Midterm 2

Xiangyang Han

April 17, 2019

1 Naive Bayes

1.1 1

The reason why naive Bayesian classifier is called naive is: all naive Bayes classifiers assume that the value of a particular feature is **independent** of the value of any other feature, given the class variable. In real world, the features of an object actually have connections with each other. However, this naive assumption can help the classifier to be trained very efficiently.

1.2 2

Code a naive Bayes classifier in python to classify the input :

$$X = (age = youth, income = low, student = yes, creditrating = fair)$$

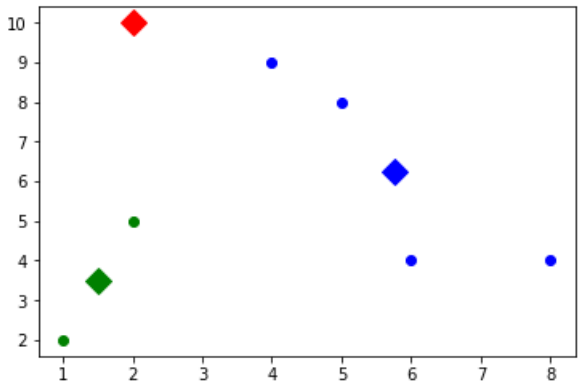
$P(yes|X) = P(youth|yes) * P(low|yes) * P(student_{yes}|yes) * P(fair|yes)$
$P(no|X) = P(youth|no) * P(low|no) * P(student_{yes}|no) * P(fair|no)$
The result is P(yes|X)=0.3292,P(no|X)=0.0096. So the class for X is yes.

2 k-means

2.1 1

Given data:A1(2,10),A2(2,5),A3(8,4),B1(5,8),B3(6,4),C1(1,2),C2(4,9).
Code a k-means cluster with python.
If break after the first loop, the centroids are:(2,10),(5.75,6.25),(1.5,3.5). The clusters is shown as well.

```
In [130]: runfile('F:/project-gwu/Machine Learning/mi
step 1:load data...
step 2: clustering...
Centroids are:
[[ 2. 10.]
 [ 5.  8.]
 [ 1.  2.]]
Congratulations, cluster complete!
step 3: show the result...
First round complete, centroids are:
[[ 2.    10.  ]
 [ 5.75  6.25]
 [ 1.5   3.5 ]]
```

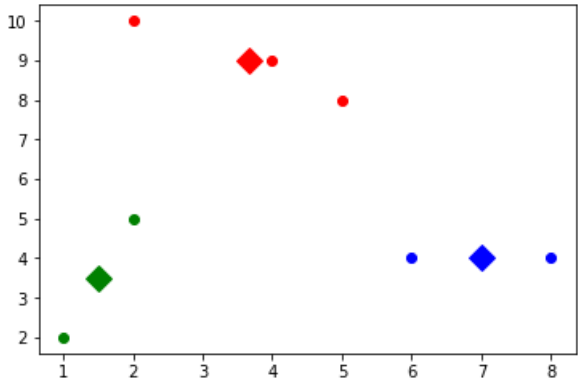


2.2 2

Running the code, the final result is Class 1:(A_1, B_1, C_2);Class 2:(A_3, B_3);Class 3:(A_2, C_1).

```
In [128]: runfile('F:/project-gwu/Machine Learnir
step 1:load data...
step 2: clustering...
Centroids are:
[[ 2. 10.]
 [ 5.  8.]
 [ 1.  2.]]
Congratulations, cluster complete!
step 3: show the result...
The final result is:centroid:
[[3.66666667 9.        ]
 [7.         4.        ]
 [1.5        3.5       ]]

assignment:
[[ 0.    0.  ]
 [ 2.   10.  ]
 [ 1.   25.  ]
 [ 0.    6.25]
 [ 1.   17.  ]
 [ 2.    0.  ]
 [ 0.    5.  ]]
```



3 PCA

3.1 1

Firstly we generate the data:[[1, 1], [1, 2], [2, 2], [2, 3], [3, 3], [3, 4], [4, 4], [4, 5], [5, 5], [5, 6], [6, 6], [6, 7], [7, 7], [7, 8], [8, 8], [8, 9], [9, 9], [9, 10], [10, 10], [10, 11]].
Then center the data by subtract its average, and transpose it:
((-4.5 -4.5 -3.5 -3.5 -2.5 -2.5 -1.5 -1.5 -0.5 -0.5 0.5 0.5 1.5 1.5 2.5 2.5 3.5 3.5 4.5 4.5)
(-5. -4. -4. -3. -3. -2. -2. -1. -1. 0. 0. 1. 1. 2. 2. 3. 3. 4. 4. 5.))

3.2 2

Compute the covariance matrix for centered data X using:

$$S = \frac{1}{n-1} X^T X$$

.
Covariance matrix for X is:

8.684	8.684
8.684	8.947

3.3 3

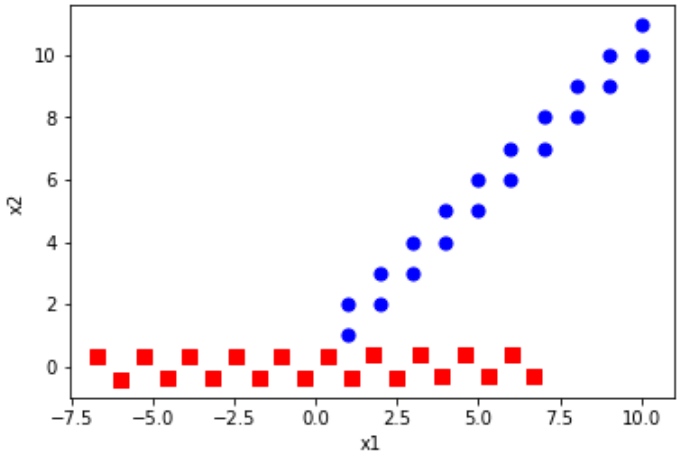
Using numpy package in python, we can perform eigen-decomposition and get two principal components v_1, v_2.
Two principal component is:

v_1	-0.712	-0.702
v_2	0.702	-0.712

3.4 4

Compute the new two-dim coordinates is:
((6.72 6.008 5.306 4.593 3.892 3.179 2.477 1.765 1.063 0.351 -0.351 -1.063 -1.765 -2.477 -3.179 -3.892 -4.593 -5.306 -6.008 -6.72)
(-0.303 0.399 -0.313 0.388 -0.324 0.378 -0.335 0.367 -0.346 0.356 -0.356 0.346 -0.367 0.335 -0.378 0.324 -0.388 0.313 -0.399 0.303))
The visualized result is shown below(blue dots are original data,red dots are new data):

```
In [1]: runfile('F:/project-gwu/Machine Learning/midterm/pca.py', wdir='F:
step 1: generating data...
[[1, 1], [1, 2], [2, 2], [2, 3], [3, 3], [3, 4], [4, 4], [4, 5], [5, 5], [
step 2: running PCA...
covariance matrix for X is:
[[8.684 8.684]
 [8.684 8.947]]
Two principal component is:
[[-0.712 -0.702]
 [ 0.702 -0.712]]
final data is:
[[ 6.72   6.008  5.306  4.593  3.892  3.179  2.477  1.765  1.063  0.351
  -0.351 -1.063 -1.765 -2.477 -3.179 -3.892 -4.593 -5.306 -6.008 -6.72 ]
 [-0.303  0.399 -0.313  0.388 -0.324  0.378 -0.335  0.367 -0.346  0.356
  -0.356  0.346 -0.367  0.335 -0.378  0.324 -0.388  0.313 -0.399  0.303]]
recon mat is:
[[ 1.  1.  2.  2.  3.  3.  4.  4.  5.  5.  6.  6.  7.  7.  8.  8.  9.  9.
  10. 10.]
 [ 1.  2.  2.  3.  3.  4.  4.  5.  5.  6.  6.  7.  7.  8.  8.  9.  9. 10.
  10. 11.]]
step 3: plot...
```



Using $reconMat = (newdata * eigenvector) + average$ to reconstruct the original data:
((1. 1. 2. 2. 3. 3. 4. 4. 5. 5. 6. 6. 7. 7. 8. 8. 9. 9. 10. 10.)
(1. 2. 2. 3. 3. 4. 4. 5. 5. 6. 6. 7. 7. 8. 8. 9. 9. 10. 10. 11.))
Reconstructed data is the same as original data, showing the result is correct.