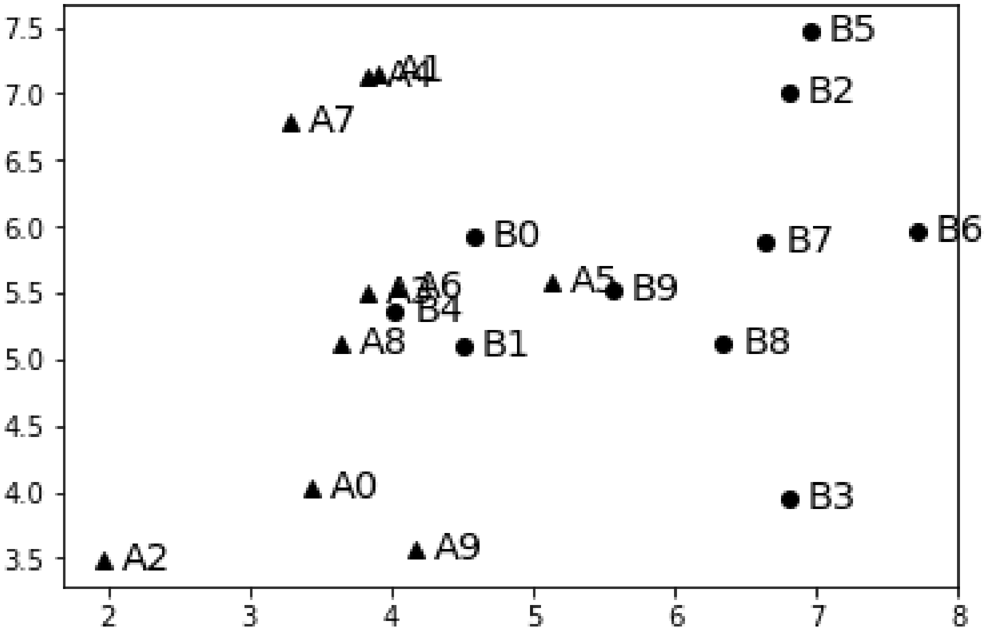
# Assignment 1

August 26, 2018

```
In [8]: import pylab, random, math, string

        def minkowskiDist(v1, v2, p):
            """Assumes v1 and v2 are equal-length arrays of numbers
               Returns Minkowski distance of order p between v1 and v2"""
            dist = 0.0
            for i in range(len(v1)):
                dist += abs(v1[i] - v2[i])**p
            return dist**(1.0/p)


        class Example(object):

            def __init__(self, name, features, label = None):
                #Assumes features is an array of numbers
                self.name = name
                self.features = features
                self.label = label

            def dimensionality(self):
                return len(self.features)

            def getFeatures(self):
                return self.features[:]

            def getLabel(self):
                return self.label

            def getName(self):
                return self.name

            def distance(self, other):
                return minkowskiDist(self.features, other.getFeatures(), 2)

            def __str__(self):
                return self.name +':'+ str(self.features) + ':' + str(self.label)
```

```
class Cluster(object):

    def __init__(self, examples):
        """Assumes examples is a list of example of type exampleType"""
        self.examples = examples
        self.centroid = self.computeCentroid()

    def update(self, examples):
        """Replace the examples in the cluster by new examples
           Return how much the centroid has changed"""
        oldCentroid = self.centroid
        self.examples = examples
        self.centroid = self.computeCentroid()
        return oldCentroid.distance(self.centroid)

    def computeCentroid(self):
        vals = pylab.array([0.0]*self.examples[0].dimensionality())
        for e in self.examples:
            vals += e.getFeatures()
        centroid = Example('centroid', vals/len(self.examples))
        return centroid

    def getCentroid(self):
        return self.centroid

    def variability(self):
        totDist = 0.0
        for e in self.examples:
            totDist += (e.distance(self.centroid))**2
            return totDist

    def members(self):
        for e in self.examples:
            yield e

    def __str__(self):
        names = []
        for e in self.examples:
            names.append(e.getName())
        names.sort()
        result = 'Cluster with centroid '\
                 + str(self.centroid.getFeatures()) + ' contains:\n  '
        for e in names:
            result = result + e + ', '
        return result[:-2]


def dissimilarity(clusters):
    totDist = 0.0
    for c in clusters:
        totDist += c.variability()
    return totDist

def trykmeans(examples, numClusters, numTrials, verbose = False):

    """Calls kmeans numTrials times and returns the result with the
          lowest dissimilarity"""

    best = kmeans(examples, numClusters, verbose)
    minDissimilarity = dissimilarity(best)
    trial = 1
    while trial < numTrials:
        try:
            clusters = kmeans(examples, numClusters, verbose)
        except ValueError:
            continue
        currDissimilarity = dissimilarity(clusters)
        if currDissimilarity < minDissimilarity:
            best = clusters
            minDissimilarity = currDissimilarity
        trial += 1
    return best

def kmeans(examples, k, verbose = False):
    """Assumes examples is a list of examples of type exampleType,
         k is a positive int, verbose is a Boolean
       Returns a list containing k clusters. If verbose is
         True it prints result of each iteration of k-means"""
    #Get k randomly chosen initial centroids
    initialCentroids = random.sample(examples, k)
    clusters = []
    for e in initialCentroids:
        clusters.append(Cluster([e]))
    #Iterate until centroids do not change
    converged = False
    numIterations = 0
    while not converged:
        numIterations += 1
        #Create a list containing k distinct empty lists
        newClusters = []
        for i in range(k):
            newClusters.append([])

        #Associate each example with closest centroid
        for e in examples:
            #Find the centroid closest to e
            smallestDistance = e.distance(clusters[0].getCentroid())
            index = 0
            for i in range(1, k):
                distance = e.distance(clusters[i].getCentroid())
                if distance < smallestDistance:
                    smallestDistance = distance
                    index = i
            #Add e to the list of examples for the appropriate cluster
            newClusters[index].append(e)

        for c in newClusters:
            if len(c) == 0:
                raise ValueError('Empty Cluster')

        #Upate each cluster; check if a centroid has changed
        converged = True
        for i in range(k):
            if clusters[i].update(newClusters[i]) > 0.0:
                converged = False
        if verbose:
            print('Iteration #' + str(numIterations))
            for c in clusters:
                print(c)
            print('') #add blank line
    return clusters

def genDistribution(xMean, xSD, yMean, ySD, n, namePrefix):
    samples = []
    for s in range(n):
        x = random.gauss(xMean, xSD)
        y = random.gauss(yMean, ySD)
        samples.append(Example(namePrefix+str(s), [x, y]))
    return samples

def plotSamples(samples, marker):
    xVals, yVals = [], []
    for s in samples:
        x = s.getFeatures()[0]
        y = s.getFeatures()[1]
        pylab.annotate(s.getName(), xy = (x, y),
                       xytext = (x+0.13, y-0.07),
                       fontsize = 'x-large')
        xVals.append(x)
        yVals.append(y)
    pylab.plot(xVals, yVals, marker)

def contrivedTest(numTrials, k, verbose = False):
    xMean = 3
    xSD = 1
    yMean = 5
    ySD = 1
    n = 10
    d1Samples = genDistribution(xMean, xSD, yMean, ySD, n, 'A')
    plotSamples(d1Samples, 'k^')
    d2Samples = genDistribution(xMean+3, xSD, yMean+1, ySD, n, 'B')
    plotSamples(d2Samples, 'ko')
    clusters = trykmeans(d1Samples + d2Samples, k, numTrials, verbose)
    print('Final result')
    for c in clusters:
        print('', c)

contrivedTest(1, 2, True)
```

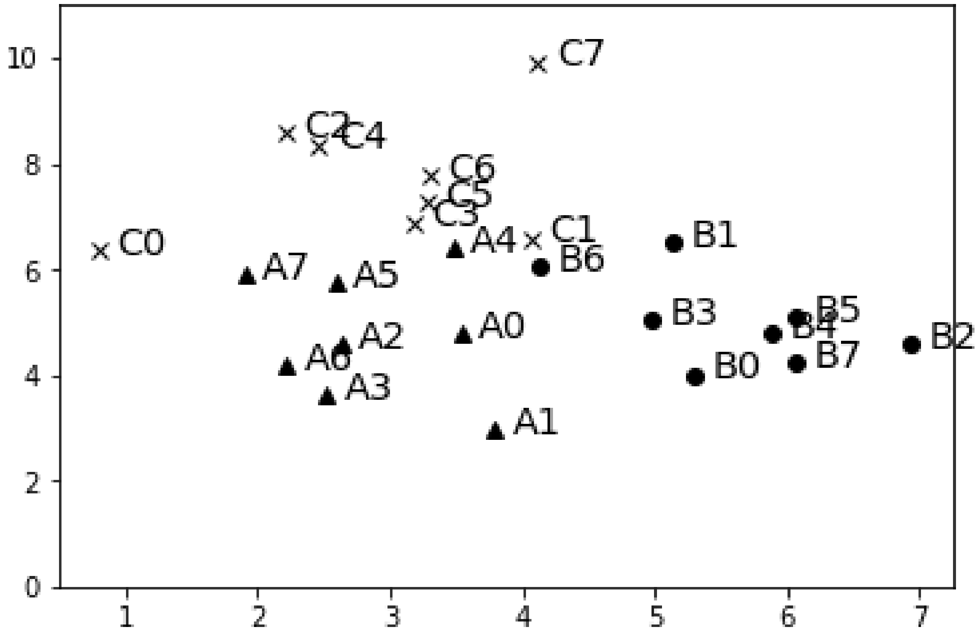
```
Iteration #1
Cluster with centroid [4.31659854 5.30185927] contains:
  A0, A1, A2, A3, A4, A5, A6, A7, A8, A9, B0, B1, B3, B4, B8, B9
Cluster with centroid [7.02870312 6.57702296] contains:
  B2, B5, B6, B7

Iteration #2
Cluster with centroid [3.99392126 5.41124215] contains:
  A0, A1, A2, A3, A4, A5, A6, A7, A8, A9, B0, B1, B4, B9
Cluster with centroid [6.87758191 5.89674168] contains:
  B2, B3, B5, B6, B7, B8

Iteration #3
Cluster with centroid [3.87357133 5.40339345] contains:
  A0, A1, A2, A3, A4, A5, A6, A7, A8, A9, B0, B1, B4
Cluster with centroid [6.6891374  5.84196077] contains:
  B2, B3, B5, B6, B7, B8, B9

Iteration #4
Cluster with centroid [3.87357133 5.40339345] contains:
  A0, A1, A2, A3, A4, A5, A6, A7, A8, A9, B0, B1, B4
Cluster with centroid [6.6891374  5.84196077] contains:
  B2, B3, B5, B6, B7, B8, B9

Final result
 Cluster with centroid [3.87357133 5.40339345] contains:
  A0, A1, A2, A3, A4, A5, A6, A7, A8, A9, B0, B1, B4
 Cluster with centroid [6.6891374  5.84196077] contains:
  B2, B3, B5, B6, B7, B8, B9
```



```
In [ ]: # Results:
        # We can see 4 itirations and the final result, they cotain
        # A`s and B`s cluster examples and the graph where we can see
        # those results in circles and triangles randomly placed by the functions
        # A`s and B`s are vectors of a set of examples
        # and two distributions were created of 10 examples
```

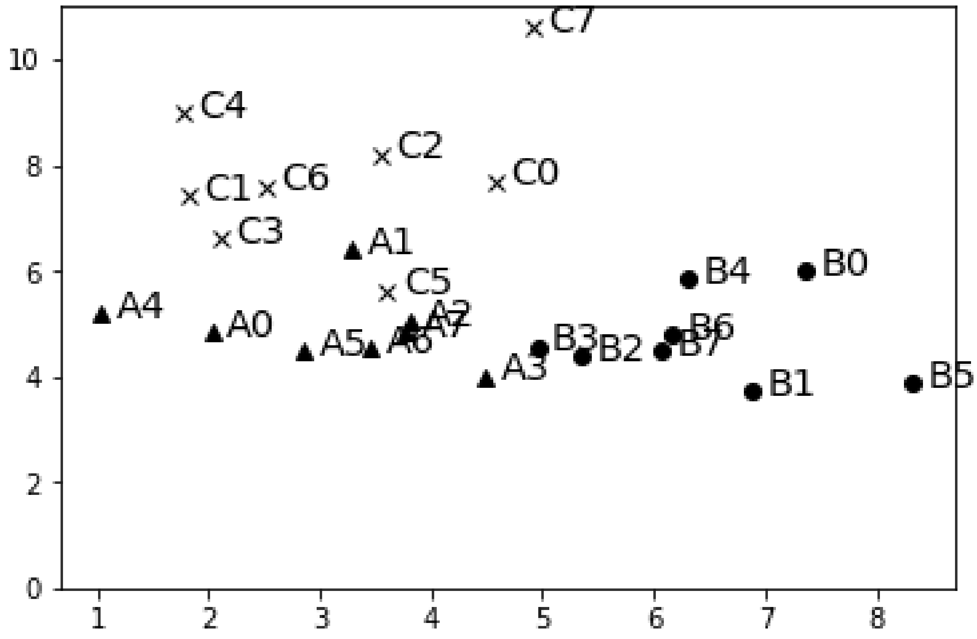
```
In [5]: def contrivedTest2(numTrials, k, verbose = False):
            xMean = 3
            xSD = 1
            yMean = 5
            ySD = 1
            n = 8
            d1Samples = genDistribution(xMean,xSD, yMean, ySD, n, 'A')
            plotSamples(d1Samples, 'k^')
            d2Samples = genDistribution(xMean+3,xSD,yMean, ySD, n, 'B')
            plotSamples(d2Samples, 'ko')
            d3Samples = genDistribution(xMean, xSD, yMean+3, ySD, n, 'C')
            plotSamples(d3Samples, 'kx')
            clusters = trykmeans(d1Samples + d2Samples + d3Samples, k, numTrials, verbose)
            pylab.ylim(0,11)
            print('Final result has dissimilarity', round(dissimilarity(clusters), 3))
            for c in clusters:
                print('', c)

        contrivedTest2(40, 2)
```

```
Final result has dissimilarity 1.323
 Cluster with centroid [4.35210255 6.53763646] contains:
  A4, B1, B2, B3, B4, B5, B6, B7, C1, C2, C3, C4, C5, C6, C7
 Cluster with centroid [2.80907687 4.68359213] contains:
  A0, A1, A2, A3, A5, A6, A7, B0, C0
```

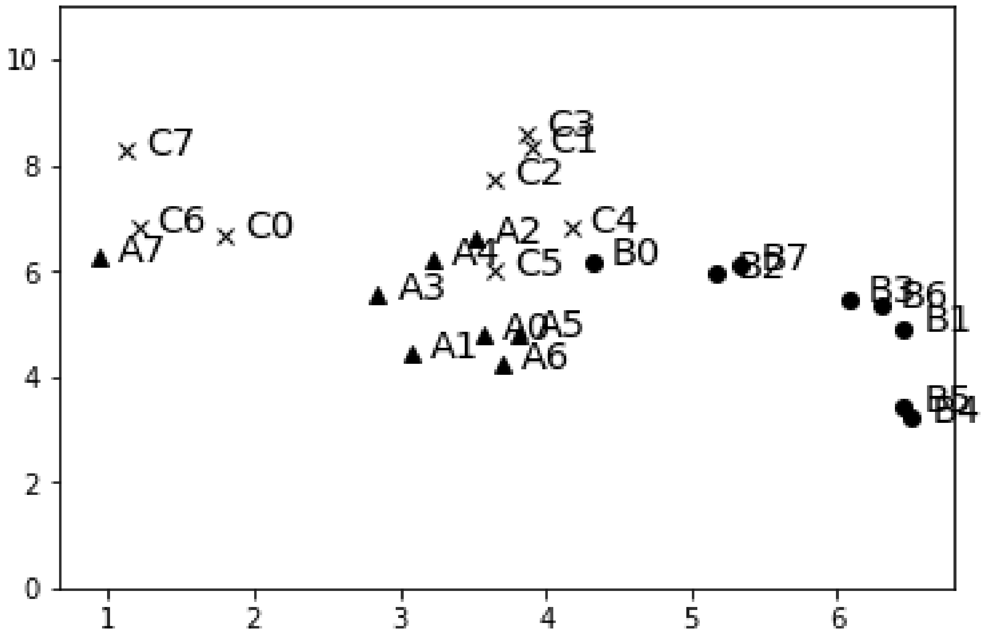


```
In [6]: contrivedTest2(40, 3)
```

```
Final result has dissimilarity 6.134
 Cluster with centroid [6.42983481 4.70847475] contains:
  B0, B1, B2, B3, B4, B5, B6, B7
 Cluster with centroid [3.04880189 5.15443488] contains:
  A0, A1, A2, A3, A4, A5, A6, A7, C3, C5
 Cluster with centroid [3.19329817 8.42063265] contains:
  C0, C1, C2, C4, C6, C7
```



```
In [7]: contrivedTest2(40, 6)

Final result has dissimilarity 0.658
 Cluster with centroid [3.80564136 8.21969969] contains:
  C1, C2, C3
 Cluster with centroid [3.54393708 4.56539868] contains:
  A0, A1, A5, A6
 Cluster with centroid [6.04217429 4.92137599] contains:
  B1, B2, B3, B4, B5, B6, B7
 Cluster with centroid [1.11934381 8.27570377] contains:
  C7
 Cluster with centroid [3.62415804 6.22903879] contains:
  A2, A3, A4, B0, C4, C5
 Cluster with centroid [1.31658007 6.58545796] contains:
  A7, C0, C6
```



```
In [ ]: # Results:
        # Three overlapping Gaussian distributions with various values ok k
        # The outcomes show dissimilarity of results we call of k (2, 3, 6)
        # and we can see that every time k is bigger the dissimilarity is smaller
```

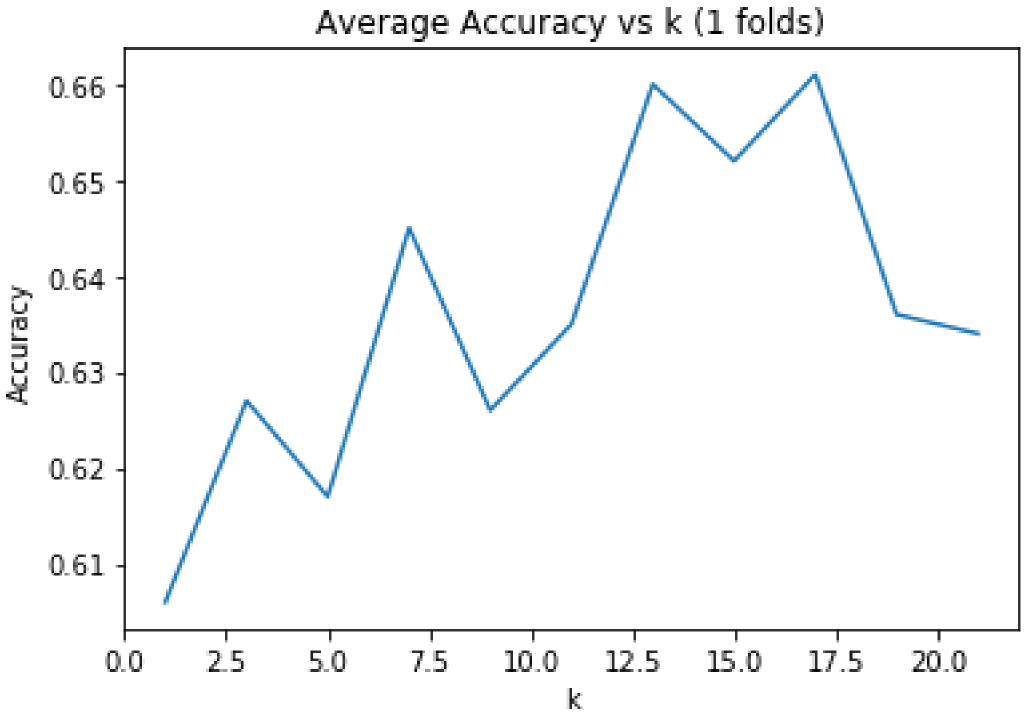
# Assignment 2

August 26, 2018

```
In [2]: # Assignment 2, p.408 -412
        # k -nearest neighbour

        import random
        import pylab
        import math


        def getBMData(filename):
            """Plot of Boston marathon based on given data """
            data = {}
            f = open('bm_results2012.txt', 'r')
            line = f.readline()
            data['name'], data['gender'], data['age'] = [], [], []
            data['division'], data['country'], data['time'] = [], [], []
            while line != '':
                split = line.split(',')
                data['name'].append(split[0])
                data['gender'].append(split[1])
                data['age'].append(split[2])
                data['division'].append(split[3])
                data['country'].append(split[4])
                data['time'].append(float(split[5][:-1]))
                line = f.readline()
            f.close()
            return data

        def accuracy(truePos, falsePos, trueNeg, falseNeg):
            numerator = truePos + trueNeg
            denominator = truePos + trueNeg + falsePos + falseNeg
            return numerator/denominator

        def sensitivity(TruePos, falseNeg):
            try:
                return truePos/(truePos+falseNeg)
            except ZeroDivisionError:
                return float('nan')
```

```python
def specificity(trueNeg, falsePos):
    try:
        return trueNeg/(trueNeg + falsePos)
    except ZeroDivisionError:
        return float('nan')

def posPredVal(truePos, falsePos):
    try:
        return truePos/(truePos + falsePos)
    except ZeroDivisionError:
        return float('nan')

def negPredVal(trueNeg, falseNeg):
    try:
        return trueNeg/(trueNeg + falseNeg)
    except ZeroDivisionError:
        return float('nan')

def getStats(truePos, falsePos, trueNeg, falseNeg, toPrint = True):
    accur = accuracy(truePos, falsePos, trueNeg, falseNeg)
    sens = sensitivity(truePos, falseNeg)
    spec = specificity(trueNeg, falsePos)
    ppv = posPredVal(truePos, falsePos)
    if toPrint:
        print(' Accuracy = ', round(accur, 3))
        print(' Sensitivity =', round(sens, 3))
        print(' Specificity =', round(spec, 3))
        print(' Pos. Pred. Val =', round(ppv, 3))
    return(accur, sens, spec, ppv)

class Runner(object):
    def __init__(self, gender, age, time):
        self.featureVec = (age, time)
        self.label = gender

    def featureDist(self, other):
        dist = 0.0
        for i in range(len(self.featureVec)):
            dist = dist + abs(float(self.featureVec[i]) - float(other.featureVec[i]))**
        return dist**0.5

    def getTime(self):
        return self.featureVec[1]
    def getAge(self):
        return self.featureVec[0]
    def getLabel(self):
        return self.label
    def getFeatures(self):
        return self.featureVec

    def __str__(self):
        return str(self.getAge()) + ', ' + str(self.getTime())\
                                  + ', ' + self.label


def buildMarathonExamples(fileName):
    data = getBMData(fileName)
    examples = []
    for i in range(len(data['age'])):
        a = Runner(data['gender'][i], data['age'][i], data['time'][i])
        examples.append(a)
    return examples

def divide80_20(examples):
    sampleIndices = random.sample(range(len(examples)), len(examples)//5)
    trainingSet, testSet = [], []
    for i in range(len(examples)):
        if i in sampleIndices:
            testSet.append(examples[i])
        else:
            trainingSet.append(examples[i])
    return trainingSet, testSet


def findKNearest(example, exampleSet, k):
    kNearest, distances = [], []
    for i in range(k):
        kNearest.append(exampleSet[i])
        distances.append(example.featureDist(exampleSet[i]))
    maxDist = max(distances)
    for e in exampleSet[k:]:
        dist = example.featureDist(e)
        if dist < maxDist:
            maxIndex = distances.index(maxDist)
            kNearest[maxIndex] = e
            distances[maxIndex] = dist
            maxDist = max(distances)
    return kNearest, distances

def KNearestClassify(training, testSet, label, k):
    """ Assumes"""
    truePos, falsePos, trueNeg, falseNeg = 0, 0, 0, 0
    for e in testSet:
        nearest, distances = findKNearest(e, training, k)
        numMatch = 0
        for i in range(len(nearest)):
            if nearest[i].getLabel() == label:
                numMatch += 1
        if numMatch > k//2:
            if e.getLabel() == label:
                truePos += 1
            else:
                falsePos += 1
        else:
            if e.getLabel() != label:
                trueNeg += 1
            else:
                falseNeg += 1
    return truePos, falsePos, trueNeg, falseNeg

def prevalenceClassify(training, testSet, label):
    numWithLabel = 0
    for e in training:
        if e.getLabel() == label:
            numWithLabel += 1
    probLabel = numWithLabel/len(training)
    truePos, falsePos, trueNeg, falseNeg = 0, 0, 0, 0
    for e in testSet:
        if random.random() < probLabel:
            if e.getLabel() == label:
                truePos += 1
            else:
                falsePos += 1
        else:
            if e.getLabel() != label:
                trueNeg += 1
            else:
                falseNeg += 1
    return truePos, falsePos, trueNeg, falseNeg

def findK(training, minK, maxK, numFolds, label):
    accuracies = []
    for k in range(minK, maxK + 1, 2):
        score = 0.0
        for i in range(numFolds):
            fold = random.sample(training, min(5000, len(training)))
            examples, testSet = divide80_20(fold)
            truePos, falsePos, trueNeg, falseNeg =\
            KNearestClassify(examples,testSet, label, k)
            score += accuracy(truePos, falsePos, trueNeg, falseNeg)
        accuracies.append(score/numFolds)
    pylab.plot(range(minK, maxK +1, 2), accuracies)
    pylab.title('Average Accuracy vs k (' + str(numFolds) + ' folds)')
    pylab.xlabel('k')
    pylab.ylabel('Accuracy')

examples = buildMarathonExamples('bm_results2012.txt')
training, testSet = divide80_20(examples)
reducedTraining = random.sample(training, len(training)//10)
truePos, falsePos, trueNeg, falseNeg =\
          KNearestClassify(reducedTraining, testSet, 'M', 9)
getStats(truePos, falsePos, trueNeg, falseNeg)

findK(training, 1, 21, 1, 'M')
```

```
Accuracy =  0.658
Sensitivity = 0.712
Specificity = 0.583
Pos. Pred. Val = 0.703
```



```
In [ ]: # Results:
        # The goal of the code is to predict the gender of a runner
        # We use k-nearest neighbors classifier which predicts the
        # gender of a runner based on runner`s age and finishing time.
        # on the output we can see 65% accuracy given age and finishing time
        # once we chose the value for k (9), then we produced a plot which shows
        # us a better accuracy and how it was changing
```

# Assignment 3

August 26, 2018

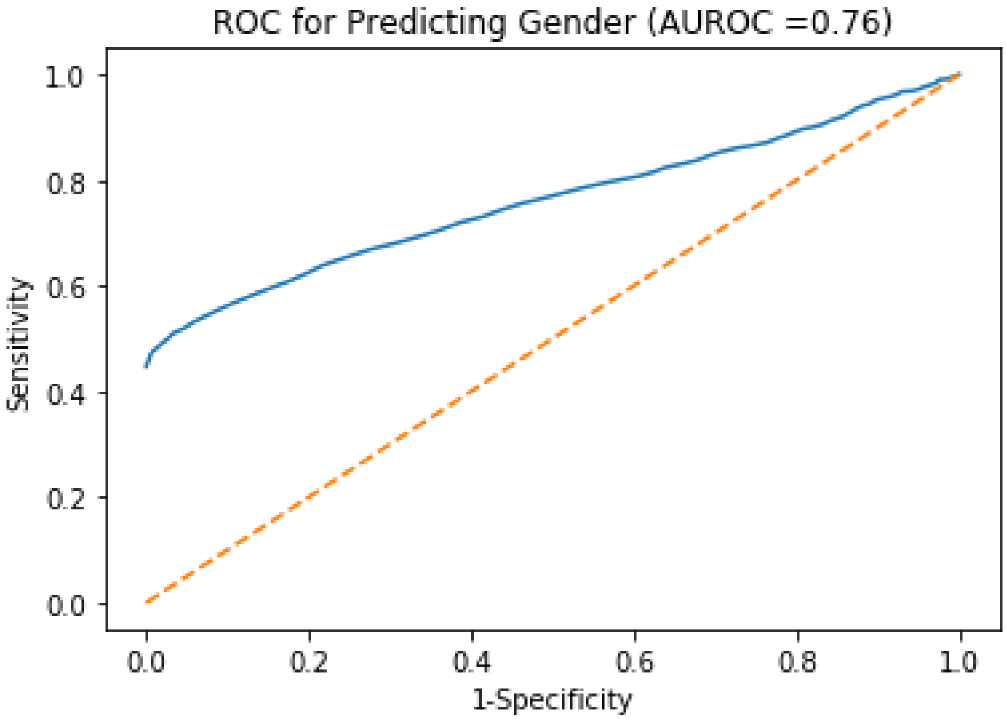
```
In [15]: # finger exercise p 424
         # ROC and AUROC

         import random
         import pylab
         import math
         import sklearn.linear_model

         def getBMData(filename):
             """Plot of Boston marathon based on given data """
             data = {}
             f = open('bm_results2012.txt', 'r')
             line = f.readline()
             data['name'], data['gender'], data['age'] = [], [], []
             data['division'], data['country'], data['time'] = [], [], []
             while line != '':
                 split = line.split(',')
                 data['name'].append(split[0])
                 data['gender'].append(split[1])
                 data['age'].append((float(split[2])))
                 data['division'].append(split[3])
                 data['country'].append(split[4])
                 data['time'].append(float(split[5][:-1]))
                 line = f.readline()
             f.close()
             return data

         def accuracy(truePos, falsePos, trueNeg, falseNeg):
             numerator = truePos + trueNeg
             denominator = truePos + trueNeg + falsePos + falseNeg
             return numerator/denominator

         def sensitivity(TruePos, falseNeg):
             try:
                 return truePos/(truePos+falseNeg)
             except ZeroDivisionError:
                 return float('nan')
```

```
def specificity(trueNeg, falsePos):
    try:
        return trueNeg/(trueNeg + falsePos)
    except ZeroDivisionError:
        return float('nan')

def posPredVal(truePos, falsePos):
    try:
        return truePos/(truePos + falsePos)
    except ZeroDivisionError:
        return float('nan')

def negPredVal(trueNeg, falseNeg):
    try:
        return trueNeg/(trueNeg + falseNeg)
    except ZeroDivisionError:
        return float('nan')

def getStats(truePos, falsePos, trueNeg, falseNeg, toPrint = True):
    accur = accuracy(truePos, falsePos, trueNeg, falseNeg)
    sens = sensitivity(truePos, falseNeg)
    spec = specificity(trueNeg, falsePos)
    ppv = posPredVal(truePos, falsePos)
    if toPrint:
        print(' Accuracy = ', round(accur, 3))
        print(' Sensitivity =', round(sens, 3))
        print(' Specificity =', round(spec, 3))
        print(' Pos. Pred. Val =', round(ppv, 3))
    return(accur, sens, spec, ppv)


class Runner(object):
    def __init__(self, gender, age, time):
        self.featureVec = (age, time)
        self.label = gender

    def featureDist(self, other):
        dist = 0.0
        for i in range(len(self.featureVec)):
            dist = dist + abs(float(self.featureVec[i]) - float(other.featureVec[i]))*
        return dist**0.5

    def getTime(self):
        return self.featureVec[1]
    def getAge(self):
        return self.featureVec[0]
    def getLabel(self):
        return self.label
    def getFeatures(self):
        return self.featureVec

    def __str__(self):
        return str(self.getAge()) + ', ' + str(self.getTime())\
                                  + ', ' + self.label


def buildMarathonExamples(fileName):
    data = getBMData(fileName)
    examples = []
    for i in range(len(data['age'])):
        a = Runner(data['gender'][i], data['age'][i], data['time'][i])
        examples.append(a)
    return examples

def divide80_20(examples):
    sampleIndices = random.sample(range(len(examples)), len(examples)//5)
    trainingSet, testSet = [], []
    for i in range(len(examples)):
        if i in sampleIndices:
            testSet.append(examples[i])
        else:
            trainingSet.append(examples[i])
    return trainingSet, testSet

# f24.15
def applyModel(model, testSet, label, prob = 0.5):

    testFeatureVecs = [e.getFeatures() for e in testSet]
    probs = model.predict_proba(testFeatureVecs)
    truePos, falsePos, trueNeg, falseNeg = 0, 0, 0, 0
    for i in range(len(probs)):
        if probs[i][1] > prob:
            if testSet[i].getLabel() == label:
                truePos += 1
            else:
                falsePos += 1
        else:
            if testSet[i].getLabel() != label:
                trueNeg += 1
            else:
                falseNeg += 1
    return truePos, falsePos, trueNeg, falseNeg

examples = buildMarathonExamples('bm_results2012.txt')
training, test = divide80_20(examples)
```

```
featureVecs, labels = [], []
for e in training:
    featureVecs.append([float(e.getAge()), float(e.getTime())])
    labels.append(e.getLabel())
model = sklearn.linear_model.LogisticRegression().fit(featureVecs, labels)
print('Feature weights for label M:', 'age =', str(round(model.coef_[0][0],3)) +
      ',' , 'time =', round(model.coef_[0][1], 3))
truePos, falsePos, trueNeg, falseNeg = applyModel(model, test, 'M', 0.5)
getStats(truePos, falsePos, trueNeg, falseNeg)

def buildROC(model, testSet, label, title, plot = True):
    xVals, yVals = [], []
    p = 0.0
    while p <= 1.0:
        truePos, falsePos, trueNeg,falseNeg = applyModel(model, testSet, label, p)
        xVals.append(1.0 - specificity(trueNeg, falsePos))
        yVals.append(sensitivity(truePos, falseNeg))
        p += 0.01
    auroc = sklearn.metrics.auc(xVals,yVals,True)
    if plot:
        pylab.plot(xVals, yVals)
        pylab.plot([0,1], [0,1,], '--')
        pylab.title(title + ' (AUROC =' + str(round(auroc,3)) + ')')
        pylab.xlabel('1-Specificity')
        pylab.ylabel('Sensitivity')
    return auroc

buildROC(model, test, 'M', 'ROC for Predicting Gender')
```

```
Feature weights for label M: age = 0.056, time = -0.012
 Accuracy =  0.636
 Sensitivity = 0.808
 Specificity = 0.394
 Pos. Pred. Val = 0.652
```

```
Out[15]: 0.7599362635899656
```



```
In [ ]: # Plot shows ROC curve and calculated AUROC wic is 0.766 in my case,
        # Feature weights for label M, accuracy, sensitivity, specifity and Pos.Pred.Val
        # Since the number of training examples shouls very from 10 to 1010,
        # I understood that we would have to go back to function divide80_20
        # and change the range from 10 to 1010 in increment of 50
```