Yang_Jinxin_1168646_homework-3

BY JINXIN YANG

brianyang1106@gmail.com

November 4, 2012

1 Question 1

1.1 Data preprocessing

Since *glass.data* is not a standard format file for WEKA, so I wrote a small program to do the preprocessing which will pick up 6 attributes with class, generate its .arff file and the entire attributes .arff file. The source file is *src/h3/src/h3/q1.java* and two .arff file is *Question1/toWEKA/glass.arff* and *all.arff*.

1.2 6 attributes

After briefly analyzing the *glass.data* and *glass.names*, I used 2nd to 7th attributes as my dataset:

- RI: refractive index
- Na: Sodium (unit measurement: weight percent in corresponding oxide, as are attributes 4-10)
- Mg: Magnesium
- Al: Aluminum
- Si: Silicon
- K: Potassium

1.3 10 algorithms and meta-learner

- Support Vector Machine - SMO - Polykernel 1.0
- Support Vector Machine - SMO - Polykernel 2.0
- Decision trees - J48 - confidenceFactor 0.25
- Decision trees - J48 - confidenceFactor 0.10
- Decision trees - Random Forest - maxDepth unlimited
- Decision trees - Random Forest - maxDepth 2
- Naïve Bayes - useSupervisedDiscretization False (**meta-learner**)
- Naïve Bayes - useSupervisedDiscretization True
- Neural networks - MultilayerPerceptron - learningRate 0.3
- Neural networks - MultilayerPerceptron - learningRate 0.6

1.4 Confusion matrix

```
=== Confusion Matrix ===

  a  b  c  d  e  f  g   <-- classified as
 49  9 12  0  0  0  0 |  a = building_windows_float_processed
 14 22 32  0  4  0  4 |  b = building_windows_non_float_processed
  7  3  7  0  0  0  0 |  c = vehicle_windows_float_processed
  0  0  0  0  0  0  0 |  d = vehicle_windows_non_float_processed
  0  4  0  0  7  0  2 |  e = containers
  0  2  0  0  0  5  2 |  f = tableware
  1  4  0  0  2  0 22 |  g = headlamps
```

Figure 1. Question-a-6-attributes

```
=== Confusion Matrix ===

  a  b  c  d  e  f  g   <-- classified as
 25  1 44  0  0  0  0 |  a = building_windows_float_processed
  7 19 45  0  4  1  0 |  b = building_windows_non_float_processed
  5  0 12  0  0  0  0 |  c = vehicle_windows_float_processed
  0  0  0  0  0  0  0 |  d = vehicle_windows_non_float_processed
  0  3  0  0  9  0  1 |  e = containers
  0  1  0  0  0  8  0 |  f = tableware
  0  2  1  0  2  0 24 |  g = headlamps
```

Figure 2. Question-b-all-attributes

1.5 Cross validation information

First, take 6-attributes' (Figure 1) confusion matrix as an example, we calculate TP, FP, TN and FN like this:

```
=== Confusion Matrix ===

  a  b  c  d  e  f  g   <-- classified as
 49  9 12  0  0  0  0 |  a = building_windows_float_processed
 14 22 32  0  4  0  4 |  b = building_windows_non_float_processed
  7  3  7  0  0  0  0 |  c = vehicle_windows_float_processed
  0  0  0  0  0  0  0 |  d = vehicle_windows_non_float_processed
  0  4  0  0  7  0  2 |  e = containers
  0  2  0  0  0  5  2 |  f = tableware
  1  4  0  0  2  0 22 |  g = headlamps
```


Figure 3. definition

Then, we can calculate them, the results are:

	TP	FP	TN	FN
Question-a	22	8	177	7
Question-b	24	1	184	5

Table 1. TP, FP, TN and FN

Through the Table 1, we can find that the difference between two matrices which cannot give us accurate judgment about two results. So, we may includes other important parameters:

	TP Rate	FP Rate	Precision	Recall	F-Measure	MCC	ROC Area	PRC Area
Question-a	0.759	0.043	0.733	0.759	0.746	0.705	0.905	0.693
Question-b	0.828	0.005	0.96	0.828	0.889	0.876	0.938	0.886

Table 2. other parameters

We can easily find that Question-b has a better classification after comparing every parameter.

But whether we can assert that it's ubiquitous for every meta-learner is still unknown, so I tested other four original algorithm as meta-learner and got this result:

	TP Rate	FP Rate	Precision	Recall	F-Measure	MCC	ROC Area	PRC Area
Question-a	0.69	0.022	0.833	0.69	0.755	0.725	0.826	0.657
Question-b	0.828	0.032	0.8	0.828	0.814	0.784	0.904	0.718

Table 3. J48 as meta-learner

	TP Rate	FP Rate	Precision	Recall	F-Measure	MCC	ROC Area	PRC Area
Question-a	0.724	0.054	0.677	0.724	0.7	0.652	0.879	0.73
Question-b	0.828	0.011	0.923	0.828	0.873	0.856	0.942	0.897

Table 4. MultilayerPerceptron as meta-learner

	TP Rate	FP Rate	Precision	Recall	F-Measure	MCC	ROC Area	PRC Area
Question-a	0.552	0.022	0.8	0.552	0.653	0.623	0.875	0.604
Question-b	0.862	0.005	0.962	0.862	0.909	0.897	0.951	0.869

Table 5. SMO as meta-learner

	TP Rate	FP Rate	Precision	Recall	F-Measure	MCC	ROC Area	PRC Area
Question-a	0.759	0.016	0.88	0.759	0.815	0.791	0.962	0.836
Question-b	0.793	0.011	0.92	0.793	0.852	0.834	0.954	0.892

Table 6. RandomForest as meta-learner

All the outputs and saved models are in the *Question1/fromWEKA* directory.

1.6 Conclusion

From Table 2 to Table 6, we know that, even though there's slightly decrease in Precision of J48, we may still can conclude that the result of 9-attributes (all attributes) is better than 6-attributes'. Increasing the dimension of attributes may guarantee a better classification in stacking algorithm.

2 Question 2

2.1 Generate the data

My PSID is 1168646, which means XX=68, YY=64 and Z=6, so for two distributions:

	mean	standard deviation	covariance matrix
first	[68, 64]	9	$\begin{pmatrix} 9^2 & 0 \\ 0 & 9^2 \end{pmatrix}$
second	[86, 82]	9	$\begin{pmatrix} 9^2 & 0 \\ 0 & 9^2 \end{pmatrix}$

Table 7. two distribution parameters

Then we should use bi-variate Gaussian distribution to generate the two datasets:

$$P(x,y) = \frac{1}{2\pi\sigma_x\sigma_y\sqrt{1-\rho^2}} \exp\left\{-\frac{z}{2(1-\rho^2)}\right\}, \tag{1}$$

where

$$z \equiv \frac{(x-\mu_x)^2}{\sigma_x^2} - \frac{2\rho(x-\mu_x)(y-\mu_y)}{\sigma_x\sigma_y} + \frac{(y-\mu_y)^2}{\sigma_y^2}, \tag{2}$$

and

$$\rho \equiv \mathrm{cov}(x, y) \tag{3}$$

Here, we know $\rho = 0$, so we can generate the bi-variate Gaussian distribution through multiplying two uni-variate Gaussian distribution of (x, y) sparately.

The generated data file is *Question2/data/output.data*, and ploted as:

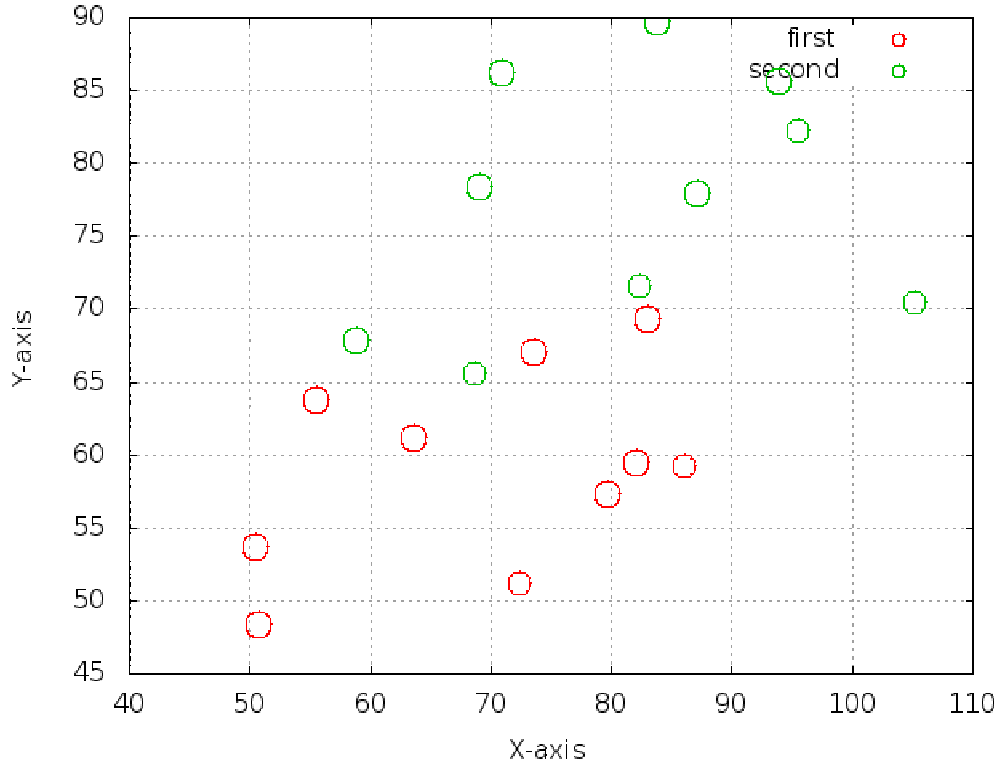

Figure 4. two distribution

2.2 Gaussian Mixture Model

The basic idea of the EM algorithm is that sometimes it is easier to add extra variables that are not actually known (called `hidden` or `latent` variables) and then to maximise the function over those variables. If we know how many classes there are in the data, then we can try to estimate the parameters for that many Gaussians, all at once. If we don't know, the we can try different numbers and see which one works best. It is perfectly possible (and reasonable) to use any other probability distribution instead of a Gaussian, but Gaussians are by far the most common choice. Then the output for any particular datapoint that is input to the algorithm will be the sum of the values expected by all of the M Gaussians [1]:

$$f(x) = \sum_{m=1}^{M} \alpha_m \, \phi(x; \mu_m, \Sigma_m), \tag{4}$$

where $\phi(x; \mu_m, \Sigma_m)$ is a Gaussian function with mean μ_m and covariance matrix Σ_m, and the α_m are weights with the constraint that $\sum_{m=1}^{M} \alpha_m = 1$. In this question, we suppose that we have two classes, so the equation we will use is:

$$P(y) = \pi\phi(y; \mu_1, \sigma_1) + (1-\pi)\phi(y; \mu_2, \sigma_2) \tag{5}$$

Also, for this question, we generate the two original dataset by using bi-variate Gaussian distribution which $\rho = 0$, so we will use:

$$P(x, y) = \frac{1}{2\pi\sigma_x\sigma_y} \exp\left\{-\frac{1}{2} \cdot \left(\frac{(x-\mu_x)^2}{\sigma_x^2} + \frac{(y-\mu_y)^2}{\sigma_y^2}\right)\right\}, \tag{6}$$

where σ_x, σ_y, μ_x and μ_y, which are vector in nature, are all two dimensional parameters.

2.3 Expectation Maximisation Algorithm

After getting all the datasets (20 points), we will regard them as the entire data and apply them EM algorithm to check whether we can get a good classification of two classes by testing different parameters. The EM algorithm we will use for this question are:

- Initialisation
  - set $\hat{\mu}_1$, $\hat{\mu}_2$ to be randomly chosen values from the dataset
  - set $\hat{\sigma}_1, \hat{\sigma}_1 = \sqrt{\sum_{i=1}^{N} \frac{(y_i - \bar{y})^2}{N}}$ (where $\bar{y}$ is the mean of the entire dataset)
  - set $\hat{\pi} = 0.5$
- repeat five iterations ($N = 20$ here)
  - (E-step) $\hat{\gamma}_i = \frac{\hat{\pi}\phi(y_i; \mu_2, \sigma_2)}{(1-\hat{\pi})\phi(y; \mu_1, \sigma_1) + \hat{\pi}\phi(y_i; \mu_2, \sigma_2)}, i = 1, 2, 3, ...N$
  - (M-step 1) $\hat{\mu}_1 = \frac{\sum_{i=1}^{N}(1-\hat{\gamma}_i)y_i}{\sum_{i=1}^{N}(1-\hat{\gamma}_i)}$
  - (M-step 2) $\hat{\mu}_2 = \frac{\sum_{i=1}^{N}\hat{\gamma}_i y_i}{\sum_{i=1}^{N}\hat{\gamma}_i}$
  - (M-step 3) $\hat{\sigma}_1 = \sqrt{\frac{\sum_{i=1}^{N}(1-\hat{\gamma}_i)(y_i - \hat{\mu}_1)^2}{\sum_{i=1}^{N}(1-\hat{\gamma}_i)}}$
  - (M-step 4) $\hat{\sigma}_1 = \sqrt{\frac{\sum_{i=1}^{N}\hat{\gamma}_i(y_i - \hat{\mu}_2)^2}{\sum_{i=1}^{N}\hat{\gamma}_i}}$
  - (M-step 5) $\hat{\pi} = \sum_{i=1}^{N}\frac{\hat{\gamma}_i}{N}$

We should notice that from M-step 1 to M-step 4, all the $\hat{\mu}$ and $\hat{\sigma}$ are vectors which have x and y dimension. So, we have to calculate the $[x, y]$ saparately and combine them together again. After we writing the program to implement the EM algorithm, we should find that $\hat{\mu}_1$, $\hat{\mu}_2$, $\hat{\sigma}_1$ and $\hat{\sigma}_2$ is getting close to the original value for generating the two distributions and converge to certain values (may not be exactly the same as the original one, which is still reasonable).

2.4 Results of my program

I implemented the algorithm with Java, all the outputs of initialisation and iterations are in the file *Question2/outputs/output*, there might be some formatting problems using Windows Notepad to open it, so I organize all the data in the following table:

	$\hat{\mu}_1$	$\hat{\mu}_2$	$\text{cov}(\hat{\sigma}_{x_1}, \hat{\sigma}_{y_1})$	$\text{cov}(\hat{\sigma}_{x_2}, \hat{\sigma}_{y_2})$	$\hat{\pi}$
Origin	(68.0000, 64.0000)	(86.0000, 82.0000)	$\begin{pmatrix} 9.0000^2 & 0 \\ 0 & 9.0000^2 \end{pmatrix}$	$\begin{pmatrix} 9.0000^2 & 0 \\ 0 & 9.0000^2 \end{pmatrix}$	-
Initial	(50.4463, 53.7589)	(70.9320, 86.2706)	$\begin{pmatrix} 14.6421^2 & 0 \\ 0 & 11.7349^2 \end{pmatrix}$	$\begin{pmatrix} 14.6421^2 & 0 \\ 0 & 11.7349^2 \end{pmatrix}$	0.5000
iteration-1	(65.8977, 58.8470)	(82.8133, 75.3579)	$\begin{pmatrix} 12.2474^2 & 0 \\ 0 & 6.7513^2 \end{pmatrix}$	$\begin{pmatrix} 11.8517^2 & 0 \\ 0 & 9.4836^2 \end{pmatrix}$	0.5760
iteration-2	(65.7424, 59.0426)	(83.4652, 75.7197)	$\begin{pmatrix} 12.0395^2 & 0 \\ 0 & 6.5734^2 \end{pmatrix}$	$\begin{pmatrix} 11.4285^2 & 0 \\ 0 & 9.4666^2 \end{pmatrix}$	0.5585
iteration-3	(65.8396, 59.1772)	(83.8279, 76.0249)	$\begin{pmatrix} 11.9782^2 & 0 \\ 0 & 6.5651^2 \end{pmatrix}$	$\begin{pmatrix} 11.2407^2 & 0 \\ 0 & 9.3566^2 \end{pmatrix}$	0.5449
iteration-4	(66.0340, 59.2819)	(84.0531, 76.3036)	$\begin{pmatrix} 11.9863^2 & 0 \\ 0 & 6.5691^2 \end{pmatrix}$	$\begin{pmatrix} 11.1684^2 & 0 \\ 0 & 9.2325^2 \end{pmatrix}$	0.5331
iteration-5	(66.2551, 59.3640)	(84.2112, 76.5688)	$\begin{pmatrix} 12.0218^2 & 0 \\ 0 & 6.5688^2 \end{pmatrix}$	$\begin{pmatrix} 11.1488^2 & 0 \\ 0 & 9.0978^2 \end{pmatrix}$	0.5227

Table 8. outputs of all the iterations

Question2/outputs/output file shows all the output of program, since `gamma[i]` has 20 values every iteration, I used screenshots for convinience:

```
1 iteration:
.4717  .2389  .1408  .1228  .0353
.6356  .8794  .0080  .0024  .3908
.9990  .3364  .9920  .9990  .9980
.9875  .9423  .4300  .9878  .9218

2 iteration:
.4702  .2373  .0867  .0588  .0553
.5685  .8807  .0053  .0032  .3712
1.0000  .2150  .9992  .9999  .9997
.9912  .9714  .3278  .9955  .9332

3 iteration:
.4533  .2144  .0631  .0375  .0463
.5281  .8774  .0030  .0019  .3467
1.0000  .1581  .9994  .9999  .9998
.9922  .9703  .2766  .9961  .9329

4 iteration:
.4237  .1899  .0510  .0287  .0374
.4928  .8670  .0021  .0013  .3173
1.0000  .1290  .9993  .9999  .9998
.9918  .9658  .2431  .9959  .9270

5 iteration:
.3874  .1659  .0435  .0241  .0302
.4623  .8542  .0016  .0009  .2857
1.0000  .1127  .9992  .9999  .9998
.9907  .9619  .2189  .9955  .9198
```

Figure 5. gamma values

All the program is in *src* folder, *h3* is a Java project file folder, you can just import this directory as "Existing Projects into Workspace" by Eclipse and run *src→h3→q2.java*.

If there is something wrong in Windows Eclipse (Linux works well), please make some lines to comment:

- line 204: `new q2().plot(); // display to window`
- line 205: `new q2().plotToPNG(); // generate a png picture under directory`

and run the h3.q2.java again. This just skips the plotting part and doesn't change anything else. I cannot help with Windows because it will take a long time for you to configure environment.

2.5 Analysis of results

From Sec. 2.4, we can find that only after first iteration, the means and the standard variance become kind of stable through five iterations, actually every iteration it has slight change, I test 100 iteration to find the convergence point, the result is:

```
56 iteration:
68.0051 60.0230 | 85.2568 78.8524  ||  12.3415 6.6295 | 11.2551 7.6261 | .4426
.0756  .0213  .0087  .0064  .0014
.2121  .6622  .0001  .0000  .0525
.9999  .0436  .9982  .9998  .9994
.9656  .9279  .0743  .9896  .8133
```

Figure 6. convergence point

From 56th iteration, all the values never change (I am sure about this) again. Even the final means and standard variances are still different from the original ones generated the data, we may still say the means that I obtain are actually moving close to the means using which the dataset has been generated. Because in this question, we only generated 10 examples and the points are really scattered, it is pretty good we can get these results actually. So, taking the number of examples and the number of iterations into consideration, if we increase both of them, we may get a much more accurate classification. But in terms of the algorithm itself, the idea behind it is very simple while the performance is really good.

Bibliography

[1] S. Marsland. *Machine Learning: An Algorithmic Perspective*. Chapman & Hall/CRC machine learning & pattern recognition series. Taylor & Francis, 2009.