CS440 Assignment 3 - Pattern Recognition

Hanfei Lin(hanfeil2)
Jean Cai (zijunc2)
Zeyu Pan(zeyupan2)

- 4 credit -

Statement of individual contribution

Name	Part 1.1	Part 1.2	Part 1.3	Part 2.1	Part 2.2	Part 2.3.1	Part 2.3.2	Part 2.3.3
	Single pixels as features	Pixel groups as features	Face Classification	perceptrons	KNN	Visualization	Differentiable perceptron	Other learning algorithm
Hanfei Lin				☑	☑	☑	☑	☑
Jean Cai	☑	☑	☑					
Zeyu Pan	☑		☑					

Part 1: Naive Bayes Classifiers for Digit Classification

1.0 General Algorithm

We implemented the whole part 1 classifier in one framework, with ***smoothing constant 0.3***. There are six important parameters in classifier initiation:

int **typeNum**	The number of categories. In the part1.1 & 1.2, the typeNum would be 10 (from 0 to 9). In part1 extra, the typeNum would be 2 (is a face & is not a face).
int **imgHeight**	The height of each input image. In the part1.1 & 1.2, the imgHeight would be 32. In part1 extra, the typeNum would be 70.
int **imgWidth**	The width of each input image. In the part1.1 & 1.2, the imgHeight would be 32. In part1 extra, the typeNum would be 60.
int **windowHeight**	The height of square. In the part1.1 & 1.2, the windowHeight would be 1, because it is just one pixel. In part1 extra, the typeNum would be 2, 3, and 4.
int **windowWidth**	The width of square. In the part1.1 & 1.2, the windowHeight would be 1, because it is just one pixel. In part1 extra, the typeNum would be 2, 3, and 4.
boolean **isOverlap**	Determines if the square is overlapping. Designed for part1 extra.

So, by changing the parameters of the classifier, we can address different goals in part 1.1, 1.2 and 1.3 extra.

There are also some important member variables in the classifier:

int[] **typeCount**	The occurrence of training images of digit *i*, which is also the index of this array.
double[] **typeP**	P(class) of 0,1,2,3,4,5,6,7,8,9 = (**typeCount**[i] + ***SMOOTHINGVALUE***) / (total number of trainning images+ ***SMOOTHINGVALUE***)
int[][][][] **cMap**	Designed for training. Indexes are [i][j][G][type]. The value stores the number of occurence of this situation: the sum of 1(or #) in the square would be *G* in index *i,*

	j of the input image, when digit image shows digit *type*. i&j are indexs of image. G & type are mainly used for 1.2. G is total possible situation of occurrence of 1 in square. In the part1.1 & 1.2, G would be 0 or 1, type would be 0,1,2,3..9. In part1 extra, G could be from 0 to the sum of 1 in the square, type would be 0,1. For example, in part1.1, cMap[2][3][1][8] stores the number of occurence of this situation: 1 would occur in index 2, 3 of the input image when digit image shows 8; in part1.2, cMap[2][3][5][3] stores the number of occurence of this situation: the sum of 1 in the square would be 5 in index 2, 3 of the input image, when digit image shows 3; In part1.3 extra, cMap[2][3][4][1] stores the number of occurence of this situation: the sum of # in the square would be 4 in index 2, 3 of the input image, when input image shows a face.
double[][][][] **pMap**	Designed for training, stores possibility P(g(i,j)|class). **pMap**[i][j][G][type] = **cMap**[i][j][k][t] + ***SMOOTHINGVALUE***) / (**this**.**typeCount**[t] + ***SMOOTHINGVALUE***)

Core pseudocode:

TrainOneImg:

```
for i from 0 to cMap.length:
   for j from 0 to cMap[i].length:
      if( isOverlap)
         G <- image.getG(i, j, windowHeight, windowWidth)
      else
          G <- image.getG(i * windowHeight, j * windowWidth, windowHeight,
windowWidth);
      cMap[i][j][G][image.type]++

typeCount[image.type]++
```

TrainAllImg:

```
for each image:
  TrainOneImg(image)

for i from 0 to pMap.length:
   for j from 0 to pMap[i].length:
      for g from 0 to pMap[i][j].length:
        for t from 0 to typeCount.length:
            pMap[i][j][k][t] <- (cMap[i][j][k][t] + SMOOTHINGVALUE) /  (typeCount[t] +
SMOOTHINGVALUE);
```

```
totalCount <- typeCount.length

for i from 0 to typeP.length:
    typeP[i] <- ( typeCount[i] + SMOOTHINGVALUE) / (totalCount + SMOOTHINGVALUE)
```

PredictOneImg:

```
Res <- prediction score for all class (the higher, the more match)
Find the highest and lowest scores from res
Compare highest (and lowest) scores to the existing highest (and lowest) scores and update
them if higher (lower).

Return the predicted digit <- index of the highest score in res
```

PredictAllImg:

```
int[] testTypeCount <- new int[typeCount.length]
double[][] confusionArray <- new double[typeCount.length][typeCount.length]

For image in testImgList:
    predictOneImg
    If prediction is correct:
        totalCount++
    confusionArray[predictType][image.type]++
     testTypeCount[image.type]++

For i from 0 to confusionArray.length
    For j from 0 to confusionArray[i].length
         confusionArray[i][j] /= testTypeCount[i]

Return overall accuracy <- totalCount / number of images in imageList
```

1.1 Single Pixels as Features

1.1.1 Algorithm

General Algorithm has been stated above.

Set the typeNum to be 10 (digit 0 to 9), imgHieght and imgWidth to be 32, windowHeight and windowWidth to be 1 (one pixel), and isOverlap to be false.

1.1.2 Smoothing constant

We tried the several smoothing constant below to compare the test accuracy, and decided to use 0.3 as the smoothing constant in part 1.

Constant Value	0.1	0.3	0.6	0.8	1
Accuracy	0.939	0.939	0.937	0.935	0.935

1.1.3 Accuracy & Confusion Matrix

Percentage	**Prediction**									
Actual Num	**0**	**1**	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**
0	0.972	0	0	0	0.017	0	0	0	0	0
1	0	0.933	0	0	0	0	0	0.021	0.025	0.024
2	0	0	0.854	0	0	0	0	0	0.125	0.024
3	0	0	0	0.909	0	0	0	0.021	0	0.048
4	0	0	0	0	0.881	0	0	0.085	0.075	0
5	0	0	0	0	0	0.931	0	0	0	0.095
6	0	0	0	0	0.017	0	0.977	0	0	0
7	0	0	0	0	0	0	0	1	0	0
8	0	0	0	0	0	0	0	0	1	0
9	0	0	0	0.03	0	0	0	0.021	0	0.952

➔ Accuracy for each digit ： **diagonal element** on above confusion matrix.

➔ Overall accuracy : **0.939**

1.1.4 highest and lowest posterior probabilities

Below are the test tokens from each class that have the highest and lowest posterior probabilities.

digit	**Lowest**	**Highest**

0	00000000000001110000000000000000 00000000000011111100000000000000 00000000000111111110000000000000 00000000000111111111000000000000 00000000000111111111000000000000 00000000000011111111110000000000 00000000000111111001111000000000 00000000000111111001111000000000 00000000000111111001111000000000 00000000001111110001111000000000 00000000001111110000001110000000 00000000011111000000001110000000 00000000011111000000001110000000 00000000111110000000001110000000 00000000011110000000000111000000 00000000111110000000000111000000 00000000111100000000000111000000 00000001111100000000000111000000 00000001111000000000000111000000 00000000111100000000000011100000 00000000111100000000000011100000 00000000111100000000000011100000 00000001111000000000000011100000 00000000111100000000000011100000 00000000011110000000000111100000 00000000011110000000001111100000 00000000011111000011111111100000 00000000011111111111111111000000 00000000000111111111111111000000 00000000000111111111111110000000 00000000000011111111110000000000 00000000000000111111000000000000	00000000000000011000000000000000 00000000000011111110000000000000 00000000001111111111000000000000 00000000001111111111100000000000 00000000011111111111100000000000 00000000011111111111110000000000 00000000111111111111110000000000 00000000111111110001111100000000 00000000111111110000011110000000 00000000111111110000011110000000 00000001111111000000011110000000 00000001111111000000011110000000 00000011111100000000011110000000 00000011111100000000011110000000 00000001111100000000001111000000 00000001111000000000001111000000 00000001111000000000001111000000 00000001111000000000001111000000 00000001111000000000001111000000 00000000111100000000000111100000 00000000111000000000001111000000 00000001111000000000011111000000 00000001111000000000011110000000 00000001111000000000111110000000 00000001111000000011111110000000 00000000111100001111111110000000 00000000011111111111111100000000 00000000011111111111111100000000 00000000011111111111110000000000 00000000001111111111100000000000 00000000000011111111100000000000 00000000000000111000000000000000
1	00000000000000000111111000000000 00000000000000000111111100000000 00000000000000000111111110000000 00000000000000000111111110000000 00000000000000000111111110000000 00000000000000000111111110000000 00000000000000000111111110000000 00000000000000000111111110000000 00000000000000001111111110000000 00000000000000001111111110000000 00000000000000111111111100000000 00000011111111111111111100000000 00000111111111111111111100000000 00000011111111111111111100000000 00000011111111111111111100000000 00000011111111111111111100000000 00000001111111111111111100000000 00000000111110000111111100000000 00000000000000001111111100000000 00000000000000001111111100000000 00000000000000001111111110000000 00000000000000001111111110000000 00000000000000001111111110000000 00000000000000001111111110000000 00000000000000001111111110000000 00000000000000001111111110000000 00000000000000001111111110000000 00000000000000000111111111000000 00000000000000000111111111000000 00000000000000000111111111100000 00000000000000000011111111000000 00000000000000000001111110000000	00000000000000000000000000000000 00000000000000001111111000000000 00000000000011111111111000000000 00000000000011111111111000000000 00000000000011111111110000000000 00000000000111111111111000000000 00000000011111111111110000000000 00000000011111111111100000000000 00000000011111111111110000000000 00000000011111111111111000000000 00000000000111111111110000000000 00000000000111111111110000000000 00000000000111111111110000000000 00000000111111111111110000000000 00000000011111111111100000000000 00000000000111111111100000000000 00000000000111111111100000000000 00000000000111111111110000000000 00000000000111111111111000000000 00000000011111111111110000000000 00000000011111111111110000000000 00000000011111111111110000000000 00000000011111111111110000000000 00000000011111111111110000000000 00000000011111111111111000000000 00000000000111111111111000000000 00000000000111111111111000000000 00000000000111111111110000000000 00000000000111111111100000000000 00000000000111111111110000000000 00000000000111111111100000000000 00000000000000000000000000000000
2	00000000011111000000000000000000 00000000111111100000000000000000 00000001111111110000000000000000 00000011111111110000000000000000 00000011111111111000000000000000 00000111111111111000000000000000 00000111111111111100000000000000 00001111110011111100000000000000 00001111110001111100000000000000 00001111110001111100000000000000 00000111111000111110000000000000 00000111110000111100000000000000 00000011100001111100000000000000 00000000000001111100000000000000 00000000000001111100000000000000 00000000000001111100000000000000 00000000000001111100000000000000	00000000000011100000000000000000 00000000001111111000000000000000 00000000011111111100000000000000 00000001111111111100000000000000 00000001111111111110000000000000 00000001111111111110000000000000 00000001111111111110000000000000 00000001111101111110000000000000 00000001110000111110000000000000 00000000000000001111000000000000 00000000000000001111000000000000 00000000000000000111100000000000 00000000000000001111000000000000 00000000000000001111000000000000 00000000000000001111000000000000 00000000000000001111000000000000 00000000000000001111100000000000

	00000000000011111000000000000000 00000000000011110000000000000000 00000000000011110000000000000000 00000000000111110000000000000000 00000000001111110000000000000000 00000000001111100000000000000000 00000000000111110000000000000000 00000000001111100000000000000000 00000000011111101111111111100000 00000000111111111111111111110000 00000001111111111111111111110000 00000011111111111111111111100000 00000001111111111111111111000000 00000001111111111111111100000000 00000000111111111111111000000000	00000000000000011110000000000000 00000000000000011110000000000000 00000000000000111100000000000000 00000000000001111100000000000000 00000000000011111000000000000000 00000000001111110000000000000000 00000000001111110000000010000000 00000000011111110000001110000000 00000000011111111111111110000000 00000000011111111111111110000000 00000000011111111111111110000000 00000000011111111111111110000000 00000000011111111111111110000000 00000000011111111111100000000000 00000000001111111000000000000000
3	00000000001111111111110000000000 00000000011111111111111100000000 00000001111111111111111100000000 00000011111111111111111111000000 00000011111111111111111111100000 00000011111111000000111111110000 00000011100000000000011111110000 00000011100000000000011111110000 00000001000000000000011111110000 00000000000000000000111111100000 00000000000000000011111111000000 00000000000000001111111110000000 00000000000000011111111100000000 00000000000000011111110000000000 00000000000000111111100000000000 00000000000001111111000000000000 00000000000001111110000000000000 00000000000001111110000000000000 00000000000000111111000000000000 00000000000000111111000000000000 00000000000000011111100000000000 00000000000000001111110000000000 00000000000000000111111000000000 00000000000000001111110000000000 00000000000000000111111000000000 00000000000000000111111000000000 00000000011000000111111000000000 00000000111100111111111100000000 00000001111111111111110000000000 00000001111111111111100000000000 00000000111111111111000000000000 00000000111111111100000000000000	00000000000011111110000000000000 00000000011111111111100000000000 00000001111111111111110000000000 00000011111111111111111000000000 00000011111111111111111000000000 00000011111111111111110000000000 00000001110000001111110000000000 00000000000000000111111000000000 00000000000000000111111000000000 00000000000000001111100000000000 00000000000000001111100000000000 00000000000000011111100000000000 00000000000000111111100000000000 00000000000001111111100000000000 00000000000001111111100000000000 00000000000001111111110000000000 00000000000000011111111000000000 00000000000000001111111100000000 00000000000000000111111110000000 00000000000000000011111110000000 00000000000000000000011111000000 00000000000000000000011110000000 00000000000000000000011111000000 00000000000000000000011111000000 00000000000000000000011111000000 00000000000000000000011111000000 00000000011110000000011111100000 00000001111111111111111111000000 00000001111111111111111111000000 00000001111111111111111110000000 00000000111111111111111100000000 00000000000000000111000000000000
4	00000000000000000011110000000000 00000000000000000011111000000000 00000000000000000011111000000000 00000000000000001111110000000000 00000000000000011111110000000000 00000000000000011111100000000000 00000000000000111111000000000000 00000000000001111111000000000000 00000000000111111000000000000000 00000000000111111000000000000000 00000000000111110000000111000000 00000000001111100000011111100000 00000000011111000000011111110000 00000000111111000000011111110000 00000001111110000000011111100000 00000011111100000000011111100000 00000011111111111111111111100000 00000111111111111111111111100000 00000011111111111111111111100000 00000111111111111111111111000000 00000111111111111111111111000000 00000011111111111111111111000000 00000000000001011111111111000000 00000000000000000001111110000000 00000000000000000011111110000000 00000000000000000011111110000000 00000000000000000011111110000000 00000000000000000011111100000000 00000000000000000011111100000000 00000000000000000011111100000000 00000000000000000111110000000000 00000000000000000111100000000000	00000000000000111100000000000000 00000000000000111110000000000000 00000000000000111110000000000000 00000000000001111110000000000000 00000000000001111111000000000000 00000000000001111111000000000000 00000000001111111000000000000000 00000000001111110000000000000000 00000000001111110000000000000000 00000000001111111000000000000000 00000000011111110000000011000000 00000000111111100000000111100000 00000000111111100000001111110000 00000001111111000000111111100000 00000011111111000000111111100000 00000011111110000001111111100000 00000011111100000111111111100000 00000011111111111111111111000000 00000111111111111111111111000000 00000111111111111111111111000000 00000111111111111111111100000000 00000011111111111111111100000000 00000011111111111111111000000000 00000000001111001111111000000000 00000000000000001111110000000000 00000000000000001111110000000000 00000000000000011111100000000000 00000000000000111111000000000000 00000000000000111111100000000000 00000000000000111111100000000000 00000000000000111111100000000000 00000000000000011111100000000000

5	00000000000000000000111111111000 00000000000000111111111111110000 00000000000111111111111111111000 00000000111111111111111111110000 00000000111111111111111111110000 00000000111111111111111111000000 00000000000111111111100000000000 00000000000111111110000000000000 00000000000111111000000000000000 00000000001111110000000000000000 00000000000111111000000000000000 00000000000111111110000000000000 00000000000011111111000000000000 00000000000001111111000000000000 00000000000001111111100000000000 00000000000000111111100000000000 00000000000000001111110000000000 00000000000000000111111000000000 00000000000000000011111100000000 00000000000000000111111000000000 00000000000000000111110000000000 00000000000000000111111000000000 00000000000000000111110000000000 00000000000000001111110000000000 00000000000000011111100000000000 00000000000001111111000000000000 00000000000011111110000000000000 00000000000111111100000000000000 00000000001111111000000000000000 00000000000111111000000000000000 00000000001111100000000000000000 00000000001111000000000000000000	00000000000000000000100000000000 00000000000011111111111000000000 00000000001111111111111111000000 00000000011111111111111110000000 00000000111111111111111111000000 00000000111111111111100110000000 00000001111110000000000000000000 00000001111110000000000000000000 00000001111110000000000000000000 00000001111100000000000000000000 00000001111110000000000000000000 00000001111111111110000000000000 00000001111111111111000000000000 00000011111111111111100000000000 00000011111111111111100000000000 00000011111111111111100000000000 00000011111111111111110000000000 00000001111100000111111000000000 00000000000000000011111100000000 00000000000000000011111100000000 00000000000000000001111100000000 00000000000000000001111110000000 00000000000000000001111100000000 00000000000000000001111110000000 00000000000000000001111110000000 00000000000000000111111110000000 00000000000000111111111100000000 00000000001111111111111100000000 00000000001111111111110000000000 00000000001111111111100000000000 00000000001111110000000000000000 00000000001100000000000000000000
6	00000000000111100000000000000000 00000000000111110000000000000000 00000000001111110000000000000000 00000000001111110000000000000000 00000000001111100000000000000000 00000000011111110000000000000000 00000000111111100000000000000000 00000000111110000000000000000000 00000000111111000000000000000000 00000000111110000000000000000000 00000000111110000000000000000000 00000001111110000000000000000000 00000001111110000000000000000000 00000011111100000000000000000000 00000011111000000000000000000000 00000011111100000000000000000000 00000011111100000000000000000000 00000011111000000100000000000000 00000011111000111111110000000000 00000011111111111111111100000000 00000011111111111111111111000000 00000011111111111111111111100000 00000011111111111111111111100000 00000001111111000000001111110000 00000001111111000000000111110000 00000001111111000000000011111000 00000000111111000000000011111000 00000000111111100000000011111000 00000000011111111111111111111000 00000000001111111111111111110000 00000000000011111111111111110000 00000000000001111111111111000000	00000000000011100000000000000000 00000000000111100000000000000000 00000000000111100000000000000000 00000000001111100000000000000000 00000000001111100000000000000000 00000000001111110000000000000000 00000000001111110000000000000000 00000000001111110000000000000000 00000000001111100000000000000000 00000000001111100000000000000000 00000000011111100000000000000000 00000000011111000000000000000000 00000000111111000000000000000000 00000000011111100000000000000000 00000000011111100000000000000000 00000000111111000000000000000000 00000000111111000000000000000000 00000000111111011111100000000000 00000000111111111111111000000000 00000000111111111111111100000000 00000000111111111111111110000000 00000000111111111111111111000000 00000001111111111111111111000000 00000000011111100000011111100000 00000000011111100000011111100000 00000000011111100000011111000000 00000000011111100000111110000000 00000000000111111111111111100000 00000000000011111111111110000000 00000000000011111111111110000000 00000000000001111111111100000000 00000000000000001111110000000000
7	00000000000000001111111111100000 00000000000111111111111111110000 00000000001111111111111111111000 00000000011111111111111111111100 00000000011111111111111111111100 00000000111111111111111111111100 00000000111111111111111111111100 00000000011111110000000111111100 00000000000000000000011111111000 00000000000000000000011111111000 00000000000000000000111111111000 00000000000000000000111111110000 00000000000000000000111111100000 00000000000000000000111111100000 00000000000000000001111111100000 00000000000000000011111111000000 00000000000000000011111110000000	00000000000000111111111100000000 00000000000111111111111110000000 00000000001111111111111111000000 00000000011111111111111110000000 00000000111111111111111111000000 00000000111111111111111111000000 00000000011111111111111111000000 00000000000000000001111110000000 00000000000000000011111110000000 00000000000000000011111110000000 00000000000000000011111110000000 00000000000000000111111000000000 00000000000000000111111000000000 00000000000000000111111000000000 00000000000000111111111110000000 00000000000111111111111110000000 00000000001111111111111111000000

	00000000000000000001111111000000 00000000000000000111111110000000 00000000000000001111111100000000 00000000000000001111111100000000 00000000000000011111111000000000 00000000000000111111110000000000 00000000000001111111100000000000 00000000000001111111100000000000 00000000000011111111000000000000 00000000000111111110000000000000 00000000000111111110000000000000 00000000001111111100000000000000 00000000001111111100000000000000 00000000001111111000000000000000 00000000001111111000000000000000 00000000001111110000000000000000	00000000001111111111111111100000 00000000011111111111111111000000 00000000011111111111111111000000 00000000111111111111111110000000 00000000000001111110000000000000 00000000000011111100000000000000 00000000000011111100000000000000 00000000000111111000000000000000 00000000000111111000000000000000 00000000001111110000000000000000 00000000001111110000000000000000 00000000011111100000000000000000 00000000011111100000000000000000 00000000111111000000000000000000 00000000111111000000000000000000 00000000111110000000000000000000
8	00000000000000011111111100000000 00000000000111111111111110000000 00000000111111111111111111000000 00000001111111111111111111100000 00000001111111111111111111100000 00000000000111110000111111000000 00000000001111110000011111000000 00000000001111110000011111000000 00000000001111110000011111000000 00000000001111110000011111000000 00000000001111110000011111000000 00000000001111110000011111000000 00000000001111110000011111000000 00000000001111110111111100000000 00000000001111111111111100000000 00000000001111111111111100000000 00000000001111111111111000000000 00000000001111111111111000000000 00000000001111111111110000000000 00000000001111111111110000000000 00000000001111111111110000000000 00000000001111111111100000000000 00000000001111111111100000000000 00000000001111100111110000000000 00000001111100001111000000000000 00000001111000011111000000000000 00000001111000011111000000000000 00000001111000011111000000000000 00000001111000011111000000000000 00000001111100011111000000000000 00000001111111111111110000000000 00000001111111111111110000000000 00000001111111111111100000000000 00000000111111111111100000000000 00000000011111111111000000000000 00000000001111110000000000000000	00000000000000011100000000000000 00000000000001111111100000000000 00000000011111111111110000000000 00000001111111111111111100000000 00000001111111111111111100000000 00000001111111100111111110000000 00000001111111000001111110000000 00000001111111000001111110000000 00000001111111000001111110000000 00000001111111000001111110000000 00000001111111000001111110000000 00000001111111001111111100000000 00000000001111111111111110000000 00000000001111111111111100000000 00000000001111111111111000000000 00000000001111111111110000000000 00000000001111111111110000000000 00000000001111111111100000000000 00000001111111111110000000000000 00000001111111111110000000000000 00000001111111001111110000000000 00000001111110000111110000000000 00000001111000001111110000000000 00000001111000001111110000000000 00000001111000001111110000000000 00000001111000001111110000000000 00000001111100001111100000000000 00000001111111111111111000000000 00000001111111111111111000000000 00000000111111111111111000000000 00000000011111111111110000000000 00000000001111111111100000000000
9	00000000000000000000111000000000 00000000000000001111111110000000 00000000000001111111111111000000 00000000000011111111111111100000 00000000000011111111000011100000 00000000001111111000001111000000 00000000011111111000001111100000 00000000011111111000001111100000 00000000111111110000011111000000 00000000111111111111111111100000 00000000111111111111111111110000 00000000111111111111111111100000 00000000111111111111111111000000 00000000001111110111111100000000 00000000000000000000011111000000 00000000000000000000111110000000 00000000000000000000111110000000 00000000000000000000011111000000 00000000000000000000011110000000 00000000000000000000111110000000 00000000000000000000111100000000 00000000000000000000111100000000 00000000000000000000011110000000 00000000000000000000011110000000 00000000000000000000011100000000 00000000000000000000011100000000 00000000000000000000011000000000 00000000000000000000011000000000	00000000000001111100000000000000 00000000000011111110000000000000 00000000011111111110000000000000 00000001111111111110000000000000 00000001111111111110000000000000 00000001111111100111111100000000 00000001111111000011111100000000 00000001111111000011111100000000 00000001111111000011111100000000 00000001111111000011111100000000 00000001111111000011111100000000 00000001111111001111111100000000 00000000111111111111111100000000 00000000111111111111111100000000 00000000111111111111111000000000 00000000111111111111110000000000 00000000111111111111110000000000 00000000111111111111100000000000 00000001111111111110000000000000 00000001111111111110000000000000 00000001111111001111110000000000 00000001111110000111110000000000 00000001111000001111110000000000 00000001111000001111110000000000 00000001111000001111110000000000 00000001111000001111110000000000 00000001111100001111100000000000 00000001111111111111111000000000 00000001111111111111111000000000 00000000111111111111111000000000 00000000011111111111110000000000 00000000001111111111100000000000

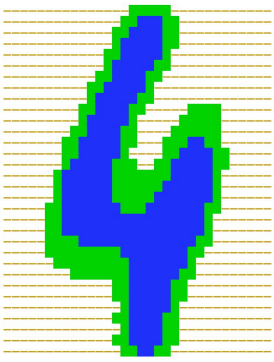
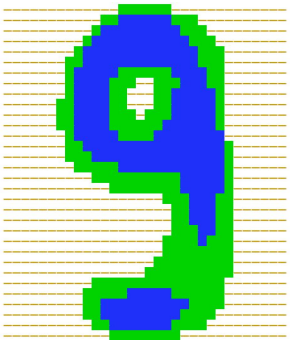
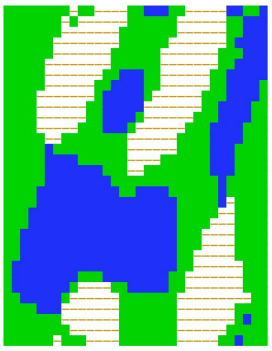
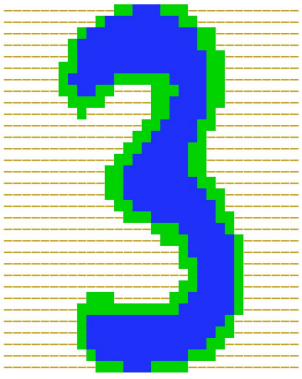
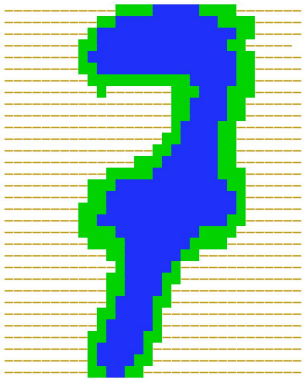
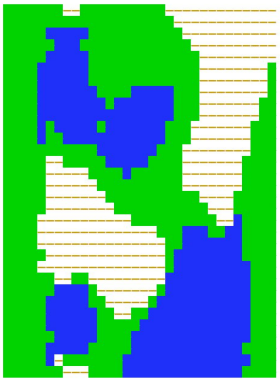
1.1.3 Likelihood & Odd Ratios

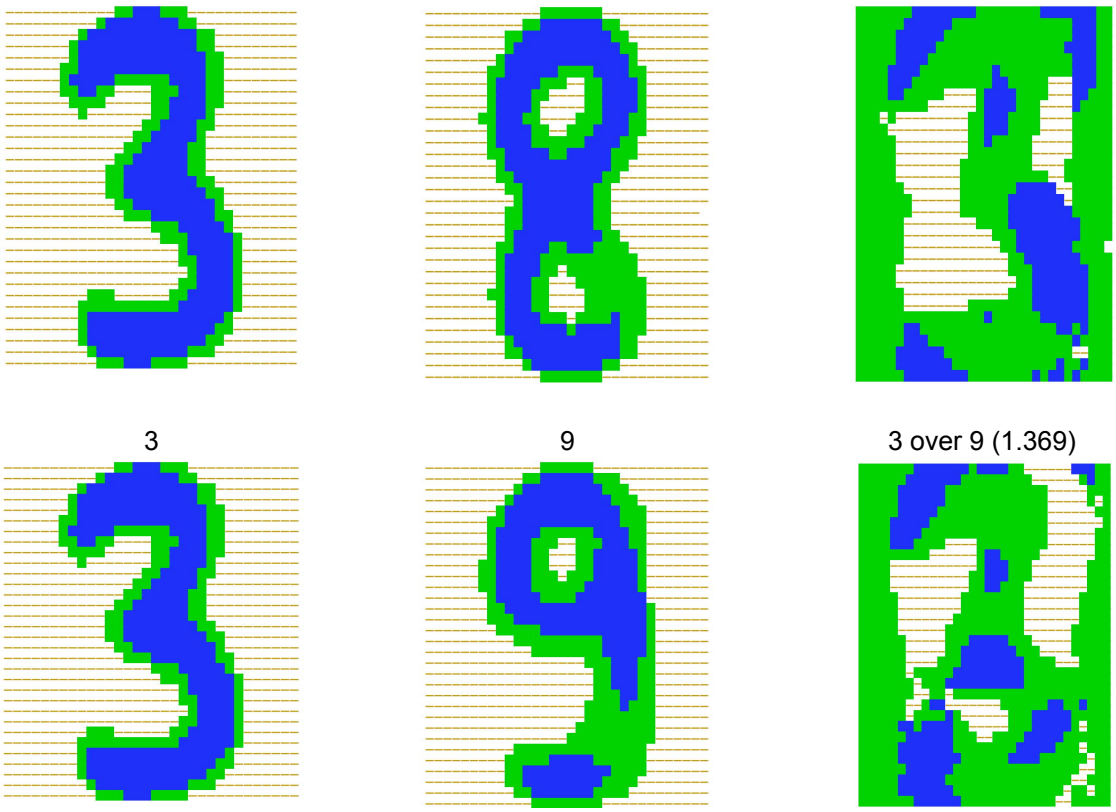
The likelihood and odd ratios maps display below.

For likelihood map, '+' to denote likelihoods **$P(F_{ij} \mid class)$** > 0.667, '~' for likelihoods **$P(F_{ij} \mid class)$** which is between 0.667 and 0.333, and '-' for likelihoods **$P(F_{ij} \mid class)$** < 0.333.

For odd ratios map, '+' to denote features with positive log odds, '~' for features with log odds is between 1 ± 0.4, and '-' for features with negative log odds.

The following four pairs of digit types that have the highest confusion rates: 4 & 9, 3 & 7, 3 & 8 (8 & 3), 3 & 9.

Likelihood	**Likelihood**	**Odd Ratios**
4	9	4 over 9 (avg 7.422)
		
3	7	3 over 7 (avg 13.632)
		
3	8	3 over 8 （avg 1.830）



1.2 Pixel Groups as Features

1.2.1 Algorithm

General Algorithm has been stated above.

Set the typeNum to be 10 (digit 0 to 9), imgHieght and imgWidth to be 32, windowHeight and windowWidth to be x (x in 2, 3, 4), and isOverlap to be true/false.

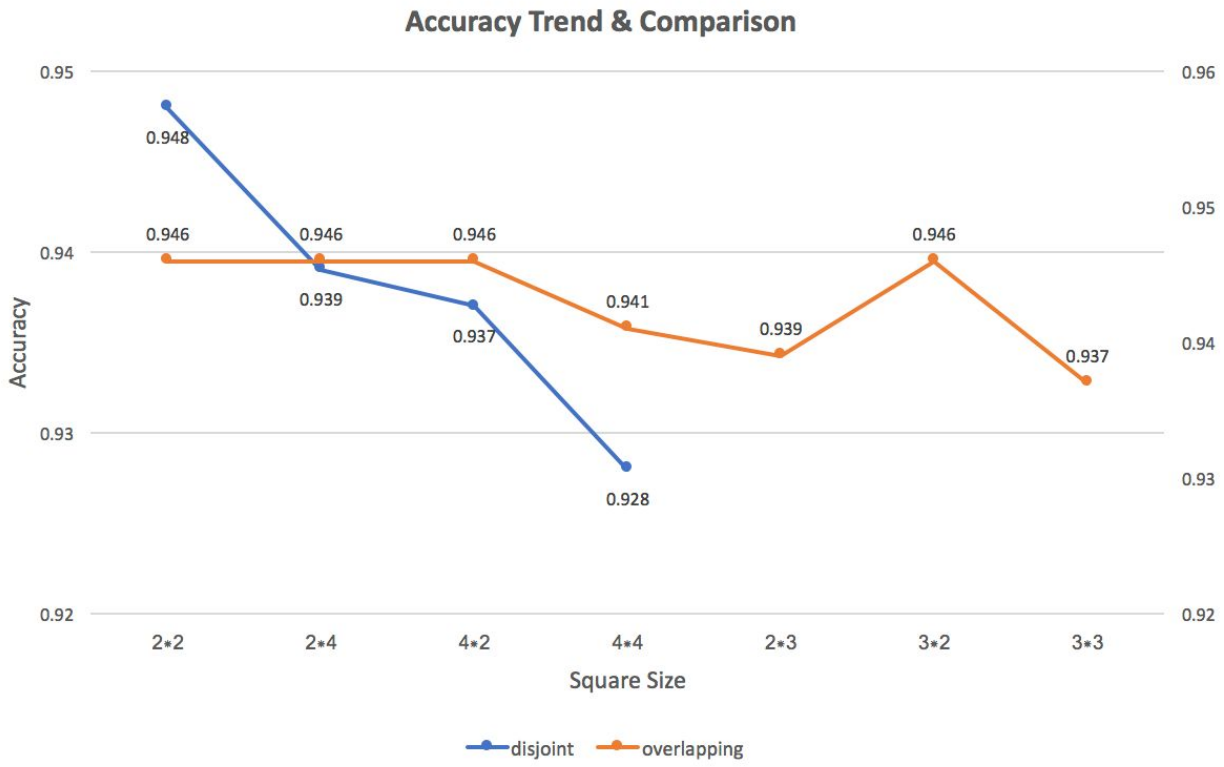
1.2.2 Accuracy & Time Comparison

disjoint patches

square size	2*2	2*4	4*2	4*4
Accuracy	0.948	0.939	0.937	0.928
Training Time (ms)	21	41	23	13

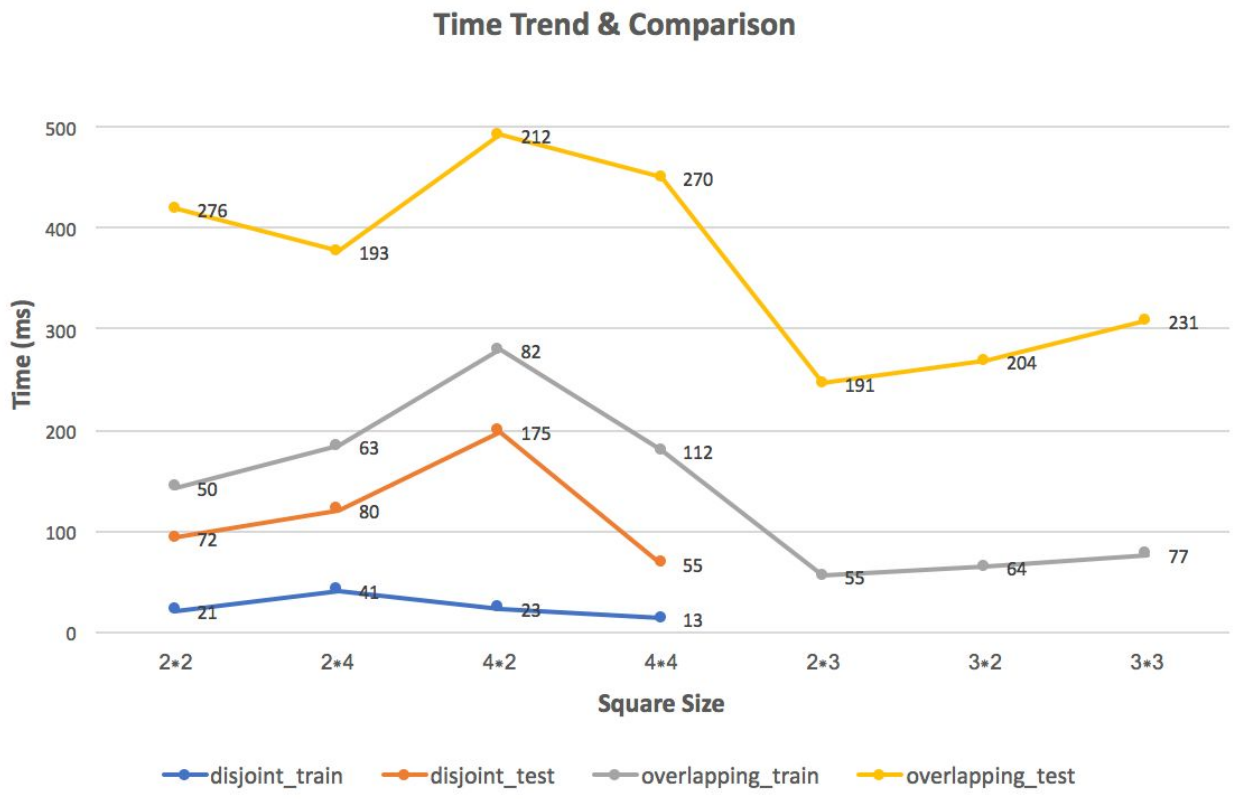
overlapping patches

square size	2*2	2*4	4*2	4*4	2*3	3*2	3*3
Accuracy	0.946	0.946	0.946	0.941	0.939	0.946	0.937
Training Time (ms)	50	63	82	112	55	64	77
Testing Time (ms)	276	193	212	270	191	204	231



As we can see in the above plot, accuracy of classifier with overlapping squares are more stable than the disjoint one, because the former classifier has more data to predict. The relatively rapid decrease of accuracy with disjoint squares is also due to the relatively lack of data.

Due to the same reason, the classifier with disjoint 2*2 squares is the most accurate.



As we can see in the above plot, training and testing time of classifier with overlapping squares are longer than the disjoint one, because the former classifier uses more data to train and predict.

Due to the same reason, the classifier with disjoint 4*4 squares is the fastest.

1.3 Extra: Face Classification

1.3.1 Algorithm

General Algorithm has been stated above.

Set the typeNum to be 2 (is face & is not face), imgHieght to be 70, imgWidth to be 60, windowHeight and windowWidth to be 4, and isOverlap to be false.

1.3.2 Accuracy & Confusion Matrix

Percentage	**predicted**	
Actual	**0 (not face)**	**1 (is face)**

0 (not face)	0.961	0.039
1 (is face)	0	1

➔ Accuracy for each digit ：**diagonal element** on above confusion matrix.

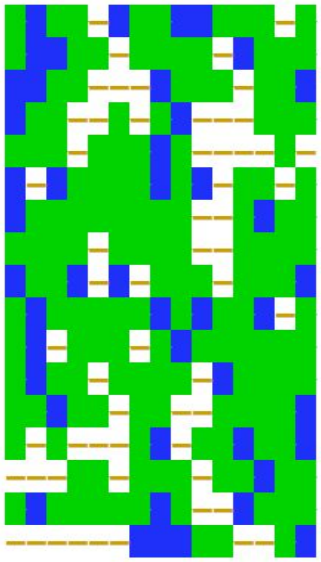
➔ Overall accuracy : **0.98**

➔ Training time: 19 ms

➔ Testing time: 40 ms

1.3.3 Odd Ratios

Below is the odd ratios of 0 (is not face) over 1(is face):



Part 2: Digit Classification using Discriminative Machine Learning Methods

2.1 Digit Classification with Perceptrons

In this part, we are asked to apply a multi-class (non-differentiable) perceptron learning rule to the digits classification problem. The decision rule of multi-class perceptron algorithm is:

$$c = argmax_c W_c \bullet X$$

In this formula, *W* stands for the weight of class c, and *X* represents current image. In other words, it means when we want to decide the predicted label of an image, we should calculate several scores (# of classes) based on each weight, and the class that has highest score should be chosen as predicted class.

2.1.1 Algorithm

Initial:

```
weight ← random/zeros
```

Trainer:

```
for t in echos:
    for i in training data:
        expected ← training label[i]
        max score ← 0
        for w in weight:
            score ← weight · training image[i] + bias
            if score > max score:
                max score ← score
                predicted ← training label[i]

    if expected != predicted:
        weight[expected] ← weight[expected] + learning rate * training image[i]
        weight[predicted] ← weight[predicted] - learning rate * training image[i]

learning rate ← decay function(learning rate)
```

Tester:

```
for i in testing data:
```

```
        expected ← testing label[i]
        max score ← 0
        for w in weight:
            score ← weight · testing image[i] + bias
            if score > max score:
                max score ← score
                predicted ← training label[i]

    confusion matrix[predicted][expected] ← confusion matrix[predicted][expected] + 1
```

2.1.2 Parameter Setting and Overall Accuracy

We have tried several combinations of parameters and set initial epochs to 100. We noticed that when epochs number is 30, the overall accuracy on training dataset is 100%. Therefore we chose 30 as our number of epochs, because the perceptron algorithm find a classifier that can correctly classifies all training data and should be terminated. Based on this epochs we chose the combination of parameters that has the highest overall accuracy on testing dataset. The analysis result is shown in Table 2-1.

Table 2-1

Learning Rate Decay Function	Bias vs. no bias	Initialization of weights (zeros vs. random)	Ordering of training examples (fixed vs. random)	Number of epochs	Overall Accuracy
$\eta = 1/t$, where t is the number of epochs	No Bias	Zeros	fixed	30	0.961711717112
			random		0.858108108108
		random	fixed		0.95045045045
			random		0.948198198198
	Bias	Zeros	fixed		0.945945945946
			random		0.927927927928
		random	fixed		0.936936936937
			random		0.939189189189
$\eta = e^{1/t}$,	No	Zeros	fixed		0.96396396396

where t is the number of epochs	Bias				4
			random		0.957207207207
		random	fixed		0.959459459459
			random		0.961711711712
	Bias	Zeros	fixed		0.959459459459
			random		0.954954954955
		random	fixed		0.968468468468
			random		0.957207207207

Parameters choosing:

- ➔ **Learning Rate Decay Function:** $\eta = e^{1/t}$
- ➔ **Bias vs. no bias:** Bias
- ➔ **Initialization of weights (zeros vs. random):** Random
- ➔ **Ordering of training examples (fixed vs. random):** Fixed
- ➔ **Number of epochs:** 30

The **overall accuracy** based on these parameters is 0.968468468468.

2.1.3 Turning Curve

Based on parameters chosen in 2.1.2, the overall turning curve on training dataset is shown as Fig 2-1.

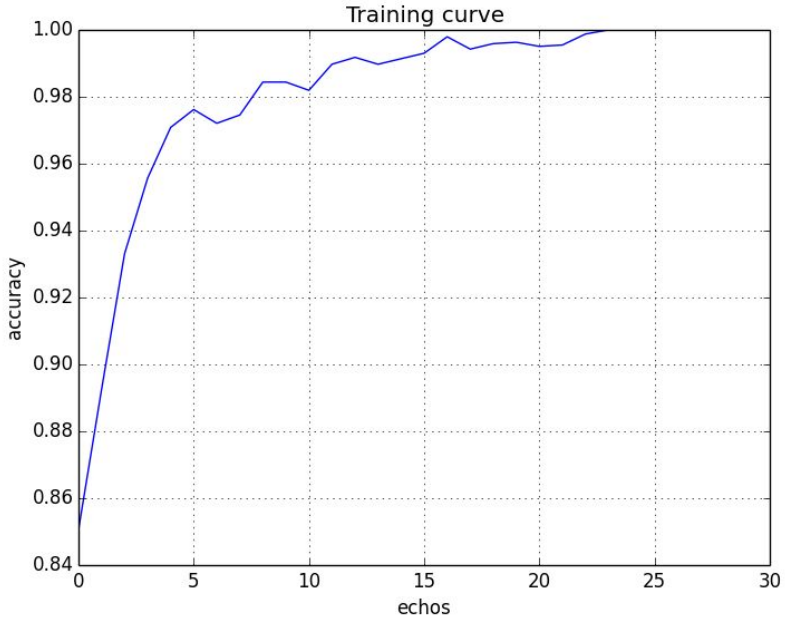


Fig 2-1

2.1.4 Confusion Matrix

Based on parameters chosen in 2.2.2, the confusion matrix is shown as Table 2-2.

Table 2-2

	Actual Label											
Predicted Label		**0**	**1**	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**	**accuracy**
	0	36	0	0	0	0	0	1	0	0	1	0.95
	1	0	43	0	0	0	0	0	0	0	0	1.0
	2	0	0	35	0	0	0	0	0	0	0	1.0
	3	0	0	0	33	0	0	0	0	0	0	1.0
	4	0	0	0	0	55	0	0	0	0	0	1.0
	5	0	0	1	0	0	58	0	0	1	1	0.95
	6	0	0	0	0	0	0	42	0	0	0	1.0
	7	0	2	0	0	0	0	0	47	0	1	0.94
	8	0	0	5	0	2	0	0	0	39	1	0.83
	9	0	0	0	0	2	0	0	0	0	38	0.95

	total	36	45	41	33	59	58	43	47	40	42	0.968

2.2 Digit Classification with Nearest Neighbor

In this part, we used brute force K nearest neighbor algorithm as the classifier. This is a very simple algorithm but has even higher accuracy than multi-class perceptron, here we implemented the brute force KNN algorithm and also discussed KD-tree and Ball-tree algorithms.

2.2.1 Algorithm

Tester:

```
for i in testing data:
    expected ← testing label[i]
    for j in training data:
        distance ← distance function(testing image[i], training image[j])
    for label in k max distance:
        predicted ← argmax(count(label))
```

2.2.2 Distance/Similarity Function

There are several ways to calculate distance/similarity of two features, here we chose three most commonly used methods.

a. Cosine Distance

Cosine Similarity is always be used for information retrieval and text analysis, for the documents can be represented by a bag or long vector, with each attribute recording the frequency of a particular term (referenced by CS412 slide 2 page 67). Here, in order to be compatible with other methods, we use 1 minus cosine similarity as cosine distance.

$$1 - \frac{u \cdot v}{||u||_2 ||v||_2}.$$

b. Correlation Distance

Correlation distance is often used to check correlation of numerical data, it is sensitive to the scale of data. Here, in order to be compatible with other methods, we use 1 minus correlation similarity as correlation distance (referenced by CS412 slide 3 page 21).

$$1 - \frac{(u - \bar{u}) \cdot (v - \bar{v})}{||(u - \bar{u})||_2 ||(v - \bar{v})||_2}$$

c. Euclidean Distance

Euclidean distance is the simplest distance, it is a kind of Minkowski distance which h in the following formula equals to 2. It always be used to calculate maximum distance between two vectors (referenced by CS412 slide 2 page 60).

$$d(i, j) = \sqrt[h]{|x_{i1} - x_{j1}|^h + |x_{i2} - x_{j2}|^h + \cdots + |x_{ip} - x_{jp}|^h}$$

After simplification, the formula seems like:

$$\left(\sum (w_i |(u_i - v_i)|^2)\right)^{1/2}$$

2.2.3 Accuracy as function of K

We tested the accuracy of KNN based on three distance function discussed in 2.2.2. The results are shown in Table 2-3 and Fig 2-2. From these results we observed that when *K* is smaller than 10, the KNN algorithm with cosine distance has the highest accuracy compared with other two distance methods, and the algorithm with euclidean distance has the lowest accuracy. But when *K* grows larger, the euclidean method become much competitive. However, the trends of overall accuracy on three methods are declining when *K* increase.

Table 2-3

K	Accuracy		
	Cosine	**Correlation**	**Euclidean**
1	1.0	0.997747747748	0.995495495495
2	0.995495495495	0.988738738739	0.986486486486

3	1.0	0.997747747748	0.993243243243
4	0.995495495495	0.990990990991	0.988738738739
5	0.997747747748	0.995495495495	0.993243243243
6	0.995495495495	0.990990990991	0.988738738739
7	0.997747747748	0.993243243243	0.988738738739
8	0.993243243243	0.988738738739	0.988738738739
9	0.988738738739	0.990990990991	0.988738738739
10	0.986486486486	0.984234234234	0.988738738739
11	0.986486486486	0.988738738739	0.990990990991
12	0.986486486486	0.986486486486	0.990990990991
13	0.984234234234	0.988738738739	0.990990990991
14	0.984234234234	0.986486486486	0.988738738739
15	0.984234234234	0.984234234234	0.988738738739
16	0.981981981982	0.981981981982	0.988738738739
17	0.984234234234	0.981981981982	0.986486486486
18	0.977477477477	0.981981981982	0.986486486486
19	0.984234234234	0.984234234234	0.988738738739
20	0.977477477477	0.977477477477	0.981981981982
21	0.97972972973	0.981981981982	0.981981981982
22	0.977477477477	0.981981981982	0.981981981982
23	0.977477477477	0.981981981982	0.97972972973
24	0.981981981982	0.984234234234	0.981981981982
25	0.97972972973	0.97972972973	0.97972972973

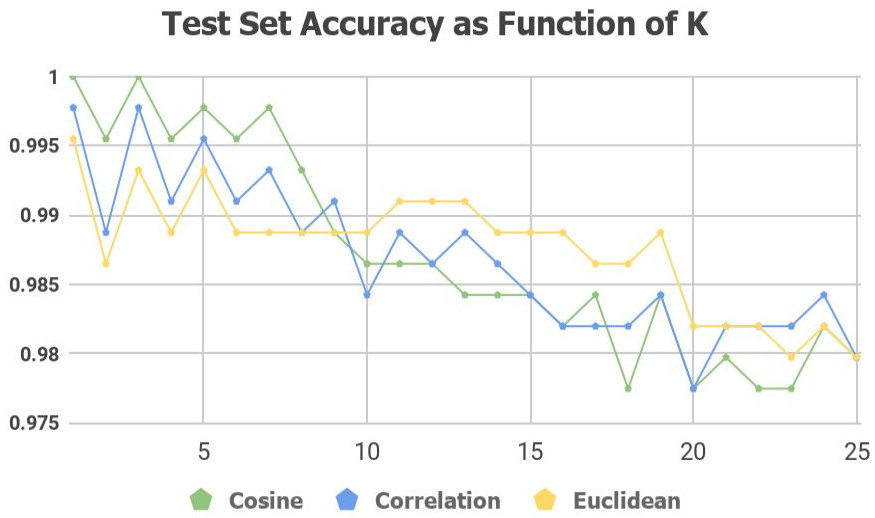


Fig 2-2

Parameters choosing:

- ➔ **K:** 3
- ➔ **Similarity/Distance Function:** Cosine Similarity

2.2.4 Confusion Matrix

Based on parameters chosen in 2.2.3, the confusion matrix is shown as Table 2-4.

Table 2-4

	Actual Label											
Predicted Label		**0**	**1**	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**	**accuracy**
	0	36	0	0	0	0	0	0	0	0	0	1.0
	1	0	45	0	0	0	0	0	0	0	0	1.0
	2	0	0	41	0	0	0	0	0	0	0	1.0
	3	0	0	0	33	0	0	0	0	0	0	1.0
	4	0	0	0	0	59	0	0	0	0	0	1.0
	5	0	0	0	0	0	58	0	0	0	0	1.0
	6	0	0	0	0	0	0	43	0	0	0	1.0
	7	0	0	0	0	0	0	0	47	0	0	1.0

	8	0	0	0	0	0	0	0	0	40	0	1.0
	9	0	0	0	0	0	0	0	0	0	42	1.0
	total	36	45	41	33	59	58	43	47	40	42	1.0

2.2.5 Brute Force Algorithm and Optimization

2.2.5.1 Running time for single query

➔ **time:** 0.074514s/image

2.2.5.2 Optimization

In order to accelerate the brute force algorithm, one method is to reduce searching space. We can split data space into several tiny subspaces and only considered points in those spaces that are closed to the target.

For example, if we want to find out ten Japanese restaurants that closest to UIUC, we don't need to traverse all Japanese restaurants in the United States and calculate their distances with our school. Instead, we can split the data space (in this case the United State) into small subspaces (in this case city) and search from the subspaces (in this case Urbana and Champaign) that nearest to the target (in this case UIUC).

The problem here is how to split space to optimize the performance to the utmost extent. K-D Tree and Ball Tree are two different methods. **K-D Tree** algorithm makes a median cut each time on the k dimension, however, when the # of dimension grows higher than 20 the performance of K-D Tree would decrease rapidly. In order to fix this issue, **Ball Tree** split space into hyperspaces.

2.2.5.3 Evaluation

We implement KD-Tree (from library scipy.spatial.ckdtree) and Ball Tree (from library sklearn.neighbors.BallTree) in our code , and compare the building and testing time with brute force search. The results are shown in table 2-5:

Table 2-5

	Brute Force Search	**KD-Tree**	**BallTree**
Building Time	0s	0.311748s	0.372873s

Testing Time	0.074514s/image	0.00357s/image	0.003345s/image

From this table, we can easily observed that although KD-Tree and BallTree algorithms need to spend sometime building the tree, but the testing complexity is highly reduced.

2.2.6 Comparison with Naive Bayes and perceptron

The accuracy based on Naive Bayes, Perceptron and KDD are shown in table 2-6.

Table 2-6

	Overall Algorithm		
	Naive Bayes	**Perceptron**	**KDD**
Accuracy	0.939	0.968	1.0

2.3 Extra Credit

2.3.1 Learned Perceptron Weights Visualization

2.3.1.1 Visualization

We implemented the heap map visualization (shown in Fig 2-3) as follow. The color of each pixel stands for the weight value at this pixel.

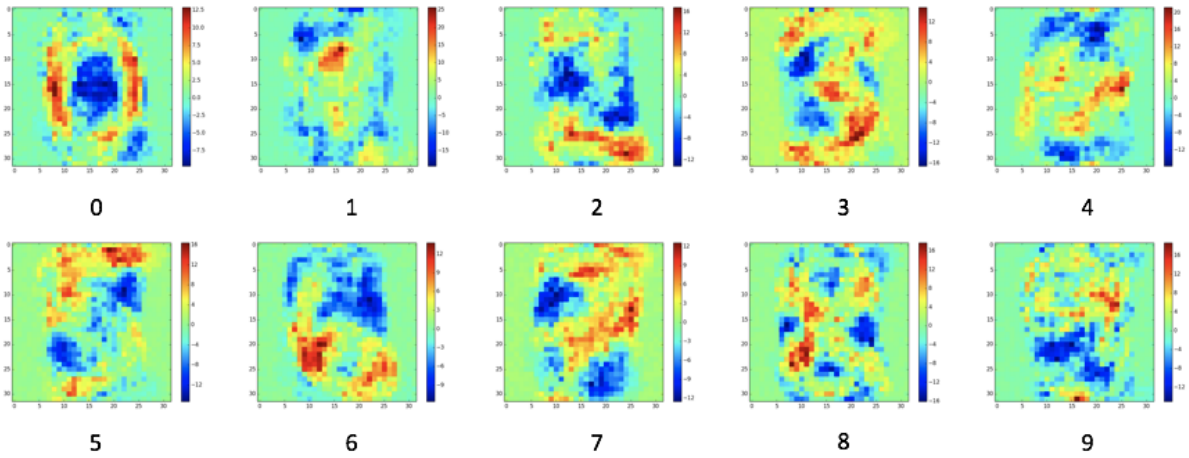


Fig 2-3

2.3.1.2 Analysis

In our visualization, a pixel filled by red color means the weight is positive, and blue vice versa. A positive weight means if the pixel value on this position is “1”, it will contribute highly to the final score of this weight. The green color means these positions are relatively not related to deciding the label of this digit.

Let’s take digit 0 as an example. Compared with other digits, the left and right margin of digit 0 is one of it’s symbol because no other digits has this property. Therefore, the color of pixels in two sides are relatively warm. Also, in the middle of 0, the pixels’ color tends to be blue, which means if there are some “1” in the middle, this digit has low possibility to be considered as 0. The green color around digit 0 means these positions does not contribute to the decision of this digit.

2.3.2 Differentiable Perceptron Learning Rule

The multi-class differentiable perceptron is similar like what we implemented in part 2.1, the only difference is the decision rule:

$$c = softmax_c W_c \bullet X,$$

$$P(c \mid x) = \frac{exp(W_c \bullet X)}{\sum_{k=1}^{c} exp(W_k \bullet X)}$$

2.3.2.1 Algorithm

Initial:

```
weight ← random/zeros
```

Trainer:

```
for t in echos:
    for i in training data:
        expected ← training label[i]
        max score ← 0
        for w in weight:
            denominator ← denominator + math.exp(weight · training image[i] + bias)
        for w in weight:
            numerator ← math.exp(weight · training image[i] + bias)
            score ← numerator / denominator
            if score > max score:
                max score ← score
                predicted ← training label[i]

     if expected != predicted:
         weight[expected] ← weight[expected] + learning rate * training image[i]
         weight[predicted] ← weight[predicted] - learning rate * training image[i]

    learning rate ← decay function(learning rate)
```

Tester:

```
    for i in testing data:
        expected ← testing label[i]
        max score ← 0
        for w in weight:
            denominator ← denominator + math.exp(weight · testing image[i] + bias)
        for w in weight:
            numerator ← math.exp(weight · testing image[i] + bias)
            score ← numerator / denominator
            if score > max score:
                max score ← score
                predicted ← testing label[i]

    confusion matrix[predicted][expected] ← confusion matrix[predicted][expected] + 1
```

2.3.2.2 Overall Accuracy and Confusion Matrix

➔ **Overall Accuracy:** 0.93018018018

Table 2-7

	Actual Label

Predicted Label		0	1	2	3	4	5	6	7	8	9	Accuracy
	0	36	0	0	0	0	0	0	0	0	0	1.0
	1	0	41	0	0	0	0	0	0	0	0	1.0
	2	0	0	39	0	0	0	0	0	0	0	1.0
	3	0	2	0	30	0	0	0	3	1	1	0.81
	4	0	0	0	0	49	0	1	1	0	0	0.96
	5	0	0	0	0	0	58	0	0	1	1	0.97
	6	0	0	0	0	0	0	42	0	0	0	1.0
	7	0	1	0	0	0	0	0	42	0	0	0.98
	8	0	0	1	0	2	0	0	0	36	0	0.92
	9	0	1	1	3	8	0	0	1	2	40	0.71
	total	36	45	41	33	59	58	43	47	40	42	0.93

2.3.3 Other Classifiers

In this part, we implemented a decision tree (from library sklearn.tree.DecisionTreeClassifier) and calculated it's confusion matrix. Decision Tree is one of the very basic classifier that with a top-down, recursive, divide-and-conquer process. In each level, we should select the attribute that contribute most to the final decision, and separate data points based on the value of this attribute (referenced from CS412 slide 8, P9-26). In our case, the value of attribute is always 2 (i.e. "0" and "1"), that is to say, this decision tree is a binary tree. There are several methods for choosing appropriate attribute: information

gain, which may bias towards multivalued attributes; Gain ratio, which tend to prefer unbalanced splits; and Gini index, which may has difficulty when # of classes is large.

2.3.3.1 Algorithm

Here we provide a decision tree building algorithm based on gini index attribute measure method:

Decision Tree Builder (data set D):

for attribute in attribute set:
D_0 ← all data in D that attribute = 0
D_1 ← all data in D that attribute = 1
gini(D_0) ← 1 - $\sum_{j=1}^{n} p_j^2$, where p_j is the relative frequency of class j in D_0
gini(D_1) ← 1 - $\sum_{j=1}^{n} p_j^2$, where p_j is the relative frequency of class j in D_1
gini(attribute) ← $\frac{|D_0|}{D}$ gini(D_0) + $\frac{|D_1|}{D}$ gini(D_1)

split_attribute ← argmax (gini(attribute))
D_0 , D_1 ← split D on split_attribute

Decision Tree Builder (D_0)
Decision Tree Builder (D_1)

(referenced from CS412)

2.3.2.2 Overall Accuracy and Confusion Matrix

➔ **Overall Accuracy:** 0.862612612613

Table 2-7

	Actual Label											
Predicted Label		**0**	**1**	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**	**Accuracy**
	0	34	1	0	0	0	0	1	0	1	0	0.92
	1	0	36	1	1	1	0	1	0	1	0	0.88
	2	1	1	34	2	0	1	1	1	1	1	0.79

	3	0	0	1	27	0	2	0	0	0	0	0.9
	4	1	2	0	0	54	1	1	1	0	0	0.9
	5	0	0	1	2	0	45	0	1	1	1	0.88
	6	0	1	1	0	1	1	39	0	0	0	0.91
	7	0	0	0	0	0	3	0	43	0	0	0.93
	8	0	3	2	0	3	1	0	1	33	2	0.73
	9	0	1	1	1	0	4	0	0	3	38	0.79
	total	36	45	41	33	59	58	43	47	40	42	0.86