HW2_TiagoGoncalves

October 29, 2018

1. Regression

Consider the following data:

In [65]:
```
import numpy as np

data = np.array([[368, 15, 1.7], [340, 16, 1.5], [665, 25, 2.8], [954, 40, 5.0], [331

x_data = data[:, 0:2]
y = data[:, 2]

print("Features: \n", x_data)
print("\nLabels: ", y)
```

```
Features:
 [[368.  15.]
 [340.  16.]
 [665.  25.]
 [954.  40.]
 [331.  15.]]

Labels:  [1.7 1.5 2.8 5.  1.3]
```

a) What's the regression solution for f(y)=w1x1+w2x2?

In [66]:
```
#Using Normal Equations Algorithm
weights = np.linalg.inv(np.dot(x_data.transpose(), x_data))
weights = np.dot(weights, x_data.transpose())
weights = np.dot(weights, y)

#Assuming no bias, so w0=0!
print("Solution is: ", "f(y) = ", str(weights[0]), "x1 + ", str(weights[1]), "x2")
```

```
Solution is:  f(y) =  0.0027731787719238003 x1 +  0.048757601172728926 x2
```

```
In [67]: #Assuming bias: Insert "virtual feature" = 1
         vf = np.array([1, 1, 1, 1, 1]).reshape(5,1)
         xdata = np.concatenate((vf, x_data), axis=1)

         #Using Normal Equations Algorithm
         weights = np.linalg.inv(np.dot(xdata.transpose(), xdata))
         weights = np.dot(weights, xdata.transpose())
         weights = np.dot(weights, y)


         print("Solution with bias is: ", "f(y) = ", str(weights[0]), " + ", str(weights[1]), "
```

```
Solution with bias is:  f(y) =  -0.6526112271325144  +  0.0009363257725659782 x1 +  0.11778650
```

b) Trying to improve the fitting, we collect another feature x3:

```
In [68]: x3_feature = np.array([383, 356, 690, 994, 346]).reshape(5, 1)
         new_xdata = np.concatenate((x_data, x3_feature), axis=1)

         print("Now, features are: \n", new_xdata)
```

```
Now, features are:
 [[368.  15. 383.]
 [340.  16. 356.]
 [665.  25. 690.]
 [954.  40. 994.]
 [331.  15. 346.]]
```

What's now the solution for f(y)=w1x1+w2x2+w3x3? Is it unique?

```
In [69]: #Using Normal Equations Algorithm
         weights = np.linalg.inv(np.dot(new_xdata.transpose(), new_xdata))
         weights = np.dot(weights, new_xdata.transpose())
         weights = np.dot(weights, y)
```

```
In [70]: #Assuming no bias, so w0=0!
         print("Solution is: ", "f(y) = ", str(weights[0]), "x1 + ", str(weights[1]), "x2 + ",
```

```
Solution is:  f(y) =  -0.0018798828125000007 x1 +  0.03164062500000009 x2 +  0.0096679687499999
```

```
In [71]: #Assuming bias: Insert "virtual feature" = 1
         xdata = np.concatenate((vf, new_xdata), axis=1)

         #Using Normal Equations Algorithm
         weights = np.linalg.inv(np.dot(xdata.transpose(), xdata))
         weights = np.dot(weights, xdata.transpose())
```

```
weights = np.dot(weights, y)


print("Solution is: ", "f(y) = ", str(weights[0]), " + ", str(weights[1]), "x1 + ", s

print("\nThe solution is not unique if we take into account:", "\na) If we assume, or
     "\nb) If we use another algorithm to compute weights, such as Least-Mean-Square
     "\nc) If the inverse of (X.T * X) doesn't exist, one could not use the Normal Equ
```

```
Solution is:  f(y) =  38.695953670989084  +  -0.044042968749999994 x1 +  0.09824218750000013 x2

The solution is not unique if we take into account:
a) If we assume, or not, the existence of bias;
b) If we use another algorithm to compute weights, such as Least-Mean-Square (LMS) or Steepest
c) If the inverse of (X.T * X) doesn't exist, one could not use the Normal Equations Algorithm
```

Classification

2. Consider the data in 'heightWeightData.txt'. The first column is the class label (1=male, 2=female), the second column is height, the third weight.

a) Write a Matlab/Python function to model each class data as follows: assuming that height and weight are independent given the class, model the height using a histogram with bins breakpoints at every 10 cm (10, 20, 30, . . . , 170, 180, 190, . . .) and the weight with a Gaussian distribution with the mean and variance learnt from the data using maximum likelihood estimation.

You can use suitable functions in Matlab/Python like histcounts. The function should receive as input the training data and the test data, making prediction (male/female) for the test point.

In [72]:
```python
#Define Functions
#Modelling Height
def height_model(heights_total, height_input):
    bins = []
    for i in range(int(height_input.min()), int(height_input.max()), 10):
        bins.append(i)
        #print("Bins : ", bins)

    hist, bin_edges = np.histogram(heights_total, bins=bins, density=True)
    return hist, bin_edges

#Plotting Height Histogram
#plt.hist(height, bins=bins, density=True)  # arguments are passed to np.histogram
#plt.title("Height Histograms")
#plt.show()

#Modelling Weight
def weight_model(weight_input, weight_input_mean, weight_input_var):
```

```
    ll_weight = norm.pdf(weight_input, weight_input_mean, weight_input_var)
    return ll_weight

#print(height_model(male_heights))
#print(height_model(female_heights))

#print(weight_model(male_weights))
#print(weight_model(female_weights))

#print(prob_female)
#print(prob_male)

#Check Bin Location
def check_bin(h_input, bin_edges):
    diffs = []
    for edge in range(len(bin_edges)):
        diffs.append(abs((bin_edges[edge] - h_input)))

    min_diff = np.min(diffs)

    for i in range(len(diffs)):
        if diffs[i] == min_diff:
            index = i

    if index == (len(bin_edges) - 1):
        index -=1

    return index

#Computing Posterior Probability
def prediction(class_probabilities, prior_probability):
    predict = class_probabilities * prior_probability

    return predict
```

In [73]:
```
#Classification Functions
def classification(training_data, test_data):
    labels = training_data[:,0]
    #Normalize labels: 0=male; 1=female
    labels = labels-1

    #Get number of males and females in data
    nr_males = 0
    nr_females = 0

    for i in labels:
        if i==0:
            nr_males +=1
```

```
    elif i==1:
        nr_females +=1

#print(nr_males, nr_females)

#Get male and female indexes
male_index = []
female_index = []
for index in range(int(labels.shape[0])):
    if labels[index] == 0:
        male_index.append(index)
    elif labels[index] == 1:
        female_index.append(index)


#Features
height = training_data[:, 1]
weight = training_data[:, 2]

#Female Features
female_heights = height[female_index]
female_weights = weight[female_index]
#Male Features
male_heights = height[male_index]
male_weights = weight[male_index]

#Prior Distribution
#Compute Prior Probabilities in each class
#prob_female = (nr_females/(nr_females+nr_males))
#prob_male = (nr_males/(nr_females+nr_males))
#Let's Assume Equal Prior Probabilities
prob_female = 0.5
prob_male = 0.5

#Save previous info's in arrays
weight_input_means = [np.average(male_weights), np.average(female_weights)]
weight_input_var =  [np.var(male_weights), np.var(female_weights)]
#Weight input mean & var
weight_mean = np.average(weight)
weight_var = np.var(weight)

#Evaluate Weight Likelihood
mw = weight_model(weight_input=test_data[1], weight_input_mean=weight_input_means
fw = weight_model(weight_input=test_data[1], weight_input_mean=weight_input_means

#Evaluate Height
#Male
male_hist, male_bin_edges = height_model(heights_total=height, height_input=male_
```

```
    male_bin_index = check_bin(bin_edges=male_bin_edges, h_input=test_data[0])
    mh = male_hist[male_bin_index]
    #Female
    female_hist, female_bin_edges = height_model(heights_total=height, height_input=fe
    female_bin_index = check_bin(bin_edges=female_bin_edges, h_input=test_data[0])
    fh = female_hist[female_bin_index]

    #Class Probs
    class_probs = [mh*mw, fh*fw]
    #Compute Naive Bayes
    #Male
    male_pred = prediction(class_probabilities=class_probs[0], prior_probability=prob_
    female_pred = prediction(class_probabilities=class_probs[1], prior_probability=pro

    #Classify
    if male_pred > female_pred:
        pred_class = 0
    elif female_pred > male_pred:
        pred_class = 1

    return pred_class, male_pred, female_pred
```

b) Use the previous function to make predictions (male / female) for the following test points:

[165 80]t, [181 65]t, [161 57]t and [181 77]t.

c) What's the estimated p([165 80]t | male)?

```
In [74]: #Read data
         import matplotlib.pyplot as plt
         from scipy.stats import norm

         data = np.genfromtxt('heightWeightData.txt', delimiter=',')
         test_1 = [165, 80]
         test_2 = [181, 65]
         test_3 = [161, 57]
         test_4 = [181, 77]
         test = [test_1, test_2, test_3, test_4]
         predictions = []
         male_probs = []
         female_probs = []
         for point in test:
             pred, male, female = classification(training_data=data, test_data=point)
             predictions.append(pred)
             male_probs.append(male)
             female_probs.append(female)

         print("Predictions: ", predictions)
         print("The estimated p([165 80] | male) is: ", male_probs[0])
```

```
Predictions:  [0, 1, 0, 1]
The estimated p([165 80] | male) is:  4.184096006667823e-05
```

Fundamentals

3. An experiment consists in randomly choosing values between 0 and 1 (a scalar in [0,1]) until the sum of the observed values is above 1.

a) In python/matlab simulate the execution of 1000000 experiments. What's the estimated number of values one needs to pick until the sum exceeds one?

```
In [75]: def experiment(number_of_experiments):
             experiments = []

             for i in range(number_of_experiments):
                 counts = 0
                 sum = 0

                 while counts <= 1000000 and sum <=1:
                     sum += np.random.random_sample()
                     counts += 1
                 experiments.append(counts)

             return np.array(experiments), np.mean(np.array(experiments)), np.around(np.mean(np
```

```
In [82]: exp_array, mean_nr_values, rounded_nr_values = experiment(1000000)

         print("For 1000000 experiments:", "\nMean number of values is: ", mean_nr_values, "\nS
```

```
For 1000000 experiments:
Mean number of values is:  2.71905
So, the rounded estimated number of values one needs to pick would be:  3.0
```

```
In [83]: hist, hist_edges = np.histogram(a=exp_array, bins=[0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 1
```

b) [1 point only in 20] Compute analytically the expected value of the number of values one needs to pick until the sum exceeds one.

```
In [84]: #Compute probs P(X = x)
         probs = []
         for number in hist:
             prob = number/1000000
             probs.append(prob)
```

```
In [85]: i = 0
         for prob in probs:
             print("P(X = "+str(i)+") is:", prob)
             i+=1
```

```
P(X = 0) is: 0.0
P(X = 1) is: 0.0
P(X = 2) is: 0.498649
P(X = 3) is: 0.334737
P(X = 4) is: 0.125303
P(X = 5) is: 0.033084
P(X = 6) is: 0.006894
P(X = 7) is: 0.001141
P(X = 8) is: 0.000173
P(X = 9) is: 1.6e-05
P(X = 10) is: 3e-06
P(X = 11) is: 0.0
P(X = 12) is: 0.0
P(X = 13) is: 0.0
P(X = 14) is: 0.0
P(X = 15) is: 0.0
P(X = 16) is: 0.0
P(X = 17) is: 0.0
P(X = 18) is: 0.0
P(X = 19) is: 0.0
```

In [86]:
```
#Compute Analytically - Nr_Of_Values = SUM xn*P(X|x=xn)
nr_of_values = 0
i = 0
for prob in probs:
    nr_of_values+= (prob * i)
    i+=1

print("The estimated number of values one needs to pick would be: ", nr_of_values)
```

```
The estimated number of values one needs to pick would be:  2.7190500000000006
```