Homework 2 (Due: 10/15/2018)

COEN 281, Fall 2018
Professor Marwah

The objective of this HW is to implement k-NN and cross-validation to find the best value of k for a binary classification task. The task to diagnose breast cancer based on 30 numeric features. However, to keep things simple, we will only use two of those features. The output is binary: 0 benign, 1 malignant. In all there are 569 examples, which we will split into training and test sets. There are no missing values.

In [21]:
```
# load the data set
import numpy as np
import sklearn.datasets
import pandas as pd
#from sklearn.datasets import load_breast_cancer
#data = load_breast_cancer()

dat = sklearn.datasets.load_breast_cancer()
```

In [105]:
```
# uncomment the following and run it to get a description of the data set
#print(dat.DESCR)
#print(dat.target_names,dat.feature_names)
```

In [23]:
```
# dat is a dictionary with the data, let's see what keys it has
dat.keys()
```

Out[23]:
```
dict_keys(['data', 'target', 'target_names', 'DESCR', 'feature_names', 'filename'])
```

In [24]:
```
# we will use two features: 'mean area' and 'mean concave points'
ix1 = np.where(dat["feature_names"] == "mean area")[0][0]
ix2 = np.where(dat["feature_names"] == "mean concave points")[0][0]

#play around
print(np.where(dat["feature_names"] == "mean area"))
print("\n",ix1," ",ix2)
X = dat["data"][:,(ix1,ix2)]
Y = dat["target"]
print(type(ix1))
print(X.ndim)
print(dat["data"])
print("\n",dat["data"][:,(2,1)])

#print("\n target \n",dat["target"])
print(X.shape, Y.shape)
print(type(X))
```

```
(array([3]),)

 3   7
<class 'numpy.int64'>
2
[[  1.79900000e+01   1.03800000e+01   1.22800000e+02 ...,   2.65400000e-01
    4.60100000e-01   1.18900000e-01]
 [  2.05700000e+01   1.77700000e+01   1.32900000e+02 ...,   1.86000000e-01
    2.75000000e-01   8.90200000e-02]
 [  1.96900000e+01   2.12500000e+01   1.30000000e+02 ...,   2.43000000e-01
    3.61300000e-01   8.75800000e-02]
 ...,
 [  1.66000000e+01   2.80800000e+01   1.08300000e+02 ...,   1.41800000e-01
    2.21800000e-01   7.82000000e-02]
 [  2.06000000e+01   2.93300000e+01   1.40100000e+02 ...,   2.65000000e-01
    4.08700000e-01   1.24000000e-01]
 [  7.76000000e+00   2.45400000e+01   4.79200000e+01 ...,   0.00000000e+00
    2.87100000e-01   7.03900000e-02]]

 [[ 122.8    10.38]
 [ 132.9    17.77]
 [ 130.     21.25]
 ...,
 [ 108.3    28.08]
 [ 140.1    29.33]
 [  47.92   24.54]]
(569, 2) (569,)
<class 'numpy.ndarray'>
```

In [89]:
```
X = dat["data"][:,(ix1,ix2)]
Y = dat["target"]
# verify shape of X and Y
print(X.shape, Y.shape)

# stats of the two features
from scipy import stats
st = stats.describe(X)
print("Number of points: %i" % st.nobs)
print("range (min, max), X1: (%.2f, %.2f), X2: (%.2f, %.2f)" % (st.minmax[0][0], st.minmax[1][0], st.minm
ax[0][1], st.minmax[1][1]))
print("mean: %.2f, %.2f" % (st.mean[0], st.mean[1]))
print("variance: %.2f, %f" % (st.variance[0], st.variance[1]))

# Given the stats, is it a good idea to normalize the features?

# Yes, as the stats show that the variance is high
# add code to normalize features
# CODE STARTS HERE
for i in range(st.nobs):
    X[i][0]=(X[i][0]-st.minmax[0][0])/(st.minmax[1][0]-st.minmax[0][0])
for i in range(st.nobs):
    X[i][1]=(X[i][1]-st.minmax[0][1])/(st.minmax[1][1]-st.minmax[0][1])

print("\n Stats after normalizing :\n")
# stats of the two features
from scipy import stats
st = stats.describe(X)
print("Number of points: %i" % st.nobs)
print("range (min, max), X1: (%.2f, %.2f), X2: (%.2f, %.2f)" % (st.minmax[0][0], st.minmax[1][0], st.minm
ax[0][1], st.minmax[1][1]))
print("mean: %.2f, %.2f" % (st.mean[0], st.mean[1]))
print("variance: %.2f, %f" % (st.variance[0], st.variance[1]))
```

```
(569, 2) (569,)
Number of points: 569
range (min, max), X1: (143.50, 2501.00), X2: (0.00, 0.20)
mean: 654.89, 0.05
variance: 123843.55, 0.001506

 Stats after normalizing :

Number of points: 569
range (min, max), X1: (0.00, 1.00), X2: (0.00, 1.00)
mean: 0.22, 0.24
variance: 0.02, 0.037194
```

In [90]:
```
# split into training set / test set
#
# usually you would do the split randomly; here for deterministic results, we assume the data
# points are already shuffled and take the first 70% as training and the rest as test
#
nTot = X.shape[0]
nTr = int(nTot*0.7)
nTs = nTot - nTr

Xtr = X[0:nTr,]
Ytr = Y[0:nTr]

Xts = X[nTr:nTot,]
Yts = Y[nTr:nTot,]

# verify shapes
print(Xtr.shape, Ytr.shape, Xts.shape, Yts.shape)
```

```
(398, 2) (398,) (171, 2) (171,)
```

k-NN Implementation

In [92]:
```
# Implement the following functions for k-NN
#


#
# knn_predict(Xtr, Ytr, Xts, k)
#
# input: Xtr - training examples input features, size nXd
#        Ytr - label (can assume to be binary 0/1), size nX1
#        Xts - test examples input features, for which labels (Yts)
#              need to be predicted, size mXd
#        k   - k for the k-NN algo
#
# output: Yts - 0/1 labes for Xts, size mX1
#
#  This function predicts the binary labels for Xts, given the training
#  data Xtr, Ytr and k, using the k-NN algorithm
#
def knn_predict(Xtr, Ytr, Xts, k):
    # compute in two steps

    # step 1: compute dist matrix between Xts and Xtr
    #
    distance = compute_dist_mat(Xtr, Xts)
    distance = np.array(distance)

    # step 2: use the dist matrix and use k-nn to find labels for Xts
    # hint: function numpy.argsort may be useful
    # in case of a tie, pick a class randomly

    sort_distance = []
    for i in range(Xts.shape[0]):
        sort_distance.append(distance[i].argsort()[:k])
    cls = [0 for i in range(Xts.shape[0])]
    for i in range(Xts.shape[0]):
        cnt = 0.0
        for j in range(k):
            if Ytr[sort_distance[i][j]] == 0:
                cnt = cnt+ 1
        if  cnt > (float(k)/2):
               cls[i] =  0
        elif cnt < (float(k)/2):
               cls[i] = 1
        else:
               cls[i] = np.random.randint(2, size=1)[0]
    return cls

#  compute_dist_mat(Xts, Xtr)
#
#  input: Xts - test examples, size mXd
#         Xtr - training examples, size nXd
#  output: L2 distance matrix mXn
#
#   if Xts is mXd, and Xtr is nXd, this function returns a matrix of size mXn with the L2 distances; the
 (i,j)
#     entry of the matrix is the L2 distance between ith test and jth training example
#

def compute_dist_mat(Xtr, Xts):
    # use two for loops to compute the matrix
    # YOUR CODE HERE
    dist = []
    for i in range(Xts.shape[0]):
        var = []
        for j in range(Xtr.shape[0]):
            var.append((((Xts[i][0]-Xtr[j][0])**2) + ((Xts[i][1]-Xtr[j][1])**2))**0.5)
        dist.append(var)
    return dist
```

Problem 1 (30 points): Fill-in code for normalizing features, and the above two functions to implement k-NN.

Problem 2 (20 points): Run your k-NN implementation on the test data set. Use k=5. Compute accuracy, recall and precision of the test data set (do not use python library functions to compute these).

In [94]:
```
# problem 2 solution

knn_test=knn_predict(Xtr, Ytr, Xts, 5)
#initializing variables for true-positives, true-negatives, false-nagatives and false-positives
true_pos=0
true_neg=0
false_pos=0
false_neg=0

for i in range(len(Zts)):
    if knn_test[i]==1 and Yts[i]==1:
        true_pos = true_pos + 1
    elif knn_test[i]==0 and Yts[i]==0:
        true_neg = true_neg + 1
    elif knn_test[i]==0 and Yts[i]==1:
        false_neg = false_neg + 1
    else:
        false_pos = false_pos + 1

print('Accuracy = ',(true_pos + true_neg)/(true_pos + true_neg + false_neg + false_pos)*100,"%")
print('Recall = ',(true_pos)/(true_pos + false_neg)*100, "%")
print('Precision = ',(true_pos)/(true_pos + false_pos)*100, "%")
```

```
Accuracy =  91.17647058823529 %
Recall =  89.31297709923665 %
Precision =  99.15254237288136 %
```

In [95]:
```
def accuarcyOfKNN(Xtr, Ytr, Xts, Yts, k):
# problem 2 solution
    knn_test=knn_predict(Xtr, Ytr, Xts, 5)
    #initializing variables for true-positives, true-negatives, false-nagatives and false-positives
    true_pos=0
    true_neg=0
    false_pos=0
    false_neg=0
    for i in range(len(knn_test)):
        if knn_test[i]==1 and Yts[i]==1:
            true_pos=true_pos+1
        elif knn_test[i]==0 and Yts[i]==0:
            true_neg=true_neg+1
        elif knn_test[i]==0 and Yts[i]==1:
            false_neg=false_neg+1
        else:
            false_pos=false_pos+1

    return (true_pos+true_neg)/(true_pos+true_neg+false_neg+false_pos)
```

Cross-Validation

Priblem 3 (30 points): Now we will implement 5-fold cross-validation to find the best value of k. And then using that value of k, re-run k-NN on the test data set. (This is adapted from a past Stanford cs231n assignment)

In [96]:
```
num_folds = 5
k_choices = [1, 3, 5, 7, 9, 11, 13, 15, 20, 30, 40, 50, 75, 100]

x_train_folds = []
y_train_folds = []
################################################################################
# TODO:                                                                        #
# Split up the training data into folds. After splitting, X_train_folds and    #
# y_train_folds should each be lists of length num_folds, where                #
# y_train_folds[i] is the label vector for the points in X_train_folds[i].     #
# Hint: Look up the numpy array_split function.                                #
################################################################################
pass
x_train_folds = np.array_split(Xtr,5)
y_train_folds = np.array_split(Ytr,5)
################################################################################
#                                END OF YOUR CODE                              #
################################################################################

# A dictionary holding the accuracies for different values of k that we find
# when running cross-validation. After running cross-validation,
# k_to_accuracies[k] should be a list of length num_folds giving the different
# accuracy values that we found when using that value of k.
k_to_accuracies = {}

################################################################################
# TODO:                                                                        #
# Perform k-fold cross validation to find the best value of k. For each        #
# possible value of k, run the k-nearest-neighbor algorithm num_folds times,   #
# where in each case you use all but one of the folds as training data and the #
# last fold as a validation set. Store the accuracies for all fold and all     #
# values of k in the k_to_accuracies dictionary.                               #
################################################################################

################################################################################
#                                END OF YOUR CODE                              #
################################################################################

# Print out the computed accuracies
for k in sorted(k_to_accuracies):
    for accuracy in k_to_accuracies[k]:
        print('k = %d, accuracy = %f' % (k, accuracy))
```

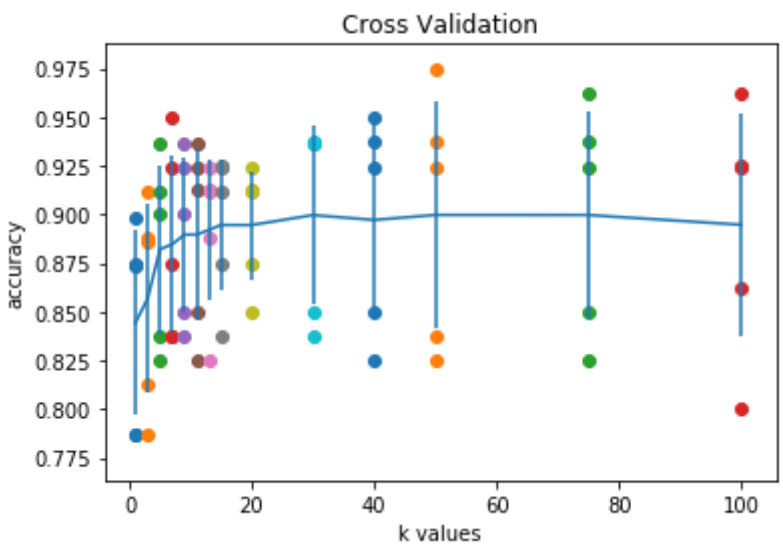
In [97]:
```
x_train_folds = np.array_split(Xtr,5)
y_train_folds = np.array_split(Ytr,5)
print("k    List Of Accuracies for every fold")
for k in k_choices:
    SumOfAccuracies = 0.0
    ListOfAccuracies = []
    for fold in range(num_folds):
        XtestingData = x_train_folds[fold]
        YtestingData = y_train_folds[fold]
        #print(testingData)
        first = True
        for f in range(num_folds):
            if f!= fold:
                if first == True:
                    XtraningData =  x_train_folds[f]
                    YtraningData =  y_train_folds[f]
                    first = False
                else:
                    XtraningData = np.concatenate((XtraningData, x_train_folds[f]), axis=0)
                    YtraningData = np.concatenate((YtraningData, y_train_folds[f]), axis=0)
        accuracyOfFold = accuracyOfKNN(XtraningData, YtraningData, XtestingData, YtestingData, k)
        ListOfAccuracies.append(accuracyOfFold)
        SumOfAccuracies += accuracyOfFold

    k_to_accuracies[k] = ListOfAccuracies
    print(k,"  ", ListOfAccuracies)
```

```
k    List Of Accuracies for every fold
1    [0.7875, 0.7875, 0.875, 0.8734177215189873, 0.8987341772151899]
3    [0.7875, 0.8125, 0.8875, 0.9113924050632911, 0.8860759493670886]
5    [0.825, 0.8375, 0.9, 0.9367088607594937, 0.9113924050632911]
7    [0.8375, 0.8375, 0.875, 0.9493670886075949, 0.9240506329113924]
9    [0.85, 0.8375, 0.9, 0.9240506329113924, 0.9367088607594937]
11    [0.85, 0.825, 0.9125, 0.9240506329113924, 0.9367088607594937]
13    [0.8875, 0.825, 0.9125, 0.9240506329113924, 0.9113924050632911]
15    [0.875, 0.8375, 0.925, 0.9240506329113924, 0.9113924050632911]
20    [0.875, 0.85, 0.9125, 0.9113924050632911, 0.9240506329113924]
30    [0.85, 0.8375, 0.9375, 0.9367088607594937, 0.9367088607594937]
40    [0.825, 0.85, 0.9375, 0.9240506329113924, 0.9493670886075949]
50    [0.825, 0.8375, 0.9375, 0.9240506329113924, 0.9746835443037974]
75    [0.825, 0.85, 0.9375, 0.9240506329113924, 0.9620253164556962]
100    [0.8, 0.8625, 0.925, 0.9240506329113924, 0.9620253164556962]
```

In [99]:
```
import matplotlib.pyplot as plt
# plot the raw observations
for k in k_choices:
    accuracies = k_to_accuracies[k]
    plt.scatter([k] * len(accuracies), accuracies)

# plot the trend line with error bars that correspond to standard deviation
accuracies_mean = np.array([np.mean(v) for k,v in sorted(k_to_accuracies.items())])
accuracies_std = np.array([np.std(v) for k,v in sorted(k_to_accuracies.items())])
plt.errorbar(k_choices, accuracies_mean, yerr=accuracies_std)
plt.title('Cross Validation')
plt.xlabel('k values')
plt.ylabel('accuracy')
plt.show()
```



Problem 4 (20 points): Based on the cross-validation results above, choose the best value for k. Repeat problem 2 with this k (using the entire training data set).

In [102]:
```
#problem 4 solution
k_best=0
accuracy_max = 0
for k in k_to_accuracies:
    avg_acc = np.mean(k_to_accuracies[k])*100
    print(k,"= ",avg_acc)
    if accuracy_max < avg_acc:
        accuracy_max = avg_acc
        k_best = k

print("Best Value of k =",k_best," having accuracy = ",accuracy_max)
```

```
1 =  84.4430379747
3 =  85.6993670886
5 =  88.2120253165
7 =  88.4683544304
9 =  88.9651898734
11 =  88.9651898734
13 =  89.2088607595
15 =  89.4588607595
20 =  89.4588607595
30 =  89.9683544304
40 =  89.7183544304
50 =  89.9746835443
75 =  89.9715189873
100 =  89.4715189873
Best Value of k = 50  having accuracy =  89.9746835443
```

In [103]:
```
# repeat problem 2 solution with best k
knn_test=knn_predict(Xtr, Ytr, Xts, 40)
#initializing variables for true-positives, true-negatives, false-nagatives and false-positives
true_pos=0
true_neg=0
false_pos=0
false_neg=0
for i in range(len(knn_test)):
    if knn_test[i]==1 and Yts[i]==1:
        true_pos=true_pos+1
    elif knn_test[i]==0 and Yts[i]==0:
        true_neg=true_neg+1
    elif knn_test[i]==0 and Yts[i]==1:
        false_neg=false_neg+1
    else:
        false_pos=false_pos+1

print('Accuracy = ',(true_pos+true_neg)/(true_pos+true_neg+false_neg+false_pos))
print('Recall = ',(true_pos)/(true_pos+false_neg))
print('Precision = ',(true_pos)/(true_pos+false_pos))
```

```
Accuracy =  0.9239766081871345
Recall =  0.9242424242424242
Precision =  0.976
```

Note: these extra credit problems are optional and only increase your score marginally.

Extra Credit Problem 1 (5 points): Plot decision boundaries for the best k, similar to how it is done here: http://scikit-learn.org/stable/auto_examples/neighbors/plot_classification.html (http://scikit-learn.org/stable/auto_examples/neighbors/plot_classification.html)

Extra Credit Problem 2 (5 points): In problem 1 above, re-write the compute_dist_mat() function with no loops. That may seem non-intuitive, but it is possible to compute the L2 distances using matrix operations (matrix multiplication, addition, etc.) without explicitly doing the double for loop. The advantage of using matrix operations is that they are highly optimized and enable "vectorization", and for such computations can give 10-100x speed improvements.

In [104]:
```
#Extra Credit Problem 2
m = Xts.shape[0] # x has shape (m, d)
n = Xtr.shape[0] # y has shape (n, d)
x2 = np.sum(Xts**2, axis=1).reshape((m, 1))
y2 = np.sum(Xtr**2, axis=1).reshape((1, n))
xy = Xts.dot(Xtr.T) # shape is (m, n)
dists = np.sqrt(x2 + y2 - 2*xy) # shape is (m, n)
print(dists[0][0])
print(np.shape(dists))
```

```
0.743608282863
(171, 398)
```

In [ ]:
```

```