Travis Munyer

Naïve Bayes Classification Discussion

September 7, 2021

1. Code

import numpy as np  
  
def train\_and\_test(training\_data, labels, testing\_data):  
 c2\_sample\_count = 0  
 c1\_sample\_count = 0  
 training\_sample\_total = training\_data.shape[0]  
 testing\_sample\_total = testing\_data.shape[0]  
 features\_per\_sample = testing\_data.shape[1]  
 c1\_0\_count = 0  
 c1\_1\_count = 0  
 c2\_0\_count = 0  
 c2\_1\_count = 0  
  
 *# Compute prior probability for each class.* for i in range(training\_sample\_total):  
 if (labels[i] == 1):  
 c2\_sample\_count += 1  
 temp\_ones\_c2, temp\_zeros\_c2 = count\_0s\_and\_1s(training\_data[i])  
 c2\_1\_count += temp\_ones\_c2  
 c2\_0\_count += temp\_zeros\_c2  
 elif (labels[i] == 0):  
 c1\_sample\_count += 1  
 temp\_ones\_c1, temp\_zeros\_c1 = count\_0s\_and\_1s(training\_data[i])  
 c1\_1\_count += temp\_ones\_c1  
 c1\_0\_count += temp\_zeros\_c1  
  
 *# Compute conditional probability for each class and feature possibility (this assumes all values are 0 or 1).* p\_of\_1\_given\_c2 = c2\_1\_count / (c2\_1\_count + c2\_0\_count)  
 p\_of\_0\_given\_c2 = c2\_0\_count / (c2\_1\_count + c2\_0\_count)  
 p\_of\_1\_given\_c1 = c1\_1\_count / (c1\_1\_count + c1\_0\_count)  
 p\_of\_0\_given\_c1 = c1\_0\_count / (c1\_1\_count + c1\_0\_count)  
  
 *# Compute prior probability for each class and feature possbility* c2\_probability = c2\_sample\_count / training\_sample\_total  
 c1\_probability = c1\_sample\_count / training\_sample\_total  
  
 print()  
 print("Training Results: ")  
 print("P(C1): " + str(c1\_probability) + ", P(C2): " + str(c2\_probability))  
 print("P(1|C1): " + str(p\_of\_1\_given\_c1) + ", P(0|C1): " + str(p\_of\_0\_given\_c1))  
 print("P(1|C2): " + str(p\_of\_1\_given\_c2) + ", P(0|C2): " + str(p\_of\_0\_given\_c2))  
 print()  
  
 print("Testing Results: ")  
 for j in range(testing\_sample\_total):  
 *# Initialized to 1 for multiplication* sample\_j\_probability\_x\_given\_c2 = 1  
 sample\_j\_probability\_x\_given\_c1 = 1  
  
 for n in range(features\_per\_sample):  
 *# Multiply by respective '1' case conditional probabilities.* if (testing\_data[j][n] == 1):  
 sample\_j\_probability\_x\_given\_c2 \*= p\_of\_1\_given\_c2  
 sample\_j\_probability\_x\_given\_c1 \*= p\_of\_1\_given\_c1  
  
 *# Multiply by respective '0' case conditional probabilities.* elif (testing\_data[j][n] == 0):  
 sample\_j\_probability\_x\_given\_c2 \*= p\_of\_0\_given\_c2  
 sample\_j\_probability\_x\_given\_c1 \*= p\_of\_0\_given\_c1  
  
 print("Testing results for sample: " + str(testing\_data[j]))  
 print("P(x|C1): " + str(sample\_j\_probability\_x\_given\_c1) + ", P(x|C2): " + str(sample\_j\_probability\_x\_given\_c2))  
  
 *# Compute probability for each class.* p\_c2\_given\_x = (sample\_j\_probability\_x\_given\_c2 \* c2\_probability) \  
 / (sample\_j\_probability\_x\_given\_c2 \* c2\_probability + sample\_j\_probability\_x\_given\_c1 \* c1\_probability)  
 p\_c1\_given\_x = (sample\_j\_probability\_x\_given\_c1 \* c1\_probability) \  
 / (sample\_j\_probability\_x\_given\_c2 \* c2\_probability + sample\_j\_probability\_x\_given\_c1 \* c1\_probability)  
  
 print("P(C1|x): " + str(p\_c1\_given\_x) + ", P(C2|x): " + str(p\_c2\_given\_x))  
 class\_result = ''  
 if (p\_c1\_given\_x < p\_c2\_given\_x):  
 class\_result = "Class 2."  
 elif (p\_c2\_given\_x < p\_c1\_given\_x):  
 class\_result = "Class 1."  
 else:  
 class\_result = "Class probabilities are equal, cannot make decision."  
  
 print("This sample classification result: " + class\_result)  
 print()  
  
*# Returns the number of ones and zeros in the data sample respectively.*def count\_0s\_and\_1s(data):  
 zeros\_count = 0  
 ones\_count = 0  
  
 for i in range(data.shape[0]):  
 if (data[i] == 1):  
 ones\_count += 1  
 elif (data[i] == 0):  
 zeros\_count += 1  
  
 return ones\_count, zeros\_count  
  
print("Results on dataset defined by homework.")  
  
*# Training dataset defined by homework.*data = np.array([[1, 1, 1], [0, 1, 0], [1, 1, 0], [0, 0, 0], [1, 0, 1], [1, 0, 0]])  
  
*# For this training, we call 0's labels class 1 and 1's labels class 2.  
#labels = np.array([1, 1, 1, 0, 0, 0])*labels = np.array([0, 0, 0, 1, 1, 1])  
  
*# Testing dataset defined by homework (with some extra for example).*testing = np.array([[0, 1, 1], [0, 0, 0], [1, 1, 1]])  
  
*# Train and test for the training and testing datasets defined by homework.*train\_and\_test(data, labels, testing)  
  
print("Results on extra dataset to show performance on variable datasets.")  
  
data\_2 = np.array([[1, 1, 1, 1, 1], [0, 1, 0, 0, 0], [1, 1, 0, 1, 1], [1, 1, 0, 1, 0], [0, 0, 0, 0, 0], [0, 0, 0, 1, 1]])  
labels\_2 = np.array([0, 0, 0, 0, 1, 1])  
testing\_2 = np.array([[1, 1, 1, 0, 0], [0, 0, 0, 1, 0], [0, 0, 0, 0, 0]])  
  
train\_and\_test(data\_2, labels\_2, testing\_2)

1. The code can be ran to get the training results and testing results on the dataset defined in the homework (plus some extra training examples). Also, the code outputs the training and testing results on the optional practice dataset defined in homework (to show variable features and samples).
   1. Here is the training and testing results for the homework dataset and and extra different dataset.

Results on dataset defined by homework.

Training Results:

P(C1): 0.5, P(C2): 0.5

P(1|C1): 0.6666666666666666, P(0|C1): 0.3333333333333333

P(1|C2): 0.3333333333333333, P(0|C2): 0.6666666666666666

Testing Results:

Testing results for sample: [0 1 1]

P(x|C1): 0.14814814814814814, P(x|C2): 0.07407407407407407

P(C1|x): 0.6666666666666666, P(C2|x): 0.3333333333333333

This sample classification result: Class 1.

Testing results for sample: [0 0 0]

P(x|C1): 0.037037037037037035, P(x|C2): 0.2962962962962963

P(C1|x): 0.1111111111111111, P(C2|x): 0.8888888888888888

This sample classification result: Class 2.

Testing results for sample: [1 1 1]

P(x|C1): 0.2962962962962963, P(x|C2): 0.037037037037037035

P(C1|x): 0.8888888888888888, P(C2|x): 0.1111111111111111

This sample classification result: Class 1.

Results on extra dataset to show performance on variable datasets.

Training Results:

P(C1): 0.6666666666666666, P(C2): 0.3333333333333333

P(1|C1): 0.65, P(0|C1): 0.35

P(1|C2): 0.2, P(0|C2): 0.8

Testing Results:

Testing results for sample: [1 1 1 0 0]

P(x|C1): 0.033641562500000007, P(x|C2): 0.005120000000000002

P(C1|x): 0.9292848202339332, P(C2|x): 0.07071517976606674

This sample classification result: Class 1.

Testing results for sample: [0 0 0 1 0]

P(x|C1): 0.009754062499999997, P(x|C2): 0.08192000000000003

P(C1|x): 0.1923344733031394, P(C2|x): 0.8076655266968605

This sample classification result: Class 2.

Testing results for sample: [0 0 0 0 0]

P(x|C1): 0.005252187499999998, P(x|C2): 0.32768000000000014

P(C1|x): 0.0310610890878681, P(C2|x): 0.9689389109121319

This sample classification result: Class 2.

1. It was a good experience to implement a simple naïve bayes algorithm from scratch. This allowed me to understand the algorithm more. Specifically, this helped me understand the capabilities and faults of the naïve bayes algorithm. The most prominent fault being that the features are considered independent, which can provide interesting results for data that is an equivalent combination to another data sample.