cs584 Assignment 2: Report

Xin Su
Department of Computer Science
Illinois Institute of Technology

March 21, 2016

Abstract

This report states the analysis of the Assignment 2's results. It resolves 5 problems one by one. First I implement a Gaussian Discriminant Analysis models to fit the datasets of 1D 2-class, nD 2-class and nD k-class. Second I implement an Naive Bayes model and fit it to the datasets with Bernoulli features and Binomial features. I evaluate the results for each model using confusion matrix, precision, recall, F-measure and accuracy.

1. Problem statement

I explored the following problems:

(1) Select a 2-class dataset with continuous 1D features. Estimate the model parameters and compute a discriminant function based on the distribution in each class. Classify the examples and measure error. Compute the confusion matrix, precision, recall, F-measure, and accuracy.

(2) Select a 2-class dataset with continuous nD features. Estimate the model parameters and compute a discriminant function based on the distribution in each class. Classify the examples and measure error. Compute the confusion matrix, precision, recall, F-measure, and accuracy.

(3) Select a k-class dataset with continuous nD features. Estimate the model parameters and compute a discriminant function based on the distribution in each class. Classify the examples and measure error. Compute the confusion matrix, precision, recall, F-measure, and accuracy.

(4) Select a 2-class dataset with binary nD features. Estimate the model parameters and compute a discriminant function based on the distribution in each class by using the Naive Bayes assumption. Classify the examples and measure error. Compute the confusion matrix, precision, recall, F-measure, and accuracy.

(5) Using maximum likelihood derive the parameter estimate equations for Naive Bayes with Binomial features: write the log likelihood function, compute its derivative, equate the derivative to zero, and solve for the parameters. Select a 2-class dataset with discrete nD features. This dataset needs to be derived from text documents. Estimate the model parameters and compute a discriminant function based on the distribution in each class by using the Naive Bayes assumption. Classify the examples and measure error. Compute the confusion matrix, precision, recall, F-measure, and accuracy.

2. Proposed solution

- Gaussian Discriminant Analysis
    - Parameters

$$\mu_j = \frac{1}{m_j} \sum_{i=1}^{m} I\left(y^{(i)} = j\right) x^{(i)}$$

$$\sigma_j^2 = \frac{1}{m_j} \sum_{i=1}^{m} I\left(y^{(i)} = j\right)\left(x^{(i)} - \mu_j\right)^2$$

    - Membership function

$$g_i(x) = \log \frac{1}{\sqrt{2\pi}} - \log\sigma_i - \frac{\left(x - \mu_i\right)^2}{2\sigma^2} + \log\alpha_i$$

    - Discriminant function

$$d(x) = g_2(x) - g_1(x)$$

- Multivariate Gaussian Discriminant Analysis
    - Covariance Matrix

$$\Sigma = \frac{1}{m} \sum_{i=1}^{m} \left(x^{(i)} - \mu\right)\left(x^{(i)} - \mu\right)^T$$

    - Membership function

$$g_i(x) = -\frac{1}{2}\log|\Sigma_j| - \frac{1}{2}(x - \mu_i)^T \Sigma^{-1}(x - \mu_i) + \log(\alpha_i)$$

    - Prior

$$\alpha_j = \frac{\sum_{i=1}^{m} I\left(y^{(i)} = j\right)}{m}$$

- Evaluation
    - Confusion Matrix

tp: true positive

tn: true negative
fp: false positive
fn: false negative

- precision

precision = 1. * tp / (tp + fp)

- recall

recall = 1. * tp / (tp + fn)

- F-measure

F = 2. * recall * precision / (recall + precision)

- accuracy

accuracy = 1. * (tp + tn) / (tp + tn + fp + fn)

- Multi-class Evaluation
    - Confusion Matrix:

		Classifier		
		C1	**…**	**Ck**
Classifier	**C1**	C11	…	C1k
	…	…	…	…
	Ck	Ck1	…	Ckk

- precision

$$precision_i = \frac{C_{ii}}{\Sigma C_{ji}}$$

- recall

$$recall_i = \frac{C_{ii}}{\Sigma C_{ij}}$$

- F-measure

F = 2. * recall * precision / (recall + precision)

- accuracy

$$accuracy = (\Sigma C_{ii})(\Sigma C_{ij})$$

- Naive Bayes
  - Naive Bayes assumption

$$P(x_1 \ldots x_n) = \prod_{i-1}^{n} P(x_i)$$

  - Bernoulli case

$$P(x \backslash y = 1) = \alpha_{i\backslash y=1}^{x_i} \left(1 - \alpha_{i\backslash y=1}^{1-x_i}\right)$$

  - Bernoulli Membership function

$$g_{j(x)} = \prod_{i=1}^{n} \alpha_{i\backslash y=j}^{x_i} (1 - \alpha_{i\backslash y=j})^{1-x_j}$$

  - Prior

$$\alpha_{j\backslash y=l} = \frac{\sum_{i=1}^{m} I\left(y^{(i)} = l\right) x_j^{(i)} + \varepsilon}{\sum_{i=1}^{m} I\left(y^{(i)} = l\right) p^{(i)} + k\varepsilon}$$

  - Binomial case

$$P\left(x_j^{(i)} \backslash y = l\right) = \begin{pmatrix} p^{(i)} \\ x_j^{(i)} \end{pmatrix} \alpha_{j\backslash y=l}^{x_j^{(i)}} \left(1 - \alpha_{j\backslash y=l}^{p^{(i)} - x_j^{(i)}}\right)$$

  - Binomial Membership function

$$g_l(x) = \sum_{j=1}^{n} \log\left(\begin{pmatrix} p \\ x_j \end{pmatrix} \alpha_{j\backslash y=l}^{x_j} (1 - \alpha_{i\backslash y=j})^{p-x_j}\right) + \log \alpha_l$$

3. Implementation details

I use Python 2.7 as development language, and use IPython Notebook to write the code and run. It is very convenient for use. A user can view it simply using a browser, and the content will contain all output, including text, data and graphs.

When you want to run the code, you should first install the IPython Notebook (Please refer to the Jupyter official website: http://jupyter.readthedocs.org/en/latest/install.html). After this step, you can open the .ipynb file and run. Just run from the top cell to the bottom. This top-down order is required.

4. Results and discussion

Problem 1

I use Skin_NonSkin.txt dataset but only choose the first feature. The result shows that the cross validation validates the model, and because I only choose one feature and cut out all others features, both the accuracy and precision are exactly 1/3, which means that but it shows that the recall is much higher than the precision which means the model may be under fit the dataset.

Problem 1 Result

	cross validation training avg	**cross validation test avg**	**all data**
Confusion	[[[18964, 95738], [21743, 59604]], [[18998, 95679], [21749, 59623]], [[18880, 95799], [21773, 59598]], [[18540, 95615], [22123, 59772]], [[18937, 95791], [21739, 59583]]]	[[[4723, 23989], [5433, 14871]], [[4689, 24048], [5427, 14852]], [[4807, 23928], [5403, 14877]], [[4611, 23760], [5589, 15055]], [[4750, 23936], [5437, 14892]]]	[[23686, 119726], [27175, 74474]]
precision	0.164620451827	0.164612701703	0.165160516554
recall	0.463605387899	0.463544146155	0.465700635064
F-measure	0.242966389749	0.242947414517	0.243842427924
accuracy	0.400407856959	0.400392539375	0.400553331619

Problem 2

I still use Skin_NonSkin.txt dataset but use all features. The result shows that the cross validation validates the model, and because I choose all features this time, both the accuracy and precision are high, and it shows that the recall is also high which means the model fit the dataset very well.

Problem 2 Result

	cross validation training avg	**cross validation test avg**	**all data**
Confusion	[[[40621, 11705], [8, 143715]], [[40784, 11767], [7, 143491]], [[40647, 11897], [6, 143500]], [[40808, 11783], [6, 143453]], [[40554, 11790], [5, 143701]]]	[[[10232, 2974], [2, 35808]], [[10071, 2948], [1, 35996]], [[10209, 2951], [1, 35854]], [[10045, 2904], [4, 36062]], [[10302, 2963], [2, 35748]]]	[[50854, 14727], [7, 179473]]
precision	0.775336039755	0.775297347502	0.77543800796
recall	0.999842722733	0.999803039143	0.999862369989
F-measure	0.873392324703	0.873352607917	0.873464900981
accuracy	0.939837674265	0.93981483578	0.93987619409

Problem 3

I use iris.data dataset and use all features. The result shows that the cross validation validates the model, and it also shows that both the accuracy and precision are exactly 1/3 which means it can be assumed equal prior. And it shows that the recall of class 1 is very high but for class 2 and 3 the recall is too low. Also combining the confusion matrix and F-measure we can see that it seems that the class 1 is more easy to predict.

Problem 3 Result

	cross validation training avg	cross validation test avg	all data
Confusion	[array([[46., 44., 48.], [1., 1., 1.], [1., 1., 1.]]), array([[46., 48., 44.], [1., 1., 1.], [1., 1., 1.]]), array([[44., 45., 49.], [1., 1., 1.], [1., 1., 1.]]), array([[48., 42., 48.], [1., 1., 1.], [1., 1., 1.]]), array([[44., 49., 45.], [1., 1., 1.], [1., 1., 1.]]), array([[47., 45., 46.], [1., 1., 1.], [1., 1., 1.]]), array([[49., 44., 45.], [1., 1., 1.], [1., 1., 1.]]), array([[46., 48., 44.], [1., 1., 1.], [1., 1., 1.]]), array([[46., 47., 45.], [1., 1., 1.], [1., 1., 1.]]), array([[44., 48., 46.], [1., 1., 1.], [1., 1., 1.]])]	[array([[6., 8., 4.], [1., 1., 1.], [1., 1., 1.]]), array([[6., 4., 8.], [1., 1., 1.], [1., 1., 1.]]), array([[8., 7., 3.], [1., 1., 1.], [1., 1., 1.]]), array([[4., 10., 4.], [1., 1., 1.], [1., 1., 1.]]), array([[8., 3., 7.], [1., 1., 1.], [1., 1., 1.]]), array([[5., 7., 6.], [1., 1., 1.], [1., 1., 1.]]), array([[3., 8., 7.], [1., 1., 1.], [1., 1., 1.]]), array([[6., 4., 8.], [1., 1., 1.], [1., 1., 1.]]), array([[6., 5., 7.], [1., 1., 1.], [1., 1., 1.]]), array([[8., 4., 6.], [1., 1., 1.], [1., 1., 1.]])]	[[51. 51. 51.] [1. 1. 1.] [1. 1. 1.]]

	cross validation training avg	cross validation test avg	all data
precision	[array([0.33333333, 0.33333333, 0.33333333]), array([0.33333333, 0.33333333, 0.33333333]), array([0.31884058, 0.33333333, 0.33333333]), array([0.34782609, 0.33333333, 0.33333333]), array([0.31884058, 0.33333333, 0.33333333]), array([0.34057971, 0.33333333, 0.33333333]), array([0.35507246, 0.33333333, 0.33333333]), array([0.33333333, 0.33333333, 0.33333333]), array([0.33333333, 0.33333333, 0.33333333]), array([0.31884058, 0.33333333, 0.33333333])]	[array([0.33333333, 0.33333333, 0.33333333]), array([0.33333333, 0.33333333, 0.33333333]), array([0.44444444, 0.33333333, 0.33333333]), array([0.22222222, 0.33333333, 0.33333333]), array([0.44444444, 0.33333333, 0.33333333]), array([0.27777778, 0.33333333, 0.33333333]), array([0.16666667, 0.33333333, 0.33333333]), array([0.33333333, 0.33333333, 0.33333333]), array([0.33333333, 0.33333333, 0.33333333]), array([0.44444444, 0.33333333, 0.33333333])]	[0.33333333 0.33333333 0.33333333]
recall	[array([0.95833333, 0.02173913, 0.02]), array([0.95833333, 0.02 , 0.02173913]), array([0.95652174, 0.0212766 , 0.01960784]), array([0.96 , 0.02272727, 0.02]), array([0.95652174, 0.01960784, 0.0212766]), array([0.95918367, 0.0212766 , 0.02083333]), array([0.96078431, 0.02173913, 0.0212766]), array([0.95833333, 0.02 , 0.02173913]), array([0.95833333, 0.02040816, 0.0212766]), array([0.95652174, 0.02 , 0.02083333])]	[array([0.75 , 0.1 , 0.16666667]), array([0.75 , 0.16666667, 0.1]), array([0.8 , 0.11111111, 0.2]), array([0.66666667, 0.08333333, 0.16666667]), array([0.8 , 0.2 , 0.11111111]), array([0.71428571, 0.11111111, 0.125]), array([0.6 , 0.1 , 0.11111111]), array([0.75 , 0.16666667, 0.1]), array([0.75 , 0.14285714, 0.11111111]), array([0.8 , 0.16666667, 0.125])]	[0.96226415 0.01886792 0.01886792]

	cross validation training avg	cross validation test avg	all data
F-measure	[array([0.49462366, 0.04081633, 0.03773585]), array([0.49462366, 0.03773585, 0.04081633]), array([0.47826087, 0.04 , 0.03703704]), array([0.5106383 , 0.04255319, 0.03773585]), array([0.47826087, 0.03703704, 0.04]), array([0.5026738 , 0.04 , 0.03921569]), array([0.51851852, 0.04081633, 0.04]), array([0.49462366, 0.03773585, 0.04081633]), array([0.49462366, 0.03846154, 0.04]), array([0.47826087, 0.03773585, 0.03921569])]	[array([0.46153846, 0.15384615, 0.22222222]), array([0.46153846, 0.22222222, 0.15384615]), array([0.57142857, 0.16666667, 0.25]), array([0.33333333, 0.13333333, 0.22222222]), array([0.57142857, 0.25 , 0.16666667]), array([0.4 , 0.16666667, 0.18181818]), array([0.26086957, 0.15384615, 0.16666667]), array([0.46153846, 0.22222222, 0.15384615]), array([0.46153846, 0.2 , 0.16666667]), array([0.57142857, 0.22222222, 0.18181818])]	[0.49514563 0.03571429 0.03571429]
accuracy	0.333333333333	0.333333333333	0.333333333333

Problem 4
I use SPECT.data dataset and use all features. The results show that both the accuracy and precision are very low, but the recall is very high which means that this model does not fit the dataset quite well. The reason for this may be the amount of the data in the dataset is too small. But the high recall shows that this model will return relevant data from the dataset.

Problem 4 Result

	all data
Confusion	[[16, 131], [1, 43]]
precision	0.108843537415
recall	0.941176470588
F-measure	0.19512195122
accuracy	0.30890052356

Problem 5
I use lenses.data dataset and use all features. The results show that both the accuracy and precision are very low, and the recall is for class 1 is very high but relative low to class 2 and 3. This means that this model does not fit the dataset quite well. The reason for this may be the amount of the data in the dataset is too small and it cannot train the model as wished. The precision is pretty much showing these class are close to equal prior but not perfectly.

Problem 5 Result

	cross validation training avg	**cross validation test avg**	**all data**
Confusion	[array([[10., 13., 1.], [1., 1., 1.], [1., 1., 1.]]), array([[11., 12., 1.], [1., 1., 1.], [1., 1., 1.]]), array([[11., 12., 1.], [1., 1., 1.], [1., 1., 1.]]), array([[13., 10., 1.], [1., 1., 1.], [1., 1., 1.]]), array([[12., 12., 1.], [1., 1., 1.], [1., 1., 1.]]), array([[12., 12., 1.], [1., 1., 1.], [1., 1., 1.]]), array([[12., 12., 1.], [1., 1., 1.], [1., 1., 1.]]), array([[13., 11., 1.], [1., 1., 1.], [1., 1., 1.]]), array([[12., 12., 1.], [1., 1., 1.], [1., 1., 1.]]), array([[12., 12., 1.], [1., 1., 1.], [1., 1., 1.]])]	[array([[4., 1., 1.], [1., 1., 1.], [1., 1., 1.]]), array([[3., 2., 1.], [1., 1., 1.], [1., 1., 1.]]), array([[3., 2., 1.], [1., 1., 1.], [1., 1., 1.]]), array([[1., 4., 1.], [1., 1., 1.], [1., 1., 1.]]), array([[2., 2., 1.], [1., 1., 1.], [1., 1., 1.]]), array([[2., 2., 1.], [1., 1., 1.], [1., 1., 1.]]), array([[2., 2., 1.], [1., 1., 1.], [1., 1., 1.]]), array([[1., 3., 1.], [1., 1., 1.], [1., 1., 1.]]), array([[2., 2., 1.], [1., 1., 1.], [1., 1., 1.]]), array([[2., 2., 1.], [1., 1., 1.], [1., 1., 1.]])]	[[13. 13. 1.] [1. 1. 1.] [1. 1. 1.]]

	cross validation training avg	cross validation test avg	all data
precision	[array([0.41666667, 0.33333333, 0.33333333]), array([0.45833333, 0.33333333, 0.33333333]), array([0.45833333, 0.33333333, 0.33333333]), array([0.54166667, 0.33333333, 0.33333333]), array([0.48 , 0.33333333, 0.33333333]), array([0.48 , 0.33333333, 0.33333333]), array([0.48 , 0.33333333, 0.33333333]), array([0.52 , 0.33333333, 0.33333333]), array([0.48 , 0.33333333, 0.33333333]), array([0.48 , 0.33333333, 0.33333333])]	[array([0.66666667, 0.33333333, 0.33333333]), array([0.5 , 0.33333333, 0.33333333]), array([0.5 , 0.33333333, 0.33333333]), array([0.16666667, 0.33333333, 0.33333333]), array([0.4 , 0.33333333, 0.33333333]), array([0.4 , 0.33333333, 0.33333333]), array([0.4 , 0.33333333, 0.33333333]), array([0.2 , 0.33333333, 0.33333333]), array([0.4 , 0.33333333, 0.33333333]), array([0.4 , 0.33333333, 0.33333333])]	[0.48148148 0.33333333 0.33333333]
recall	[array([0.83333333, 0.06666667, 0.33333333]), array([0.84615385, 0.07142857, 0.33333333]), array([0.84615385, 0.07142857, 0.33333333]), array([0.86666667, 0.08333333, 0.33333333]), array([0.85714286, 0.07142857, 0.33333333]), array([0.85714286, 0.07142857, 0.33333333]), array([0.85714286, 0.07142857, 0.33333333]), array([0.86666667, 0.07692308, 0.33333333]), array([0.85714286, 0.07142857, 0.33333333]), array([0.85714286, 0.07142857, 0.33333333])]	[array([0.66666667, 0.33333333, 0.33333333]), array([0.6 , 0.25 , 0.33333333]), array([0.6 , 0.25 , 0.33333333]), array([0.33333333, 0.16666667, 0.33333333]), array([0.5 , 0.25 , 0.33333333]), array([0.5 , 0.25 , 0.33333333]), array([0.5 , 0.25 , 0.33333333]), array([0.33333333, 0.2 , 0.33333333]), array([0.5 , 0.25 , 0.33333333]), array([0.5 , 0.25 , 0.33333333])]	[0.86666667 0.06666667 0.33333333]

	cross validation training avg	cross validation test avg	all data
F-measure	[array([0.55555556, 0.11111111, 0.33333333]), array([0.59459459, 0.11764706, 0.33333333]), array([0.59459459, 0.11764706, 0.33333333]), array([0.66666667, 0.13333333, 0.33333333]), array([0.61538462, 0.11764706, 0.33333333]), array([0.61538462, 0.11764706, 0.33333333]), array([0.61538462, 0.11764706, 0.33333333]), array([0.65 , 0.125 , 0.33333333]), array([0.61538462, 0.11764706, 0.33333333]), array([0.61538462, 0.11764706, 0.33333333])]	[array([0.66666667, 0.33333333, 0.33333333]), array([0.54545455, 0.28571429, 0.33333333]), array([0.54545455, 0.28571429, 0.33333333]), array([0.22222222, 0.22222222, 0.33333333]), array([0.44444444, 0.28571429, 0.33333333]), array([0.44444444, 0.28571429, 0.33333333]), array([0.44444444, 0.28571429, 0.33333333]), array([0.25 , 0.25 , 0.33333333]), array([0.44444444, 0.28571429, 0.33333333]), array([0.44444444, 0.28571429, 0.33333333])]	[0.61904762 0.11111111 0.33333333]
accuracy	0.450860215054	0.367424242424	0.454545454545

5. References

[1] https://www.coursera.org/learn/machine-learning
[2] http://www.numpy.org/
[3] https://www.python.org/