Christopher Thompson
Naïve Bayes Classifier Project
CS 4342

During this project, I noticed some reasonable values for the predicted output from the table 5.9, which leads me to believe that the classifier is working correctly. However, for the single prediction provided, home-owner, married, 50.7k income, I noticed some very odd values. The classifier classified the probability that the individual will/will not default is 0%. To accommodate for the categorical attribute probabilities I utilized the *M-Estimate* to mitigate any instances of misclassification due to a class-conditional probability being zero. For the continuous variable attributes, I used a Gaussian distribution to represent the class-conditional probability for continuous attributes. The results of the prediction are below

---------- Prediction 1) ----------

Probability that the individual will default is: 0.0%

Probability that the individual will not default is: 0.0%

---------- General Predictions ----------

0) Probability that the individual WILL default is: [4.5677023]%

0) Probability that the individual WILL NOT default is: [10.65797204]%

1) Probability that the individual WILL default is: [4.86503825]%

1) Probability that the individual WILL NOT default is: [11.35175591]%

2) Probability that the individual WILL default is: [3.7167091]%

2) Probability that the individual WILL NOT default is: [8.67232122]%

3) Probability that the individual WILL default is: [4.5677023]%

3) Probability that the individual WILL NOT default is: [10.65797204]%

4) Probability that the individual WILL default is: [4.79474862]%

4) Probability that the individual WILL NOT default is: [11.18774679]%

5) Probability that the individual WILL default is: [3.08351859]%

5) Probability that the individual WILL NOT default is: [7.1948767]%

6) Probability that the individual WILL default is: [0.18141464]%

6) Probability that the individual WILL NOT default is: [0.42330083]%

7) Probability that the individual WILL default is: [4.49080268]%

7) Probability that the individual WILL NOT default is: [10.47853958]%

8) Probability that the individual WILL default is: [4.00695118]%

8) Probability that the individual WILL NOT default is: [9.34955275]%

```
import pandas as pd
import numpy as np

def createData():
    '''create the data from page 153'''
    df = pd.DataFrame()

    ''' 1 --> Yes    0 --> No '''
    homeOwner = [1, 0, 0, 1, 0, 0, 1, 0, 0, 0]

    ''' Single --> 1 Married --> 2 Divorced --> 3'''
    maritalStatus = [1, 2, 1, 2, 3, 1, 3, 1, 2, 1]

    annualIncome = [120000, 100000, 70000, 120000, 95000,
                    60000, 220000, 85000, 75000, 90000]
```

```
    ''' Did the borrower default?   1 --> Yes    0 --> No '''
    defaultedClass = [0, 0, 0, 0, 1, 0, 0, 1, 0, 1]

    df['home-owner'] = homeOwner
    df['married'] = maritalStatus
    df['income'] = annualIncome
    df['defaulted'] = defaultedClass

    return df

def priorProbability(xlist, defaultList):
    x_yes_y_yes = 0
    x_yes_y_no = 0
    x_no_y_yes = 0
    x_no_y_no = 0

    for i in range(len(defaultList)):
        if xlist[i] == 1 and defaultList[i] == 1:
            x_yes_y_yes += 1
        if xlist[i] == 1 and defaultList[i] == 0:
            x_yes_y_no += 1
        if xlist[i] == 0 and defaultList[i] == 1:
            x_no_y_yes += 1
        if xlist[i] == 0 and defaultList[i] == 0:
            x_no_y_no += 1

    numDefault = (defaultList == 1).sum()
    numNoDefault = (defaultList == 0).sum()

    tempList = [(x_yes_y_yes / numDefault), (x_no_y_yes / numDefault),
                (x_yes_y_no / numNoDefault), (x_no_y_no / numNoDefault)]

    df = pd.DataFrame([tempList], columns=['x_given_y', 'xPrime_given_y',
'x_given_yPrime', 'xPrime_given_yPrime'])
    return df

def priorProbability_Categorical(xlist, defaultList):
    x_0_y_yes = 0
    x_1_y_yes = 0
    x_2_y_yes = 0
    x_0_y_no = 0
    x_1_y_no = 0
    x_2_y_no = 0

    for i in range(len(defaultList)):
        if xlist[i] == 1 and defaultList[i] == 1:
            x_0_y_yes += 1
        if xlist[i] == 2 and defaultList[i] == 1:
            x_1_y_yes += 1
        if xlist[i] == 3 and defaultList[i] == 1:
            x_2_y_yes += 1
        if xlist[i] == 1 and defaultList[i] == 0:
            x_0_y_no += 1
        if xlist[i] == 2 and defaultList[i] == 0:
            x_1_y_no += 1
        if xlist[i] == 3 and defaultList[i] == 0:
            x_2_y_no += 1

    numDefault = (defaultList == 1).sum()
    numNoDefault = (defaultList == 0).sum()
```

```
    tempListYes = [
        x_0_y_yes,
        x_1_y_yes,
        x_2_y_yes
    ]

    for i in range(len(tempListYes)):
        tempListYes[i] = mEstimate(
            numDefault, tempListYes[i],
            tempListYes[i] / numDefault, numDefault)

    tempListNo = [
        x_0_y_no,
        x_1_y_no,
        x_2_y_no
    ]

    for i in range(len(tempListNo)):
        tempListNo[i] = mEstimate(
            numNoDefault, tempListNo[i],
            tempListNo[i] / numNoDefault, numNoDefault)

    tempList = tempListYes + tempListNo

    df = pd.DataFrame([tempList], columns=[
        'single_defaultYes',
        'married_defaultYes',
        'divorced_defaultYes',
        'single_defaultNo',
        'married_defaultNo',
        'divorced_defaultNo'
    ])
    return df

def estimateContinuousProbability(income, x_i):
    ''' calculate the variance of the income array using an unbiased estimator line
'''
    variance = np.var(income, dtype=np.float64, ddof=1)
    mean = np.mean(income, dtype=np.float64) # calculate mean
    s = np.sqrt(variance) # get the multiplier
    rootTwoPi = np.sqrt(2 * np.pi)
    mult_1 = 1 / (rootTwoPi * s)
    mult_2 = np.exp(-1 * (np.square((x_i - mean)) / (2 * variance)))
    return mult_1 * mult_2

def getContinuousProbabilities(incomeList):
    probList = []
    for i in range(len(incomeList)):
        x = estimateContinuousProbability(incomeList, incomeList[i])
        probList.append(x)
    df = pd.DataFrame([probList], columns=[str(i) + 'k' for i in incomeList],
dtype=np.float64)
    return df

def mEstimate(n, n_c, m, p):
    return n_c + (m * p) / n + m

def probability(df):
    x_1 = df['home-owner'].values
    x_2 = df['married'].values
    Y = df['defaulted'].values
```

```
    ownProb = float((x_1 == 1).sum() / len(x_1))
    rentProb = float((x_1 == 0).sum() / len(x_1))
    singleProb = float((x_2 == 1).sum() / len(x_2))
    marriedProb = float((x_2 == 2).sum() / len(x_2))
    divorcedProb = float((x_2 == 3).sum() / len(x_2))
    tempList = [ownProb, rentProb, singleProb, marriedProb, divorcedProb]
    return pd.DataFrame([tempList], columns=['ownProb', 'rentProb', 'singleProb',
                                             'marriedProb', 'divorcedProb'],
dtype=np.float64)

def makePrediction(denom, binaryValue, categoricalValue, incomeValue, defaultProb):
    numer = binaryValue * categoricalValue * incomeValue
    result = (numer / denom) * defaultProb
    return result

def constructProbDf(df, priorProbDf, priorProbCatDf):
    tempList = []
    col1 = 0
    col2 = 0

    stdProbDf = probability(df)
    for i in range(9):
        denomVal = 1
        if 1 in df['home-owner'].index and 1 in df['defaulted'].index:
            col1 = priorProbDf['x_given_y'].iloc[0]
            denomVal *= stdProbDf['ownProb'].iloc[0]

        if 1 in df['home-owner'].index and 0 in df['defaulted'].index:
            col1 = priorProbDf['x_given_yPrime'].iloc[0]
            denomVal *= stdProbDf['ownProb'].iloc[0]

        if 0 in df['home-owner'].index and 1 in df['defaulted'].index:
            col1 = priorProbDf['xPrime_given_y'].iloc[0]
            denomVal *= stdProbDf['rentProb'].iloc[0]

        if 0 in df['home-owner'].index and 0 in df['defaulted'].index:
            col1 = priorProbDf['xPrime_given_yPrime'].iloc[0]
            denomVal *= stdProbDf['rentProb'].iloc[0]

        if 1 in df['married'].index and 1 in df['defaulted'].index:
            col2 = priorProbCatDf['single_defaultYes'].iloc[0]
            denomVal *= stdProbDf['singleProb'].iloc[0]

        if 2 in df['married'].index and 1 in df['defaulted'].index:
            col2 = priorProbCatDf['married_defaultYes'].iloc[0]
            denomVal *= stdProbDf['marriedProb'].iloc[0]

        if 3 in df['married'].index and 1 in df['defaulted'].index:
            col2 = priorProbCatDf['divorced_defaultYes'].iloc[0]
            denomVal *= stdProbDf['divorcedProb'].iloc[0]

        if 1 in df['married'].index and 0 in df['defaulted'].index:
            col2 = priorProbCatDf['single_defaultNo'].iloc[0]
            denomVal *= stdProbDf['singleProb'].iloc[0]

        if 2 in df['married'].index and 0 in df['defaulted'].index:
            col2 = priorProbCatDf['married_defaultNo'].iloc[0]
            denomVal *= stdProbDf['marriedProb'].iloc[0]

        if 3 in df['married'].index and 0 in df['defaulted'].index:
            col2 = priorProbCatDf['divorced_defaultNo'].iloc[0]
```

```
            denomVal *= stdProbDf['divorcedProb'].iloc[0]

        tempList = tempList + [[col1, col2, denomVal]]
    tempList = np.array(tempList)
    return pd.DataFrame(tempList, columns=['home', 'maritalStatus', 'denomVal'],
dtype=np.float64)



def main():
    df = createData() # create the data table
    x_1 = df['home-owner'].values
    x_2 = df['married'].values
    x_3 = df['income'].values
    Y = df['defaulted'].values
    newDf = priorProbability(x_1, Y)
    dfC = priorProbability_Categorical(x_2, Y)
    continuousProbList = getContinuousProbabilities(x_3)
    stdProbs = probability(df)
    print(stdProbs)
    print(df)
    print("\n\n---------------------- Prediction 1) Home owner = yes, Marital status =
Married, Income = 50.7K -----------------------")
    continuousValue = estimateContinuousProbability(income=x_3, x_i=50700.00)
    defaultProb = (Y == 1).sum() / len(Y)
    noDefaultProb = (Y == 0).sum() / len(Y)
    predictionYes = makePrediction(0.3 * 0.3 * continuousValue, binaryValue= 0.0,
categoricalValue= 0.0,
                                   incomeValue=continuousValue,
defaultProb=defaultProb)
    predictionNo = makePrediction(0.3 * 0.3 * continuousValue,
binaryValue=newDf['x_given_y'].iloc[0],
                                  categoricalValue=dfC['married_defaultYes'].iloc[0],
                                  incomeValue=continuousValue,
defaultProb=noDefaultProb)
    defaultYesProb = "\nProbability that the individual will default is:   " +
str(predictionYes * 100) + "%"
    defaultNoPro = "\nProbability that the individual will not default is:   " +
str(predictionNo * 100) + "%"
    print(defaultYesProb)
    print(defaultNoPro)
    mainProbDf = constructProbDf(df, newDf, dfC)
    homePriorProb = mainProbDf['home'].values
    maritalPriorProb = mainProbDf['maritalStatus'].values
    incomePriorProb = np.reshape(continuousProbList.values, newshape=(10, 1))
    denomVals = mainProbDf['denomVal'].values

    tempListYes = []
    tempListNo = []

    print("\n\n---------------------- General Predictions -----------------------")
    for i in range(len(homePriorProb)):
        predYes = makePrediction(denomVals[i], homePriorProb[i], maritalPriorProb[i],
incomePriorProb[i], defaultProb)
        predNo = makePrediction(denomVals[i], homePriorProb[i], maritalPriorProb[i],
incomePriorProb[i], noDefaultProb)
        defaultYesProb = "\n" + str(i) + ") Probability that the individual WILL
default is:   " + str(predYes * 100) + "%"
        defaultNoPro = "\n" + str(i) + ") Probability that the individual WILL NOT
default is:   " + str(predNo * 100) + "%"
        print(defaultYesProb)
```

```
        print(defaultNoPro)
        tempListYes = tempListYes + [[predYes]]
        tempListNo = tempListNo + [[predNo]]

main()
```