Machine Learning Homework 2

Naive Bayes classifier

Create a Naive Bayes classifier for each handwritten digit that support **discrete** and **continuous** features.

- Input:
  1. Training image data from MNIST
     - You Must download the MNIST from HW2 in E3 system and parse the data by yourself. (Do **not** use the build in dataset or you'll not get 100.)
     - Please read the description in the link to understand the format.
     - Basically, each image is represented by $28 \times 28 \times 8$ bits (Whole binary file is in **big endian** format; you need to deal with it), you can use `char` array to store an image.
     - There are some headers you need to deal with as well, please read the link for more details.
  2. Training label data from MNIST
  3. Testing image from MNIST
  4. Testing label from MNIST
  5. Toggle option
     - 0: discrete mode
     - 1: continuous mode

TRAINING SET IMAGE FILE (`train-images-idx3-ubyte`)

Offset	Type	Value	Description
0000	32 bit integer	0x00000803 (2051)	magic number
0004	32 bit integer	60000	number of images
0008	32 bit integer	28	number of rows
0012	32 bit integer	28	number of columns
0016	unsigned byte	?	pixel
0017	unsigned byte	?	pixel
...	...	...	...
xxxx	unsigned byte	?	pixel

TRAINING SET LABEL FILE (`train-labels-idx1-ubyte`)

Offset	Type	Value	Description
0000	32 bit integer	0x00000801 (2049)	magic number
0004	32 bit integer	60000	number of images

Offset	Type	Value	Description
0008	unsigned byte	?	label
0009	unsigned byte	?	label
...	...	...	...
xxxx	unsigned byte	?	label

- Output:
  - Print out the the posterior (in log scale to avoid underflow) of the ten categories (0-9) for each image in **INPUT 3**. Don't forget to marginalize them so sum it up will equal to 1.
  - For each test image, print out your prediction which is the category having the highest posterior, and tally the prediction by comparing with **INPUT 4**.
  - Print out the imagination of numbers in your Bayes classifier
    - For each digit, print a 28×28 binary image which 0 represents a white pixel, and 1 represents a black pixel.
    - The pixel is 0 when Bayes classifier expect the pixel in this position should less then 128 in original image, otherwise is 1.
  - Calculate and report the error rate in the end.
- Function:
  1. In Discrete mode:
     - Tally the frequency of the values of each pixel into 32 bins. For example, The gray level 0 to 7 should be classified to bin 0, gray level 8 to 15 should be bin 1, ... etc. Then perform Naive Bayes classifier. **Note** that to avoid empty bin, you can use a pseudocount (such as the minimum value in other bins) for instead.
  2. In Continuous mode:
     - Use MLE to fit a Gaussian distribution for the value of each pixel. Perform Naive Bayes classifier.
- Sample output **(for reference only)**

```
Postirior (in log scale):
0: 0.11127455255545808
1: 0.11792841531242379
2: 0.1052274113969039
3: 0.10015879429196257
4: 0.09380188902719812
5: 0.09744539128015761
6: 0.1145761939658308
7: 0.07418582789605557
8: 0.09949702276138589
9: 0.08590450151262384
Prediction: 7, Ans: 7

Postirior (in log scale):
0: 0.10019559729888124
1: 0.10716826094630129
```

```
2: 0.08318149248873129
3: 0.09027637439145528
4: 0.10883493744297462
5: 0.09239544343955365
6: 0.08956194806124541
7: 0.11912349865671235
8: 0.09629347315717969
9: 0.11296897411696516
Prediction: 2, Ans: 2

... all other predictions goes here ...

Imagination of numbers in Bayesian classifier:

0:
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 1 1 1 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0 0 1 1 1 1 1 1 1 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0 1 1 1 1 1 1 1 1 1 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 1 1 1 1 1 1 1 1 1 1 1 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 1 1 1 1 1 0 0 0 1 1 1 1 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 1 1 1 1 0 0 0 0 0 0 1 1 1 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 1 1 1 0 0 0 0 0 0 0 0 1 1 1 0 0 0 0 0
0 0 0 0 0 0 0 0 1 1 1 0 0 0 0 0 0 0 0 0 1 1 1 0 0 0 0 0
0 0 0 0 0 0 0 1 1 1 1 0 0 0 0 0 0 0 0 0 1 1 1 0 0 0 0 0
0 0 0 0 0 0 0 1 1 1 0 0 0 0 0 0 0 0 0 0 1 1 1 0 0 0 0 0
0 0 0 0 0 0 0 1 1 1 0 0 0 0 0 0 0 0 0 0 1 1 0 0 0 0 0 0
0 0 0 0 0 0 1 1 1 0 0 0 0 0 0 0 0 0 0 1 1 1 0 0 0 0 0 0
0 0 0 0 0 0 1 1 1 0 0 0 0 0 0 0 0 0 1 1 1 0 0 0 0 0 0 0
0 0 0 0 0 0 1 1 1 0 0 0 0 0 0 0 0 0 1 1 1 0 0 0 0 0 0 0
0 0 0 0 0 0 1 1 1 1 0 0 0 0 0 0 1 1 1 1 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 1 1 1 1 1 0 0 1 1 1 1 1 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 1 1 1 1 1 1 1 1 1 1 1 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 1 1 1 1 1 1 1 1 1 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 1 1 1 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0

... all other imagination of numbers goes here ...

9:
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0 1 1 1 1 1 1 0 0 0 0 0 0 0 0 0 0
```

```
0 0 0 0 0 0 0 0 0 0 0 1 1 1 1 1 1 1 1 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 1 1 1 1 0 0 0 1 1 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 1 1 1 0 0 0 0 0 1 1 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 1 1 0 0 0 0 0 0 1 1 1 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 1 1 1 0 0 0 0 0 1 1 1 1 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 1 1 0 0 0 0 0 1 1 1 1 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 1 1 0 0 1 1 1 1 1 1 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 1 1 1 1 1 1 1 1 1 1 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 1 1 1 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 1 1 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 1 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 1 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0

Error rate: 0.1535
```

Online learning

Use online learning to learn the beta distribution of the parameter `p` (chance to see 1) of the coin tossing trails in batch.

- Input:
  1. A file contains many lines of binary outcomes:

  ```
  0101010111011011010101
  0110101
  010110101101
  ```

  2. parameter `a` for the initial beta prior
  3. parameter `b` for the initial beta prior
- Output: Print out the Binomial likelihood (based on MLE, of course), Beta prior and posterior probability (parameters only) for each line.
- Function: Use Beta-Binomial conjugation to perform online learning.

Sample input & output (for reference only)

- Input: A file (here shows the content of the file)

```
0101010101001011010101
0110101
010110101101
0101101011101011010
111101100011110
101110111000110
1010010111
11101110110
01000111101
110100111
01101010111
```

- Output
    - Case 1: a = 0, b = 0

```
case 1: 0101010101001011010101
Likelihood: 0.16818809509277344
Beta prior:     a = 0   b = 0
Beta posterior: a = 11  b = 11

case 2: 0110101
Likelihood: 0.29375515303997485
Beta prior:     a = 11  b = 11
Beta posterior: a = 15  b = 14

case 3: 010110101101
Likelihood: 0.2286054241794335
Beta prior:     a = 15  b = 14
Beta posterior: a = 22  b = 19

case 4: 0101101011101011010
Likelihood: 0.18286870706509092
Beta prior:     a = 22  b = 19
Beta posterior: a = 33  b = 27

case 5: 111101100011110
Likelihood: 0.2143070548857833
Beta prior:     a = 33  b = 27
Beta posterior: a = 43  b = 32

case 6: 101110111000110
Likelihood: 0.20659760529408
Beta prior:     a = 43  b = 32
Beta posterior: a = 52  b = 38

case 7: 1010010111
Likelihood: 0.25082265600000003
Beta prior:     a = 52  b = 38
Beta posterior: a = 58  b = 42

case 8: 11101110110
Likelihood: 0.2619678932864457
Beta prior:     a = 58  b = 42
Beta posterior: a = 66  b = 45
```

```
case 9: 01000111101
Likelihood: 0.23609128871506807
Beta prior:     a = 66  b = 45
Beta posterior: a = 72  b = 50

case 10: 110100111
Likelihood: 0.27312909617436365
Beta prior:     a = 72  b = 50
Beta posterior: a = 78  b = 53

case 11: 01101010111
Likelihood: 0.24384881449471862
Beta prior:     a = 78  b = 53
Beta posterior: a = 85  b = 57
```

- Case 2: a = 10, b = 1

```
case 1: 0101010101001011010101
Likelihood: 0.16818809509277344
Beta prior:     a = 10  b = 1
Beta posterior: a = 21  b = 12

case 2: 0110101
Likelihood: 0.29375515303997485
Beta prior:     a = 21  b = 12
Beta posterior: a = 25  b = 15

case 3: 010110101101
Likelihood: 0.2286054241794335
Beta prior:     a = 25  b = 15
Beta posterior: a = 32  b = 20

case 4: 0101101011101011010
Likelihood: 0.18286870706509092
Beta prior:     a = 32  b = 20
Beta posterior: a = 43  b = 28

case 5: 111101100011110
Likelihood: 0.2143070548857833
Beta prior:     a = 43  b = 28
Beta posterior: a = 53  b = 33

case 6: 101110111000110
Likelihood: 0.20659760529408
Beta prior:     a = 53  b = 33
Beta posterior: a = 62  b = 39

case 7: 1010010111
Likelihood: 0.25082265600000003
Beta prior:     a = 62  b = 39
Beta posterior: a = 68  b = 43

case 8: 11101110110
Likelihood: 0.2619678932864457
```

```
Beta prior:     a = 68  b = 43
Beta posterior: a = 76  b = 46

case 9: 01000111101
Likelihood: 0.23609128871506807
Beta prior:     a = 76  b = 46
Beta posterior: a = 82  b = 51

case 10: 110100111
Likelihood: 0.27312909617436365
Beta prior:     a = 82  b = 51
Beta posterior: a = 88  b = 54

case 11: 01101010111
Likelihood: 0.24384881449471862
Beta prior:     a = 88  b = 54
Beta posterior: a = 95  b = 58
```

Prove Beta-Binomial conjugation

- Try to proof Beta-Binomial conjugation and write the process on paper.
- You should write down the proof process on paper and take a picture.

Notes

- Use whatever programming language you prefer.
- You **can't** use `numpy.random.beta` in HW2. That would be great if you implement all distribution by yourself.
- HW2 must contain your code and proof process (can be `.pdf` or any image format).