In [18]:
```
# Import packages
import numpy as np
from sklearn.neighbors import KNeighborsClassifier as KNN
from sklearn.model_selection import KFold
from sklearn.metrics import mean_squared_error
from sklearn.neighbors import NearestNeighbors

# First, transcribe the data set to a numpy array. The y values will be
# represented as follows:
# asterisk == 1, and spade == 0. All entered in the order (ID, x1, x2, y)
data = np.array([[1, 8.18, 0.0, 1],
                 [2, 6.9, 5.0, 0],
                 [3, 5.67, 4.0, 0],
                 [4, -0.69, 9.0, 0],
                 [5, 8.16, 0.0, 1],
                 [6, 3.75, 13.0, 0],
                 [7, 2.6, 0.0, 1],
                 [8, 5.49, 3.0, 0],
                 [9, 8.69, -6.0, 1],
                 [10, -0.38, -7.0, 1]])

# Split into input and output
X = data[:, [1, 2]]
y = data[:, 3]
```

In [19]:
```python
# A) What is the leave one out cross-validation error of 1NN on this dataset
knn = KNN(n_neighbors=1, metric='euclidean')

# For leave one out, we can use kfold with k=n (n = number of samples in data
# set)
loo = KFold(n_splits=len(y))


# Create a blank list to append results to. Loop through all of the splits and
# do KNN, check results, append to list.
results = []
run = 1
for train_index, test_index in loo.split(X):
    entry = []
    xtrain, xtest = X[train_index], X[test_index]
    ytrain, ytest = y[train_index], y[test_index]
    knn.fit(xtrain, ytrain)
    prediction = knn.predict(xtest)
    mse = mean_squared_error(ytest, prediction)
    entry.append(run)
    entry.append(mse)
    run += 1
    results.append(entry)

# Take the results and put together a test MSE
# n = number of observations
n = 10
errorsum = 0
for result in results:
    errorsum += result[1]

TestMSE = errorsum/n
print('A) The leave-one-out cross-validation error using 1NN with this dataset '
      'is: ' + str(TestMSE))
print('\n')
```

```
A) The leave-one-out cross-validation error using 1NN with this dataset is: 0.1
```

In [20]:
```
# B) Report the 3 nearest neighbors for data points 3 and 5 respectively.
# Set up neighbor identifier, extract points, and create new datasets without
# those points.
neigh = NearestNeighbors(n_neighbors=3)
p3 = data[2, [1, 2]]
# Remove the point so it isn't selected - this shifts all indices down 1 past
# i = 1 (so prev 3 is now 2, etc.) Therefore, need to add 1 to any index greater
# than 1. For p5 need to add 1 to any index greater than 3
X3 = np.delete(X, 2, axis=0)
p5 = data[4, [1, 2]]
X5 = np.delete(X, 4, axis=0)
neigh.fit(X3)
ans1 = neigh.kneighbors([p3], return_distance=False)
ans1 = ans1.tolist()
ans1 = sorted(ans1[0])
new1 = []
for elem in ans1:
    if elem > 1:
        elem += 1
        new1.append(elem)
    else:
        new1.append(elem)
neigh.fit(X5)
ans2 = neigh.kneighbors([p5], return_distance=False)
ans2 = ans2.tolist()
ans2 = sorted(ans2[0])
new2 = []
for elem in ans2:
    if elem > 3:
        elem += 1
        new2.append(elem)
    else:
        new2.append(elem)

p3neighs = data[new1, :]
p5neighs = data[new2, :]
print('B) The 3 nearest data points to points 3 and 5 are listed below.')
print('The following array gives the 3 nearest neighbors of data point 3 in the '
      'format \n[ID x1 x2 y]: ')
print(p3neighs)
print('\n')
print('The following array gives the 3 nearest neighbors of data point 5 in the '
      'same \nformat as above: ')
print(p5neighs)
print('\n')
```

```
B) The 3 nearest data points to points 3 and 5 are listed below.
The following array gives the 3 nearest neighbors of data point 3 in the format
[ID x1 x2 y]:
[[2.   6.9  5.   0.  ]
 [5.   8.16 0.   1.  ]
 [8.   5.49 3.   0.  ]]


The following array gives the 3 nearest neighbors of data point 5 in the same
format as above:
```

```
[[1.   8.18 0.   1.  ]
 [3.   5.67 4.   0.  ]
 [8.   5.49 3.   0.  ]]
```

In [21]:
```
# C) What is the 3-folded cross-validation error of 1NN on this dataset
# Build function to split data set
def kfold_split(data, i):
    check = i - 1
    testdata = []
    traindata = []
    for row in data:
        if row[0] % 3 == check:
            testdata.append(row)
        else:
            traindata.append(row)
    testdata = np.array(testdata)
    traindata = np.array(traindata)
    return testdata, traindata


test1, train1 = kfold_split(data, 1)
test2, train2 = kfold_split(data, 2)
test3, train3 = kfold_split(data, 3)
data1 = [test1, train1]
data2 = [test2, train2]
data3 = [test3, train3]

def datasplitter(data):
    testdata = data[0]
    traindata = data[1]
    xtest = testdata[:, [1, 2]]
    ytest = testdata[:, 3]
    xtrain = traindata[:, [1, 2]]
    ytrain = traindata[:, 3]
    return xtrain, ytrain, xtest, ytest


X1tr, y1tr, X1test, y1test = datasplitter(data1)
X2tr, y2tr, X2test, y2test = datasplitter(data2)
X3tr, y3tr, X3test, y3test = datasplitter(data3)

knn = KNN(n_neighbors=3, metric='euclidean')
results = []
k = 3

knn.fit(X1tr, y1tr)
prediction = knn.predict(X1test)
mse = mean_squared_error(y1test, prediction)
results.append(mse)

knn.fit(X2tr, y2tr)
prediction = knn.predict(X2test)
mse = mean_squared_error(y2test, prediction)
results.append(mse)

knn.fit(X3tr, y3tr)
prediction = knn.predict(X3test)
mse = mean_squared_error(y3test, prediction)
results.append(mse)
```

```
errorsum = 0
for item in results:
    errorsum += item

error = errorsum/k

print('C) The 3-folded cross-validation error using 3NN on this dataset is:\n'
       + str(error) + '\n')
```

```
C) The 3-folded cross-validation error using 3NN on this dataset is:
0.38888888888888884
```

In [22]:

```
# D) Based on the results of (a) and (c), can we determine which is a better
# classifier, 1NN or 3NN? Why?

# Based on the results of (a) and (c) we cannot definitively determine whether
# 1NN or 3NN is the better classifier. We cannot determine which is better
# because we are using two different types of error measurement between the two
# modes of classification. Further, by specifying the method by which we
# partition the dataset, we remove the element of random sampling typically
# present in determining k-fold cross-validation error, and therefore may skew
# the true error associated with this method. To summarize, the information
# presented here is insufficient to determine which is better because: 1) the
# error may not be representative of the true k-fold cross-validation error,
# and 2) the error calculation methods are not comparable.
```

In [ ]:

```
```