```
from __future__ import division  # ensures float division for all arithmetic
from __future__ import print_function
from collections import Counter
from math import exp
from math import log

import pdb

trainFile = input("Choose a training file:")
testFile = input("Choose a testing file:")

#data structure holding the count of when Xi's equal to 1 or 0 conditioned on Y =
    0
y0 = Counter()
y0["0"] = Counter() #counter of when Xi's equal 0
y0["1"] = Counter() #counter of when Xi's equal 1

y1 = Counter()
y1["0"] = Counter()
y1["1"] = Counter()

learningRate = 0.0001

def naiveBayesTrainingMLE(trainFile):
    with open(trainFile, "r") as f:
        n = 0
        vectorLen = 0
        numVectors = 0
        numY0 = 0
        numY1 = 0
        for line in f:
            if n == 0:
                vectorLen = int(line)
            if n == 1:
                numVectors = int(line)
            if n > 1:
                splitLine = line.split(":")
                vector = splitLine[0].split(" ")
                if splitLine[1].strip() == "0":
                    updateDataStructures("0",vector)
                    numY0 += 1
                if splitLine[1].strip() == "1":
                    updateDataStructures("1",vector)
                    numY1 += 1
            n += 1
        naiveBayesTest(testFile, numY0, numY1)

def naiveBayesTrainingLaplace(trainFile):
    with open(trainFile, "r") as f:
        n = 0
        vectorLen = 0
        numVectors = 0
        numY0 = 2
        numY1 = 2
        for line in f:
            if n == 0:
                vectorLen = int(line)
                initializeDataStructureForLaplace(vectorLen)
            if n == 1:
                numVectors = int(line)
            if n > 1:
                splitLine = line.split(":")
                vector = splitLine[0].split(" ")
                if splitLine[1].strip() == "0":
                    updateDataStructures("0", vector)
                    numY0 += 1
                if splitLine[1].strip() == "1":
                    updateDataStructures("1", vector)
                    numY1 += 1
            n += 1
        naiveBayesTest(testFile, numY0, numY1)

def initializeDataStructureForLaplace(vectorLen):
    for i in xrange(vectorLen):
        y0["0"][str(i)] = 1
        y0["1"][str(i)] = 1
        y1["0"][str(i)] = 1
        y1["1"][str(i)] = 1

def updateDataStructures(y, vector):
    for i,item in enumerate(vector):
        if y == "0":
            y0[item][str(i)] += 1
        if y == "1":
            y1[item][str(i)] += 1

def naiveBayesTest(testFile, numY0, numY1):
    with open(testFile, "r") as f:
        n = 0
        vectorLen = 0
        numVectors = 0
        numAccurate = 0
        probY0 = float(numY0) / float(numY0 + numY1)
        probY1 = float(numY1) / float(numY0 + numY1)
        for line in f:
            if n == 0:
                vectorLen = int(line)
            if n == 1:
                numVectors = int(line)
            if n > 1:
                splitLine = line.split(":")
                vector = splitLine[0].split(" ")
                prediction = 0
                Y1Prediction = computePredictionForY1(vector)
                Y0Prediction = computePredictionForY0(vector)
                Y1Prediction += log(probY1)
                Y0Prediction += log(probY0)
                if Y1Prediction > Y0Prediction:
                    prediction = 1
                if float(splitLine[1].strip()) == prediction:
                    numAccurate += 1
            n += 1
        print (numAccurate / numVectors)

def computePredictionForY1(vector):
    prediction = 0
    for i,item in enumerate(vector):
        if item == "0":
            prediction += log((y1[item][str(i)] + pow(1, -10)) / (y1[item][str(i)
                ] + y1["1"][str(i)]))
        if item == "1":
            prediction += log((y1[item][str(i)] + pow(1, -10)) / (y1[item][str(i)
                ] + y1["0"][str(i)]))
    return prediction

def computePredictionForY0(vector):
    prediction = 0
    for i,item in enumerate(vector):
        if item == "0":
            prediction += log((y0[item][str(i)] + pow(1, -10)) / (y0[item][str(i)
                ] + y0["1"][str(i)]))
        if item == "1":
            prediction += log((y0[item][str(i)] + pow(1, -10)) / (y0[item][str(i)
                ] + y0["0"][str(i)]))
    return prediction

def logisticRegressionTraining(trainFile):
    with open(trainFile, "r") as f:
        lines, vectorLen, thetaVector, yValues = storeFileData(f)
        for i in xrange(10000): #number of steps
            gradientVector = vectorLen * [0]
            for j,line in enumerate(lines): #looping over training instances
                 z = makeWeightedSum(thetaVector, line, vectorLen)
                 constant = yValues[j] - (1 / (1 + exp(-z)))
                 for index in xrange(vectorLen): #looping through gradientVector
                     gradientVector[index] += line[index] * constant
            thetaVector = updateThetas(thetaVector, gradientVector, vectorLen)
        logisticRegressionTest(testFile, thetaVector)

def storeFileData(f):
    lines = []
    n = 0
    vectorLen = 0
    numVector = 0
    thetaVector = [0]
    yValues = []
    for line in f:
        if n == 0:
            vectorLen = int(line) + 1
            thetaVector = vectorLen * [0]
        if n == 1:
            numVector = int(line)
        if n > 1:
            splitLine = line.split(":")
            yValues.append(float(splitLine[1].strip()))
            vectorInstance = splitLine[0].split(" ") #getting an instance vector
            vectorInstance.append("1")
            vectorInstance = map(int, vectorInstance)
            lines.append(vectorInstance)
        n += 1
    return lines, vectorLen, thetaVector, yValues

def makeWeightedSum(thetaVector, vectorInstance, vectorLen):
    z = 0
    for index in xrange(vectorLen):
       # pdb.set_trace()
        z += thetaVector[index] * vectorInstance[index]
    return z

def updateThetas(thetaVector, gradientVector, vectorLen):
    for i in xrange(vectorLen):
       # pdb.set_trace()
        thetaVector[i] += learningRate * gradientVector[i]
    return thetaVector

def logisticRegressionTest(testFile, thetaVector):
    with open(testFile, "r") as f:
        n = 0
        vectorLen = 0
        numVector = 0
        numAccurate = 0
        numTotal = 0
        for line in f:
            if n == 0:
                vectorLen = int(line) + 1
            if n == 1:
                numVector = int(line)
            if n > 1:
                splitLine = line.split(":")
                vector = splitLine[0].split(" ")
                vector.append("1")
                vector = map(int, vector)
                z = makeWeightedSum(thetaVector, vector, vectorLen)
                probability = 1 / (1 + exp(-z))
                if probability > 0.5: #if prediction is 1
                    if splitLine[1].strip() == "1":
                        numAccurate += 1
                else:
                    if splitLine[1].strip() == "0": #if prediction is 0
                        numAccurate += 1
                numTotal += 1
            n += 1
        print(numAccurate / numTotal)
naiveBayesTrainingMLE(trainFile)
naiveBayesTrainingLaplace(trainFile)
logisticRegressionTraining(trainFile)
```