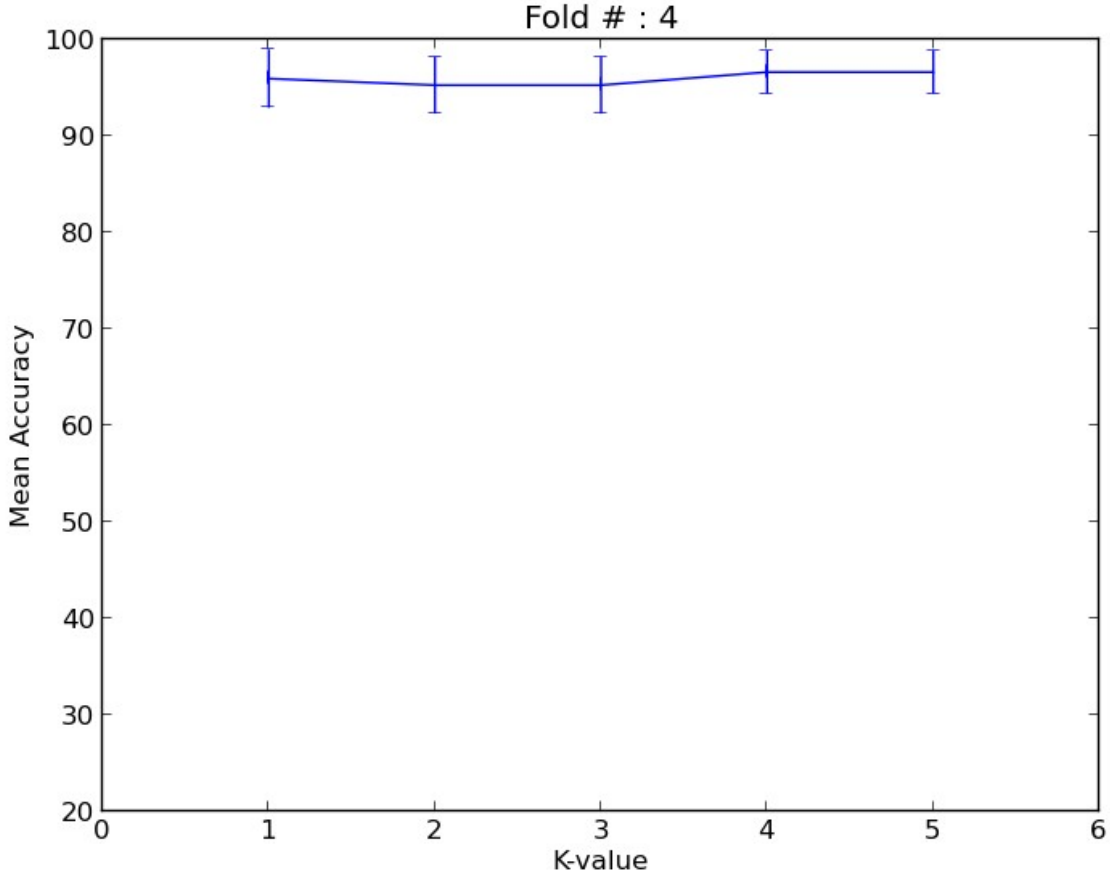
IRIS

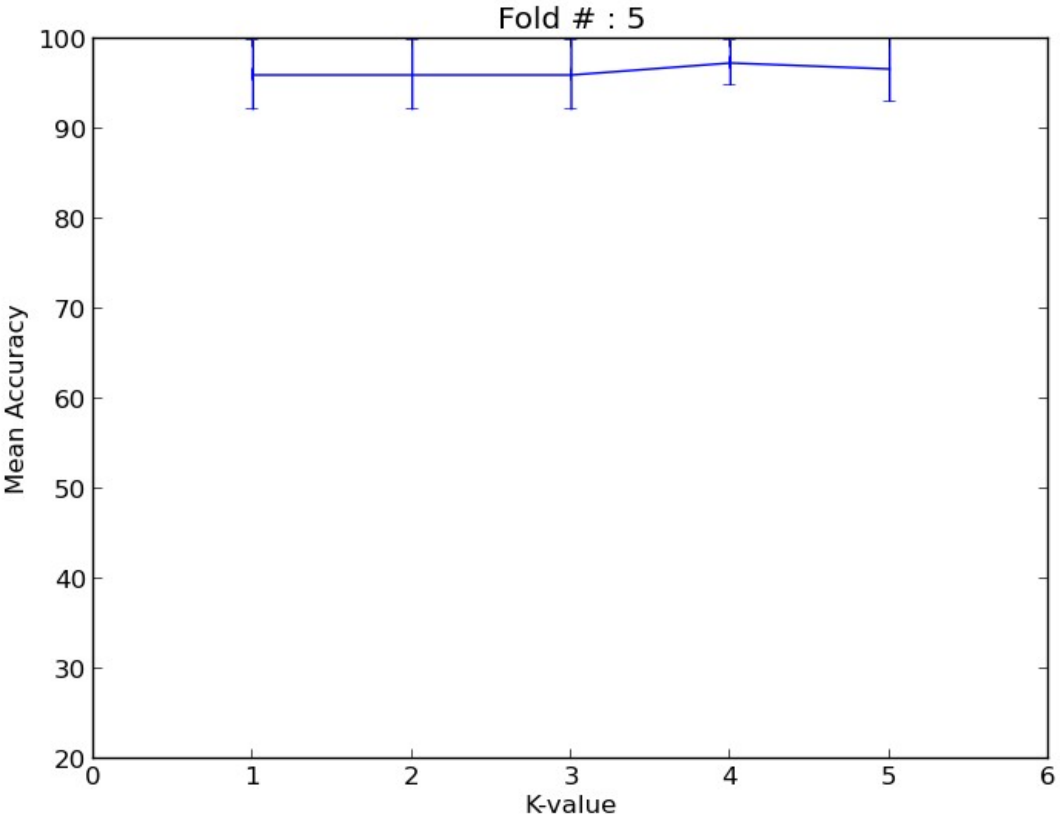


_

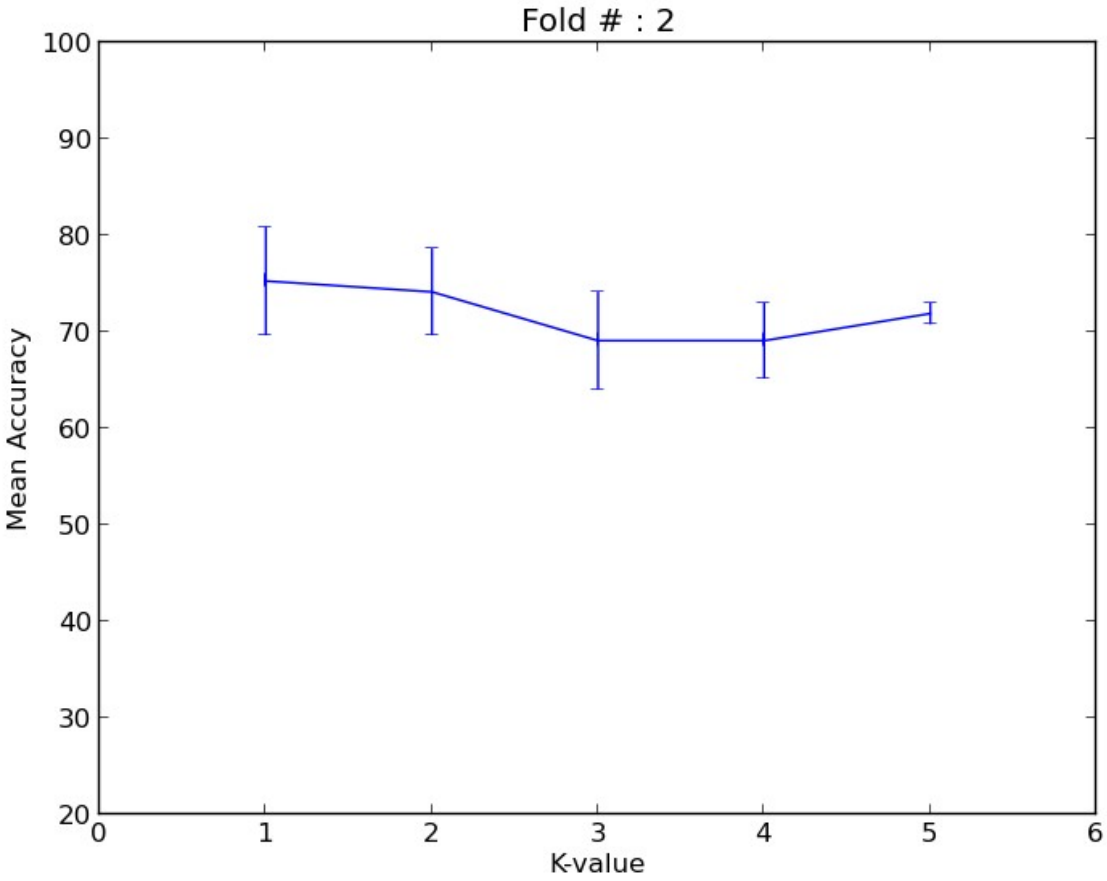

Fold # : 3
Mean Accuracy
K-value
100
90
80
70
60
50
40
30
20
0
1
2
3
4
5
6

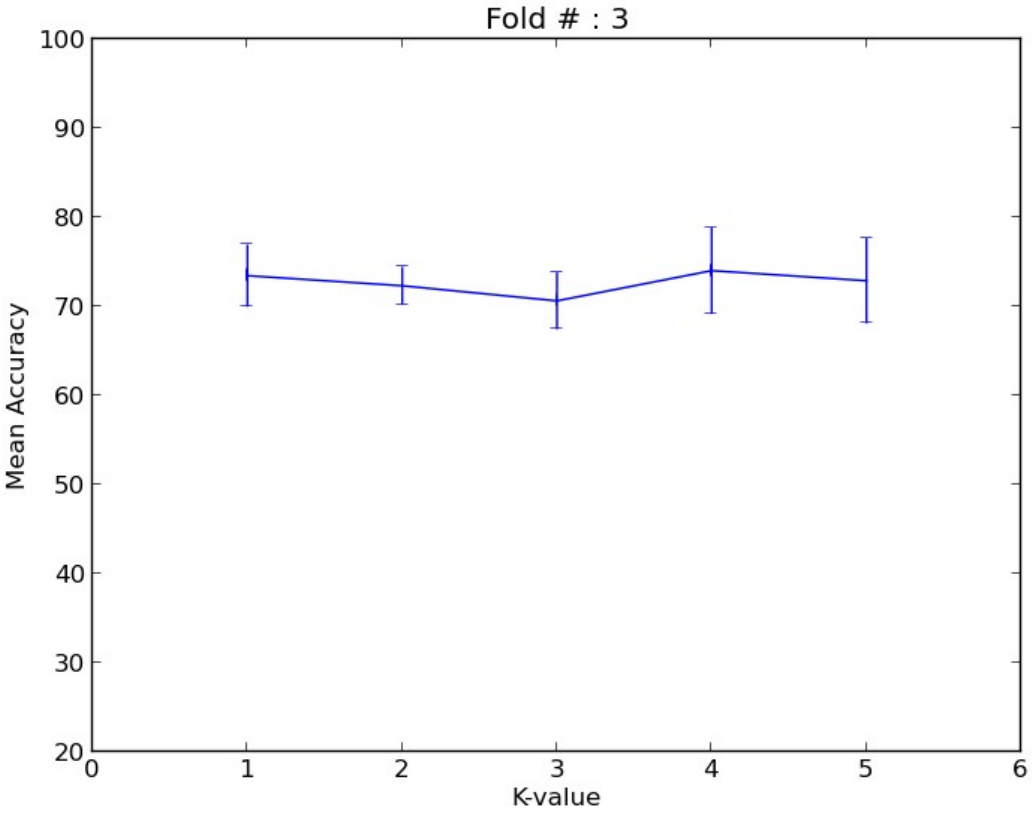
–

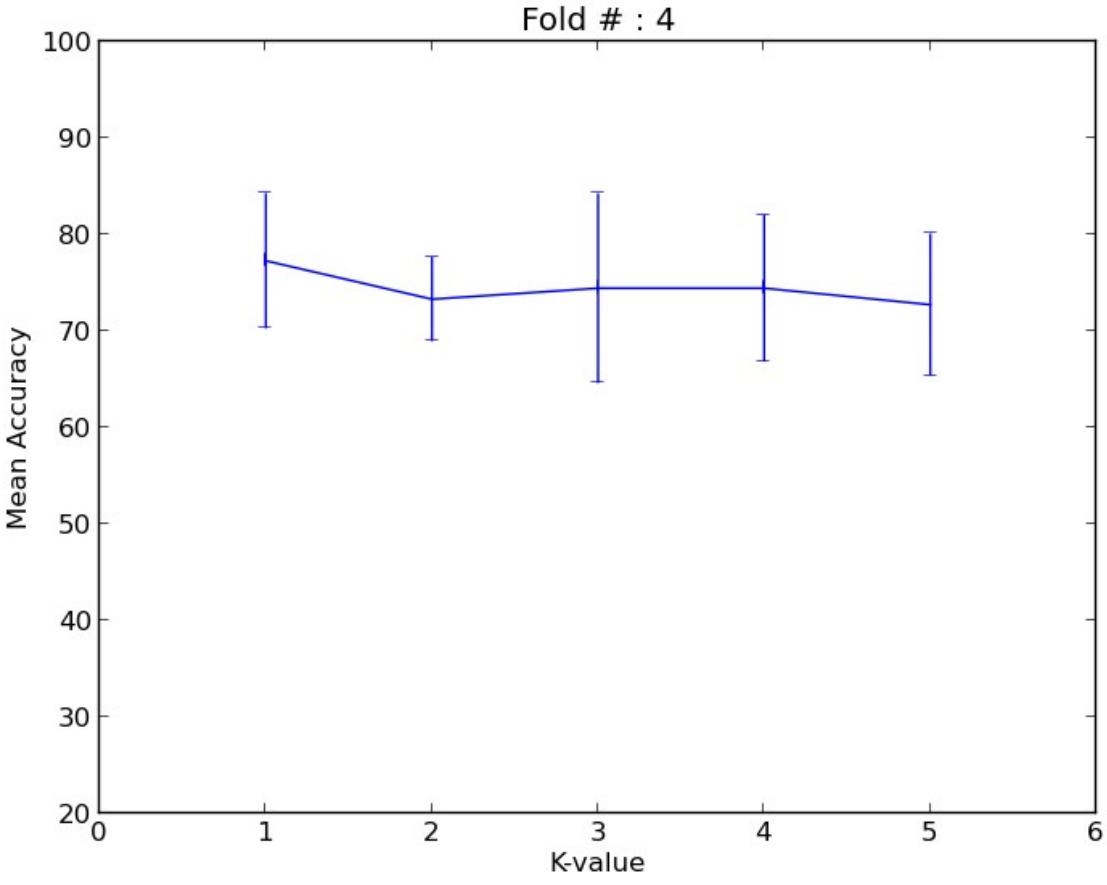

Fold # : 4
Mean Accuracy
K-value
100
90
80
70
60
50
40
30
20
0
1
2
3
4
5
6

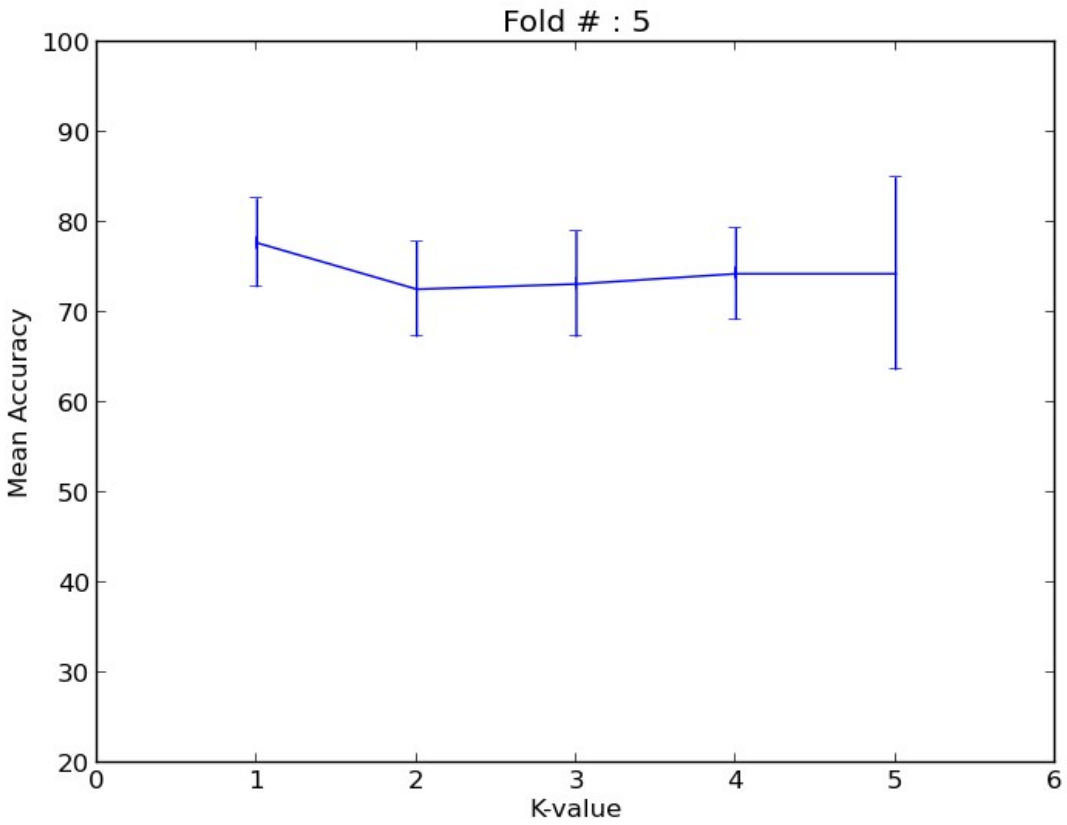

Fold # : 5
Mean Accuracy
K-value
100
90
80
70
60
50
40
30
20
0
1
2
3
4
5
6

WINE:

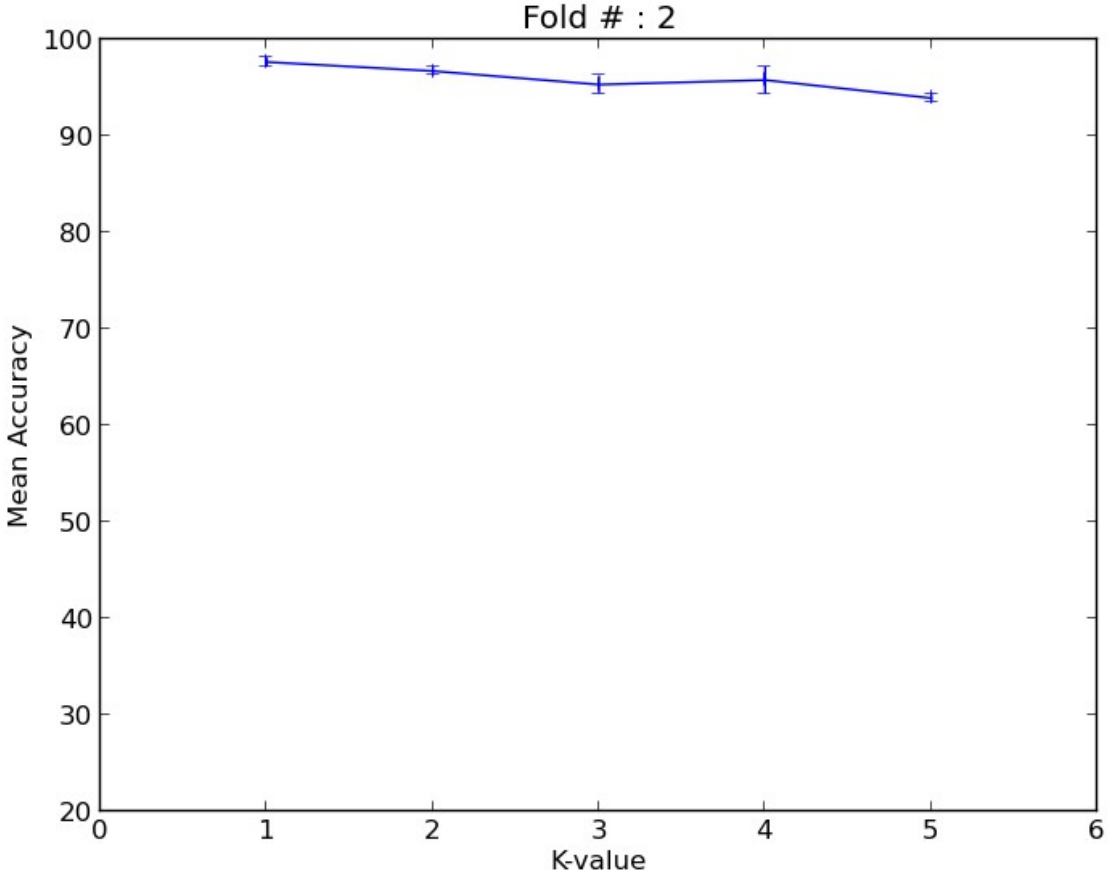

Fold # : 2
Mean Accuracy
K-value
100
90
80
70
60
50
40
30
20
0
1
2
3
4
5
6

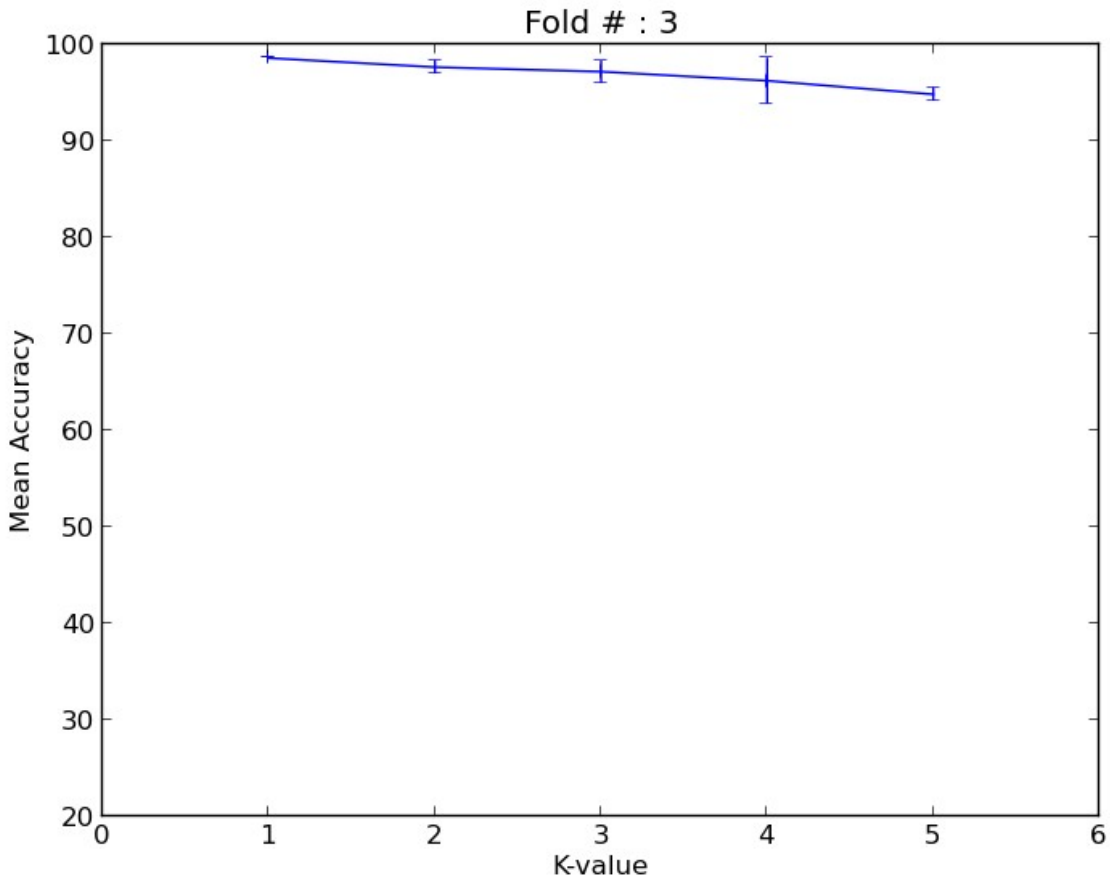

Fold # : 3
Mean Accuracy
K-value
100
90
80
70
60
50
40
30
20
0
1
2
3
4
5
6

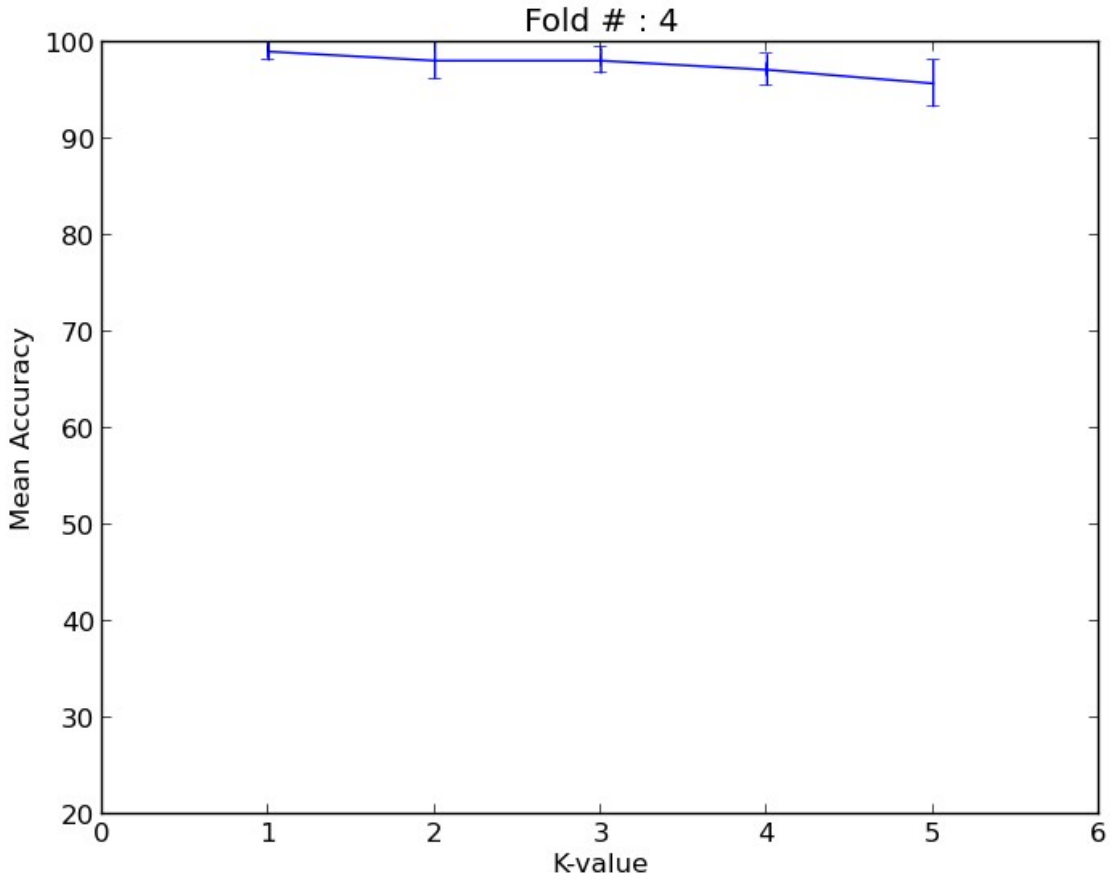

Fold # : 4
Mean Accuracy
K-value
100
90
80
70
60
50
40
30
20
0
1
2
3
4
5
6

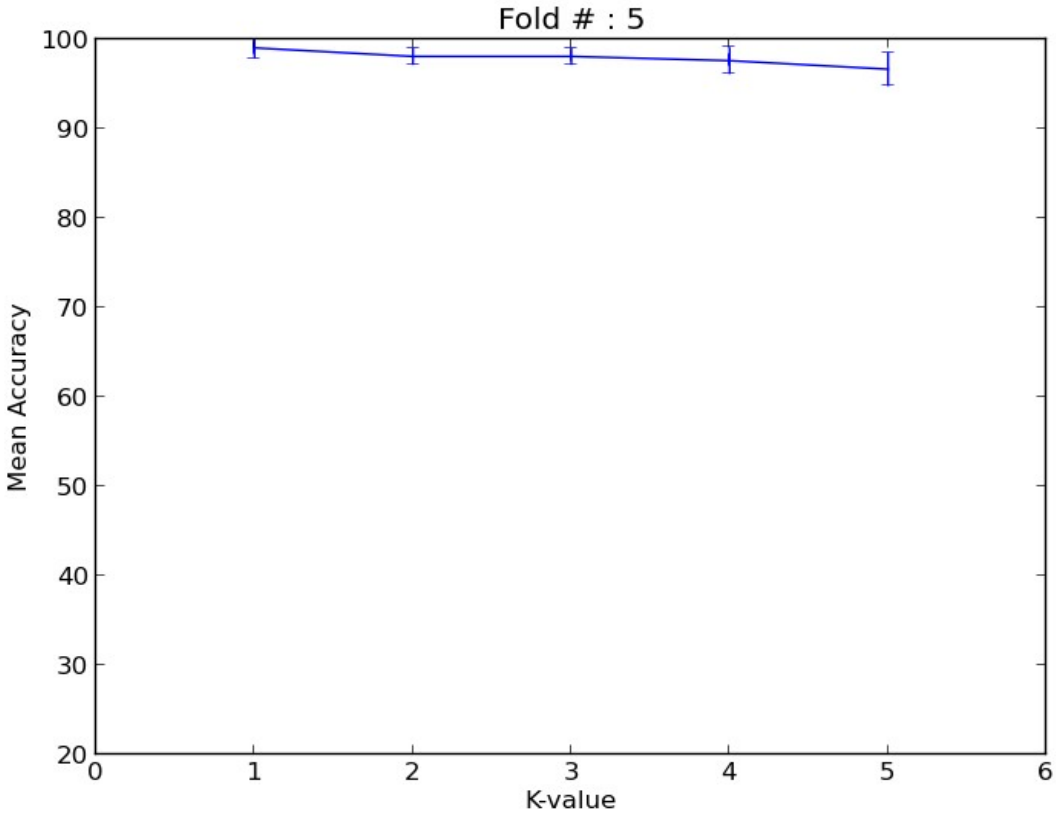

Fold # : 5
Mean Accuracy
K-value
100
90
80
70
60
50
40
30
20
0
1
2
3
4
5
6

GLASS

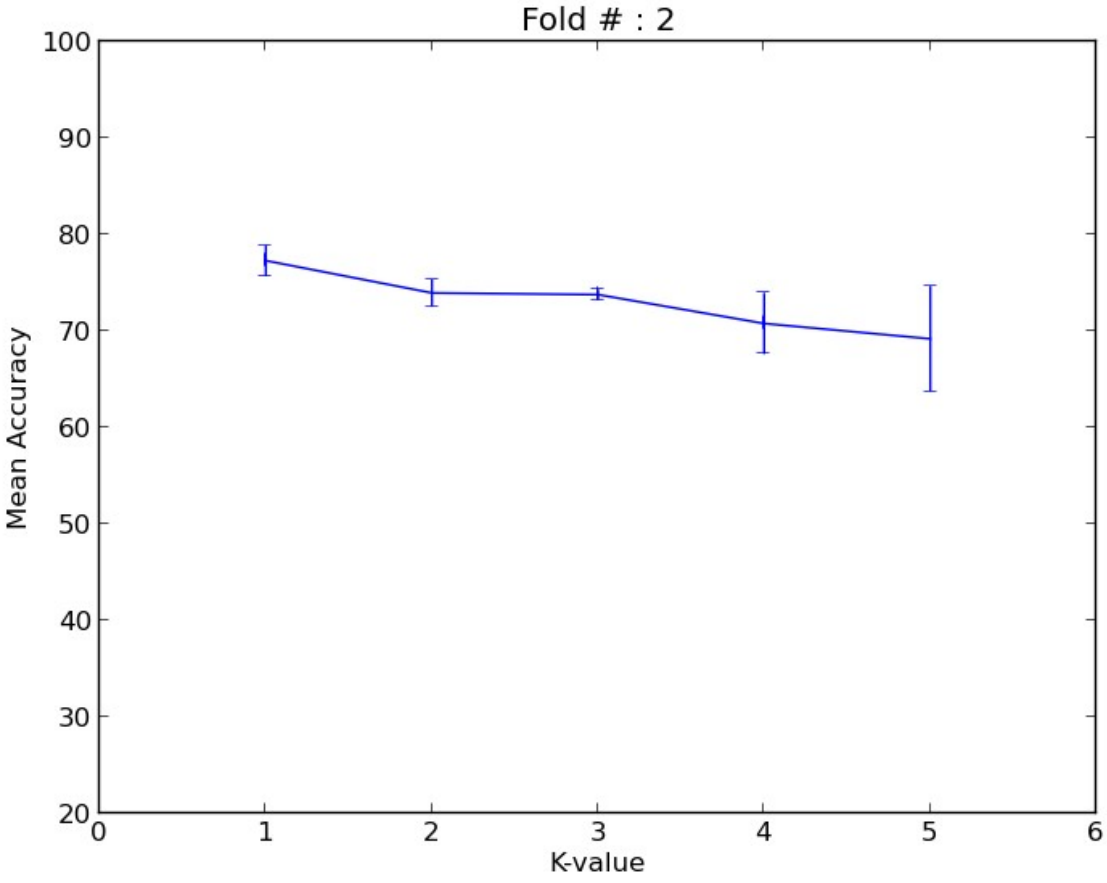

Fold # : 2
Mean Accuracy
K-value
100
90
80
70
60
50
40
30
20
0
1
2
3
4
5
6

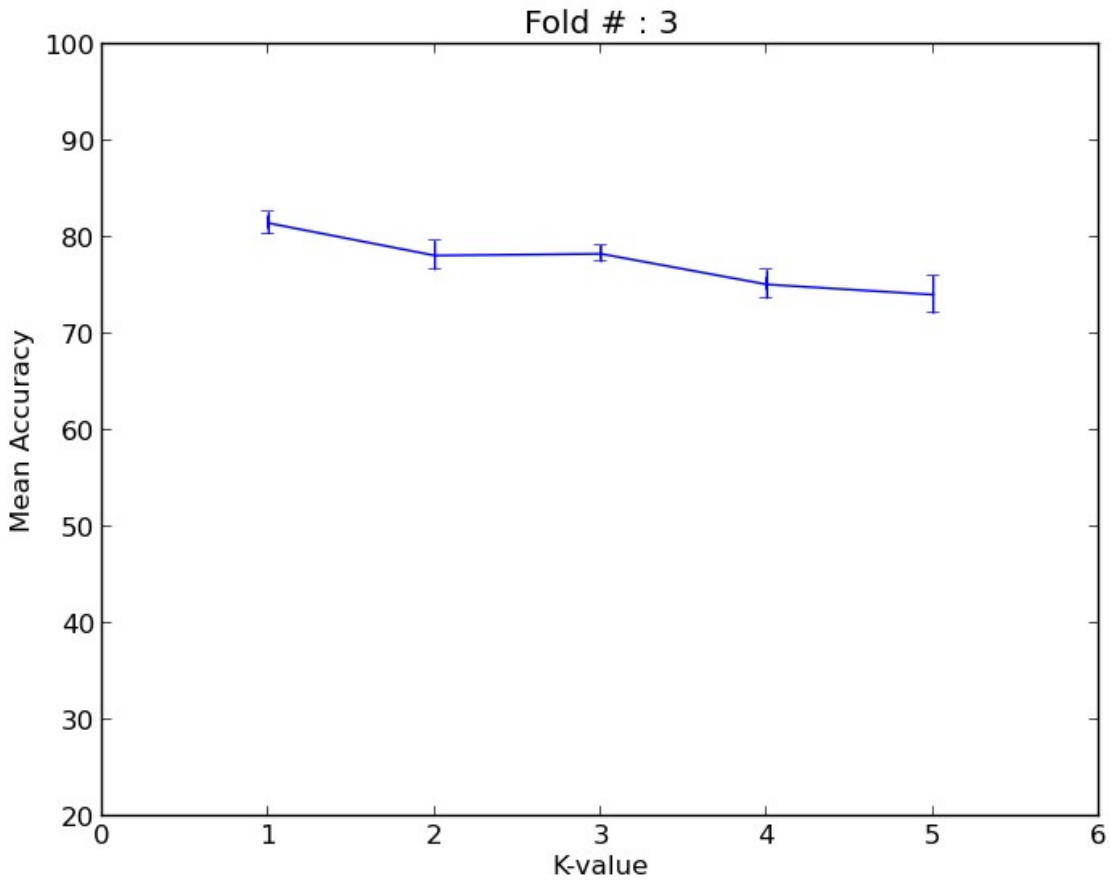

Fold # : 3
Mean Accuracy
K-value
100
90
80
70
60
50
40
30
20
0
1
2
3
4
5
6

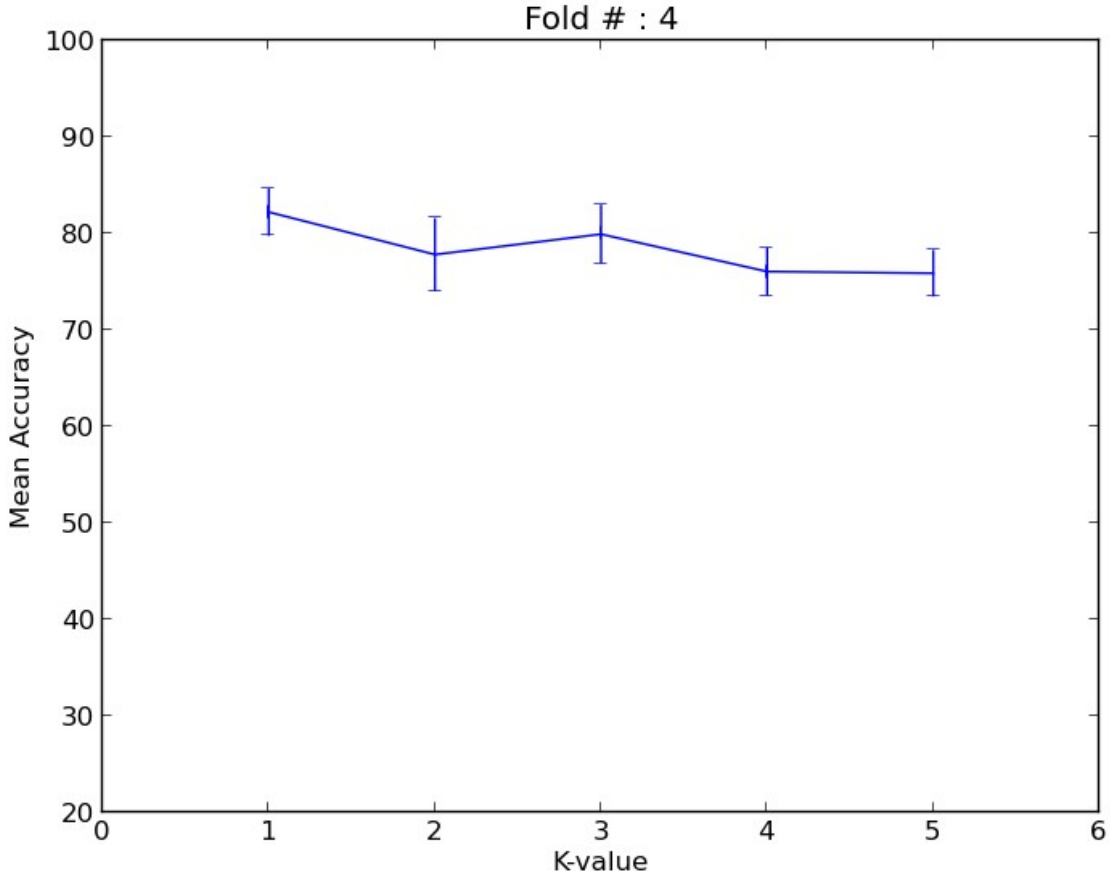

Fold # : 4
Mean Accuracy
K-value
100
90
80
70
60
50
40
30
20
0
1
2
3
4
5
6

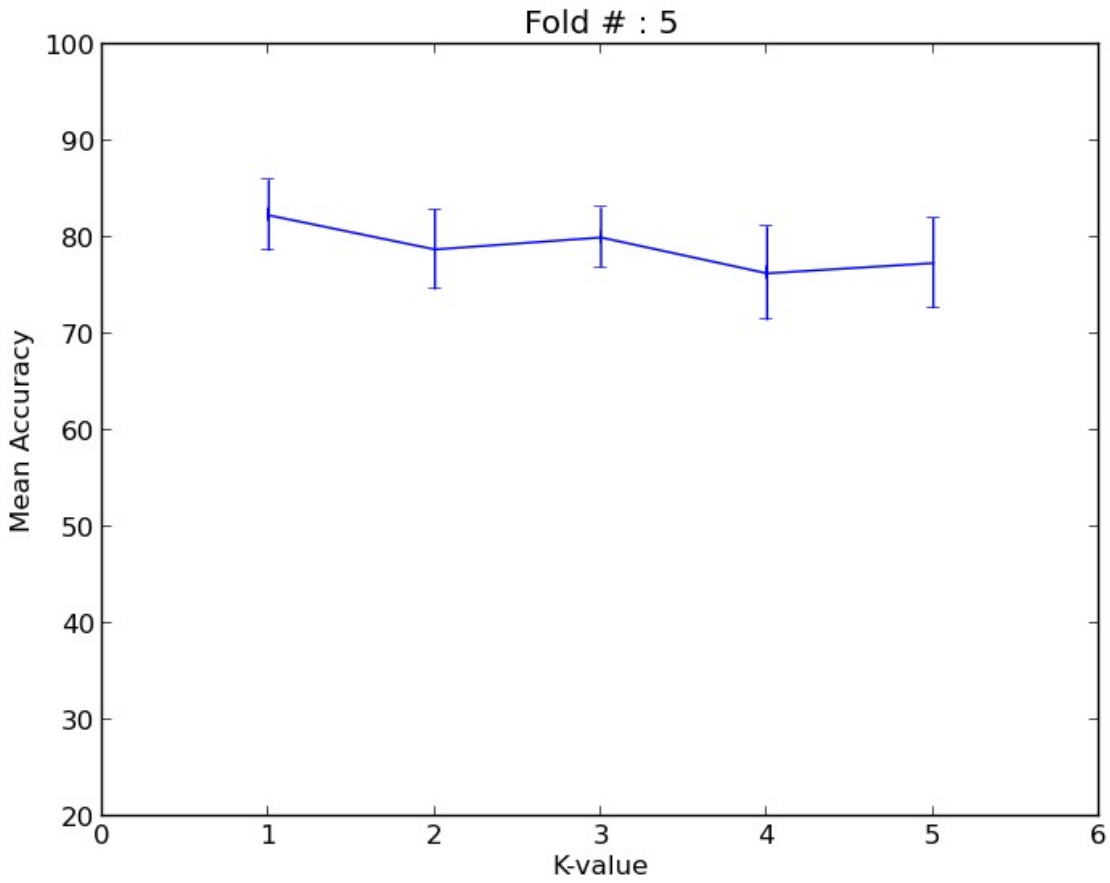


BREAST CANCER:




Fold # : 3
Mean Accuracy
K-value
100
90
80
70
60
50
40
30
20
0
1
2
3
4
5
6


Fold # : 4
Mean Accuracy
K-value
100
90
80
70
60
50
40
30
20
0
1
2
3
4
5
6



CODE:

```
import csv
import random
import math
import operator
import sys
import numpy
import matplotlib.pyplot as plot

def loadFold(fold, classifier, dataset=[], trainingSet=[] , testSet=[]):
        arr=[0]*10
        sda=[[],[],[],[],[]]
        divider=len(dataset)/fold
        for i in range(0,fold):
                testSet=dataset[i*divider:(i+1)*divider]
                trainingSet=dataset[0:i*divider]+dataset[(i+1)*divider:len(dataset) - 1]
                for k in range(1,6):
                        predictions=[]
                        for x in range(0,len(testSet)):
                                # print testSet[x]
                                neighbors = getNeighbors(trainingSet, testSet[x], k,classifier)
                                result = getResponse(neighbors,classifier)
                                predictions.append(result)
                        accuracy = getAccuracy(testSet, predictions,classifier)
                        arr[k]+=accuracy
```

```
                    sda[k-1].append(accuracy)
                    #print('Accuracy: ' + repr(accuracy) + '%'+' for KNN: '+str(k))
        mean=numpy.array(arr)
        mean=mean/fold
        print "Mean:\n"
        print mean[1:6]
        s1=numpy.std(numpy.array(sda[0]))
        s2=numpy.std(numpy.array(sda[1]))
        s3=numpy.std(numpy.array(sda[2]))
        s4=numpy.std(numpy.array(sda[3]))
        s5=numpy.std(numpy.array(sda[4]))
        print "Standard Deviations:"
        standardD =[s1,s2,s3,s4,s5]
        print standardD
        plot.title("Fold # : %d" % fold)
        plot.errorbar(range(1,6),mean[1:6],xerr=0,yerr=standardD)
        plot.xlabel('K-value')
        plot.ylabel('Mean Accuracy')
        plot.ylim([20,100])
        plot.xlim([0,6])
        nm=str(fold)+".png"
        plot.savefig(str(nm))
        plot.close()
        #plot.show()


def euclideanDistance(instance1, instance2, length, ignore_it):
        distance = 0
        for x in range(length):
                if not x == ignore_it:
                        distance += pow(float(instance1[x]) - float(instance2[x]), 2)
        return math.sqrt(distance)

def getNeighbors(trainingSet, testInstance, k,classifier):
        distances = []
        length = len(testInstance)
        for x in range(len(trainingSet)):
                dist = euclideanDistance(testInstance, trainingSet[x], length,classifier)
                distances.append((trainingSet[x], dist))
        distances.sort(key=operator.itemgetter(1))
        neighbors = []
        for x in range(k):
                neighbors.append(distances[x][0])
        return neighbors

def getResponse(neighbors,classifier):
        classVotes = {}
        for x in range(len(neighbors)):
                response = neighbors[x][classifier]
                if response in classVotes:
                        classVotes[response] += 1
                else:
                        classVotes[response] = 1
        sortedVotes = sorted(classVotes.iteritems(), key=operator.itemgetter(1), reverse=True)
```

```
        return sortedVotes[0][0]

def getAccuracy(testSet, predictions,classifier):
        correct = 0
        for x in range(len(testSet)):
                if testSet[x][classifier] == predictions[x]:
                        correct += 1
        return (correct/float(len(testSet))) * 100.0

def main():
        trainingSet=[]
        testSet=[]
        filename=sys.argv[1]
        classifier = int(sys.argv[2])
        with open(filename, 'rb') as csvfile:
           lines = csv.reader(csvfile)
           dataset = list(lines)
           random.shuffle(dataset)
        try:
                dataset.remove([])
        except:
                pass
        #random.shuffle(dataset)
        for fold in range(2,6):
                print "Fold # ",fold
                loadFold(fold, classifier, dataset, trainingSet, testSet)

main()
```