CS 573: Assignment 2

Tiantu Xu

March 6, 2019

1. **Preprocessing**

   In Question 1, data pre-processing is run on the command line below:

   ```
   $ python preprocess.py dating-full.csv dating.csv
   ```

   The output from my code is

   ```
   Quotes removed from 8316 cells.
   Standardized 5707 cells to lower case.
   Value assigned for male in column gender: 1.
   Value assigned for European/Caucasian-American in column race: 2.
   Value assigned for Latino/Hispanic American in column race_o: 3.
   Value assigned for law in column field: 121.
   Mean of attractive_important: 0.22.
   Mean of sincere_important: 0.17.
   Mean of intelligence_important: 0.20.
   Mean of funny_important: 0.17.
   Mean of ambition_important: 0.11.
   Mean of shared_interests_important: 0.12.
   Mean of pref_o_attractive: 0.22.
   Mean of pref_o_sincere: 0.17.
   Mean of pref_o_intelligence: 0.20.
   Mean of pref_o_funny: 0.17.
   Mean of pref_o_ambitious: 0.11.
   Mean of pref_o_shared_interests: 0.12.
   ```

2. **Visualizing interesting trends in data**

   (a) The barplot below shows how females and males value the six attributes in their romantic partners differently. Males favors more in attribute "attractive" in their romantic partner, while females favor more in attribute "intelligence", and attributes like "attractive", "sincere", and "funny" are pretty close to it.

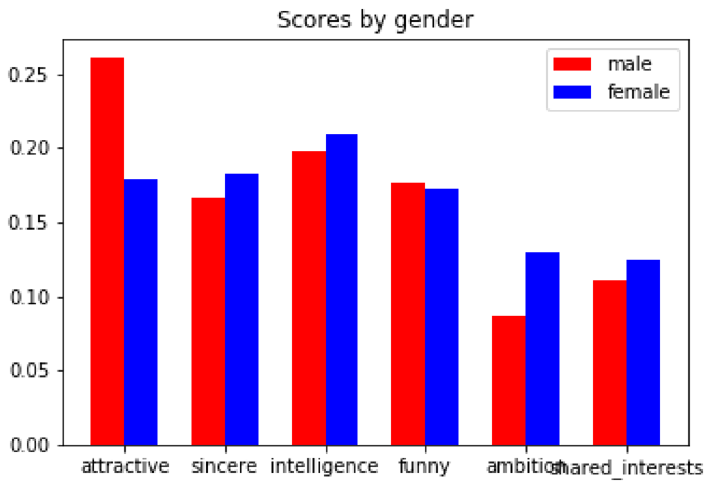

Figure 1: How females and males value the six attributes in their romantic partners

(b) The scatter plot below shows how the ratings on the partner will influence the success rate to give a second date. With high ratings on the partner, it would have higher chance that she/he will give their partner a second date.

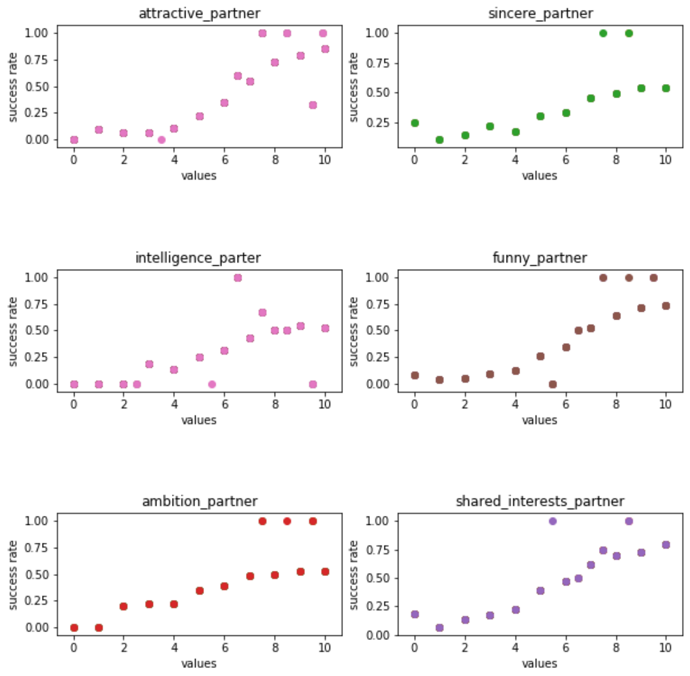

Figure 2: Success rate on ratings of the partner on 6 attributes

3. **Convert continuous attributes to categorical attributes**

Question 3 is run on the command line below.

```
$ python discretize.py dating.csv dating-binned.csv
```

The output from my code is

```
age: [3710, 2932, 97, 0, 5]
age_o: [3704, 2899, 136, 0, 5]
importance_same_race: [2980, 1213, 977, 1013, 561]
importance_same_religion: [3203, 1188, 1110, 742, 501]
pref_o_attractive: [4333, 1987, 344, 51, 29]
pref_o_sincere: [5500, 1225, 19, 0, 0]
pref_o_intelligence: [4601, 2062, 81, 0, 0]
pref_o_funny: [5616, 1103, 25, 0, 0]
pref_o_ambitious: [6656, 88, 0, 0, 0]
pref_o_shared_interests: [6467, 277, 0, 0, 0]
attractive_important: [4323, 2017, 328, 57, 19]
sincere_important: [5495, 1235, 14, 0, 0]
intelligence_important: [4606, 2071, 67, 0, 0]
funny_important: [5588, 1128, 28, 0, 0]
ambition_important: [6644, 100, 0, 0, 0]
shared_interests_important: [6494, 250, 0, 0, 0]
attractive: [18, 276, 1462, 4122, 866]
sincere: [33, 117, 487, 2715, 3392]
intelligence: [34, 185, 1049, 3190, 2286]
funny: [0, 19, 221, 3191, 3313]
ambition: [84, 327, 1070, 2876, 2387]
attractive_partner: [284, 948, 2418, 2390, 704]
sincere_partner: [94, 353, 1627, 3282, 1388]
intelligence_parter: [36, 193, 1509, 3509, 1497]
funny_partner: [279, 733, 2296, 2600, 836]
ambition_partner: [119, 473, 2258, 2804, 1090]
shared_interests_partner: [701, 1269, 2536, 1774, 464]
sports: [650, 961, 1369, 2077, 1687]
tvsports: [2151, 1292, 1233, 1383, 685]
exercise: [619, 952, 1775, 2115, 1283]
dining: [39, 172, 1118, 2797, 2618]
museums: [117, 732, 1417, 2737, 1741]
art: [224, 946, 1557, 2500, 1517]
hiking: [963, 1386, 1575, 1855, 965]
gaming: [2565, 1522, 1435, 979, 243]
clubbing: [912, 1068, 1668, 2193, 903]
reading: [131, 398, 1071, 2317, 2827]
tv: [1188, 1216, 1999, 1642, 699]
theater: [288, 811, 1585, 2300, 1760]
movies: [45, 248, 843, 2783, 2825]
concerts: [222, 777, 1752, 2282, 1711]
music: [62, 196, 1106, 2583, 2797]
shopping: [1093, 1098, 1709, 1643, 1201]
```

```
yoga: [2285, 1392, 1369, 1056, 642]
interests_correlate: [18, 758, 2453, 2942, 573]
expected_happy_with_sd_people: [321, 1262, 3292, 1596, 273]
like: [273, 865, 2539, 2560, 507]
```

4. **Training-Test Split**

   Question 4 is run on the command line below.

   ```
   $ python split.py
   ```

5. **Implement a Naive Bayes Classifier**

   (a) Question 5.1 is run on the command line below.

   ```
   $ python 5_1.py
   ```

   The output from my code is

   ```
   Training Accuracy: 0.78
   Test Accuracy: 0.75
   ```

   (b) Question 5.2 is run on the command line below.

   ```
   $ python 5_2.py
   ```

   The figure is shown below. For each bin value, the script will discretize once, so that the total time may take a few minutes.

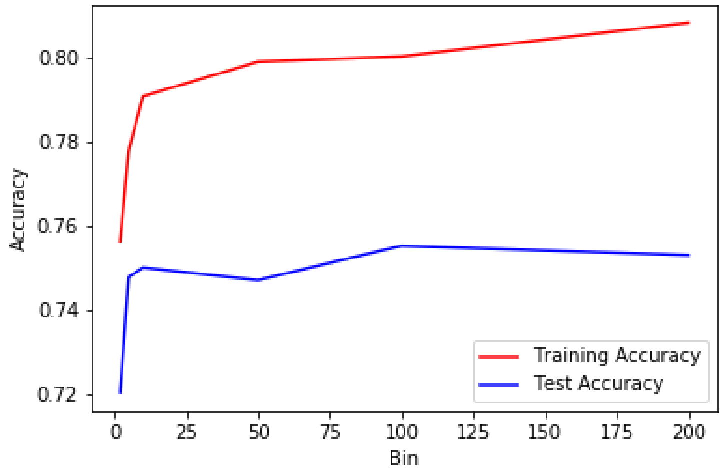

Figure 3: How bin value affects the learned NBC model's accuracy

From the figure, we can see that when b increases, the training and test accuracy tends to increase and converge. When b is small, the accuracy is not high, because the bin values are too coarse-grained. Further fine-grained discrete data will have higher accuracy, but test accuracy tends to converge earlier. This is because the trained NBC may tends to overfit with too fine-grained discrete data, which is an overkill to real case as tested in the test set.

The output from my code is

```
Bin size: 2
Training Accuracy: 0.76
Test Accuracy: 0.72
Bin size: 5
Training Accuracy: 0.78
Test Accuracy: 0.75
Bin size: 10
Training Accuracy: 0.79
Test Accuracy: 0.75
Bin size: 50
Training Accuracy: 0.80
Test Accuracy: 0.75
Bin size: 100
Training Accuracy: 0.80
Test Accuracy: 0.76
Bin size: 200
Training Accuracy: 0.81
Test Accuracy: 0.75
```

(c) Question 5.3 is run on the command line below.

```
$ python 5_3.py
```

The figure is shown below:

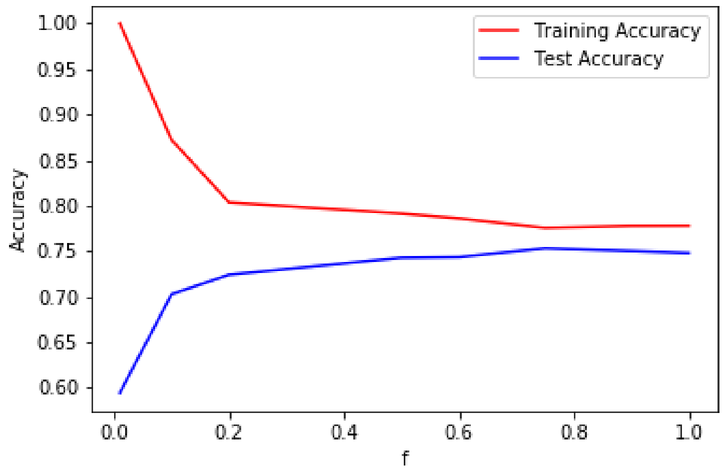

Figure 4: How f affects the learned NBC model's accuracy

From the figure, we can see that when f increases, the training accuracy will drop. The reason is when the training set is too small, NBC will learn the training data very well and captures little information on the whole distribution. When f further becomes large, the training accuracy will converge to 0.78, which is identical to the value reported in previous questions.

When f increases, the test accuracy will gradually increase until converge to 0.75. When f is too small, the learned NBC model is cannot have a good understanding of the real distribution, thus the test accuracy is low. When the training sample size is larger, the NBC model can learn the distribution well, thus will have higher test accuracy that converges to 0.75

The raw data output from python script is shown below.

```
f = 0.01
Training Accuracy: 1.00
Test Accuracy: 0.59
f = 0.1
Training Accuracy: 0.87
Test Accuracy: 0.70
f = 0.2
Training Accuracy: 0.80
Test Accuracy: 0.72
f = 0.5
Training Accuracy: 0.79
Test Accuracy: 0.74
f = 0.6
Training Accuracy: 0.79
Test Accuracy: 0.74
f = 0.75
Training Accuracy: 0.78
Test Accuracy: 0.75
f = 0.9
Training Accuracy: 0.78
Test Accuracy: 0.75
f = 1
Training Accuracy: 0.78
Test Accuracy: 0.75
```