Assignment 2 Report

Xiaopei Zhang A20302816

I. Problem Statement

This assignment requires to implement various techniques in generative learning from scratch, evaluate the performance of different algorithms, and figure out how to predict class which each given test case belongs to. The first three sections focus on Gaussian discriminant analysis. It includes 1 dimensional and n dimensional 2-class datasets and n dimensional k-class dataset. Python linear discriminant analysis is used to compare. Confusion matrix, precision, recall, F-measure and accuracy are all implemented for performance evaluation. The last two sections are mainly about Naive Bayes. Naive Bayes with Bernoulli and with Binomial features are implemented. The same evaluations are performed, and Python BernoulliNB and MultinomialNB are used for comparison.

II. Proposed Solution

In the first three sections, I use the following steps to solve the 1 dimensional and n dimensional 2-class datasets and n dimensional k-class dataset problems.

1. Select distribution model: Gaussian.

2. Define model parameters α, μ,$\sum$.

$$m_j = \sum_{i=1}^{m} I(y^{(i)} = j) \qquad \alpha_j = \frac{m_j}{m} \qquad \mu_j = \frac{\sum_{i=1}^{m} I(y^{(i)} = j) X^{(i)}}{m_j}$$

$$\sum_j = \frac{\sum_{i=1}^{m} I(y^{(i)} = j)(X^{(i)} - \mu_j)(X^{(i)} - \mu_j)^T}{m_j}$$

3. Compute membership function.

$$g_j(x) = \log(\frac{1}{\sqrt{2\pi}}) - \log(\sigma_j) - \frac{(x-\mu_j)^2}{2\sigma_j^{\ 2}} + \log(\alpha_j)$$

4. Classify by comparing g(x). Class j that yields the largest g(x) is the class x belongs to. If $g_1(x) > g_2(x)$, x belongs to class 1; otherwise, x belongs to class 2.

In the fourth section, I use the following parameter α (the frequency feature i appears in all the examples of class j over the number of examples of class j) and membership function.

$$\alpha_{i|y=j} = \frac{\sum_{i=1}^{m} I(y^{(i)} = j) x_i^{\ (j)}}{\sum_{i=1}^{m} I(y^{(i)} = j)} \qquad g_j(x) = \prod_{i=1}^{n} \alpha_{i|y=j}^{\ x_i} (1 - \alpha_{i|y=j})^{1-x_i}$$

In the last section, I use the below parameter α (the frequency feature j appears in all the examples of class l over the frequency all the features appears in all the examples of class l) and membership function with Laplace smoothing.

$$\alpha_{j|y=l} = \frac{\sum_{i=1}^{m} I(y^{(i)} = l) x_j^{\ (i)} + E}{\sum_{i=1}^{m} I(y^{(i)} = l) P^{(i)} + kE} \qquad g_l(x) = \prod_{j=1}^{n} \binom{P}{x_j} \alpha_{j|y=l}^{\ x_j} (1 - \alpha_{j|y=l})^{P-x_j} \cdot \alpha_l$$

Written up steps for maximum likelihood:

$$l_l(\theta) = \log(P\{x^{(i)}\} \mid \theta) = \log \prod_{i=1}^{m} P(x^{(i)} \mid \theta)$$

$$= \log \prod_{i=1}^{m} \prod_{j=1}^{n} \binom{P^{(i)}}{x_j^{(i)}} (\alpha_{j|y=l})^{x_j^{(i)}} (1 - \alpha_{j|y=l})^{P^{(i)} - x_j^{(i)}}$$

$$= \sum_{i=1}^{m} \sum_{j=1}^{n} (\log \binom{P^{(i)}}{x_j^{(i)}} + x_j^{(i)} \log(\alpha_{j|y=l}) + (P^{(i)} - x_j^{(i)}) \log(1 - \alpha_{j|y=l})$$

$$\frac{\partial l_l(\theta)}{\partial \alpha_{p|y=l}} = 0 \qquad \sum_{i=1}^{m} x_p^{(i)} \cdot \frac{1}{\alpha_{p|y=l}} - \sum_{i=1}^{m} (P^{(i)} - x_p^{(i)}) \cdot \frac{1}{1 - \alpha_{p|y=l}} = 0$$

$$\alpha_{p|y=l} = \frac{\sum_{i=1}^{m} x_p^{(i)}}{\sum_{i=1}^{m} P^{(i)}}$$

III. Implementation

1. Program Design

My implementations of all the functions are highly modular. According to the sequence of my implementation, I explained most related functions below.

getParameters: take an ndarray with the last column as y, and return the sorted labels, prior for each label, mean for each label, and standard deviation for each label

predictLabel: take a test case, and parameters calculated before, use Gaussian membership function, and return the label that yields the highest g(x)

gda: take a test case, and parameters, and return g(x)

confusionMatrix: take a predicted y and a true y, and add 1 to the corresponding position in confusion matrix

getPR: compute precision and recall from a confusion matrix for a given label

getF: compute F-measure with given precision and recall value

getAccuracy: compute accuracy with a given confusion matrix

plotCurve: use sklearn packages to add noise and plot precision-recall curve

NB_Bernoulli: take training data, and return alpha for each feature in each label

NB_Bernoulli_predict: calculate g(x) for a given test case x and alpha for each feature in each label

NB_Binomial: take training data, and return alpha and prior for each feature in each label

NB_Binomial_predict: calculate g(x) for a given test case x and alpha and prior for each feature in each label

In the main function, “#” printout separates the five sections. In each section, I created global

confusion matrices for results from my own functions and results from Python made functions. Under 10 fold cross validation, I perform Gaussian discriminant analysis on perfume_data_reorganized.data (1D 2-class), iris_reorganized_2class.data (nD 2-class) and iris_reorganized.data (nD k-class), Naive Bayes Bernoulli and Naive Bayes Binomial on spambase.data (nD 2-class). Correspondingly, I perform Python made functions Linear discriminant analysis, Bernoulli Naive Bayes and Multinomial Naive Bayes for comparison of confusion matrices and evaluation.

2. Problems Encountered

Python numpy takes a list object as a 1 by n matrix, so instead of doing XX^T, I actually need to do X^TX during implementation.

3. Instructions

My script is written and tested under Python 2.7. My sklearn version is 0.16.0. I commented the line 338 that shows the plotted picture of precision-recall curve to make the program continually executable. If preferred, uncommenting this line would produce pictures, and closing picture windows would continue program execution.

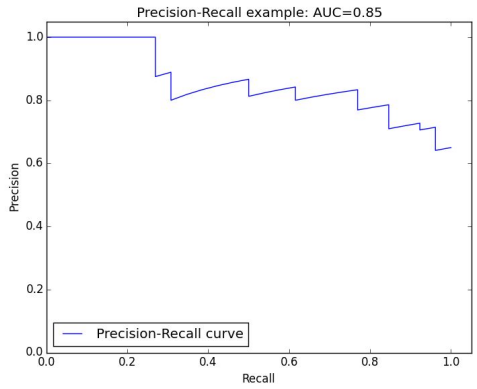
IV. Results and Discussion

1. Results

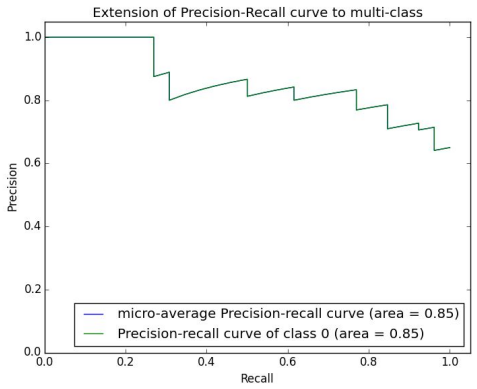
1D 2-class GDA

```
confusion matrix:          confusion matrix python:
[[ 75.   0.]               [[ 75.   0.]
 [  0.  37.]]               [  0.  37.]]
for class 1.0
precision:1.0      recall:1.0      F-measure:1.0
for class 2.0
precision:1.0      recall:1.0      F-measure:1.0
accuracy:1.0
```

nD 2-class GDA



Precision-recall curve



Precision-recall curve with area 0.85

confusion matrix:
[[83. 0.]
 [0. 17.]]

confusion matrix python:
[[83. 0.]
 [0. 17.]]

for class 1.0

precision:1.0 recall:1.0 F-measure:1.0

for class 2.0

precision:1.0 recall:1.0 F-measure:1.0

accuracy:1.0

nD k-class GDA

confusion matrix:
[[106. 0. 0.]
 [0. 34. 2.]
 [0. 5. 3.]]

confusion matrix python:
[[106. 0. 0.]
 [0. 37. 1.]
 [0. 2. 4.]]

for class 1.0

precision:1.0 recall:1.0 F-measure:1.0

for class 2.0

precision:0.944444444444 recall:0.871794871795 F-measure:0.906666666667

for class 3.0

precision:0.375 recall:0.6 F-measure:0.461538461538

accuracy:0.953333333333

NB with Bernoulli nD 2-class

confusion matrix:
[[1017. 554.]
 [47. 2983.]]

confusion matrix python:
[[1001. 457.]
 [63. 3080.]]

for class 0.0

precision: 0.647358370465 recall: 0.955827067669 F-measure: 0.771916508539

for class 1.0

precision: 0.984488448845 recall: 0.843370087645 F-measure: 0.908481802954

accuracy: 0.86937622256

NB with Binomial features nD 2-class

confusion matrix: confusion matrix python:
[[654. 103.] [[804. 79.]
[398. 3446.]] [248. 3470.]]
for class 0.0
precision:0.86393659181 recall:0.621673003802 F-measure:0.723051409619
for class 1.0
precision:0.89646201873 recall:0.970977740209 F-measure:0.932233193561
accuracy:0.891110628124

2. Demonstration of Correctness

My implementations of Gaussian discriminant analysis, Naive Bayes Bernoulli, and Naive Bayes Binomial are correct since my confusion matrices are similar with confusion matrices derived from Python made functions. Also, my implementations are based on the given functions in section II, so they are theoretically correct. In addition, class prediction comes from the highest values of membership functions, which means if the probability a case belongs to a class is high, that class would be selected.

3. Performance Evaluation

(1) Strength. All of my implementations are theoretically derived, and have high accuracy as presented in the result section.

(2) Weakness. My implementation of Naive Bayes Bernoulli and Binomial have a slightly lower accuracy than Python made functions.

4. Effect of Various Parameters on Results

My Gaussian discriminant analysis tends to perform better on 2-class datasets than k-class datasets. Also, my Naive Bayes Bernoulli has the tendency to classify more test cases to class 0.0 while my Naive Bayes Binomial has the tendency to classify more test cases to class 1.0.

References

[1]http://archive.ics.uci.edu/ml/datasets/Perfume+Data
[2]http://archive.ics.uci.edu/ml/datasets/Iris
[3]https://archive.ics.uci.edu/ml/datasets/Spambase
[4]http://scikit-learn.org/0.16/modules/generated/sklearn.lda.LDA.html
[5]http://scikit-learn.org/stable/auto_examples/model_selection/plot_roc.html