Report of Question 2

Zekun Zhang (zz364)

zekunzhang.1996@gmail.com

a)

When I firstly read the instruction of this question, I was such confused that because there's almost no obvious patterns I can find. So, the first thing I need to do is to find a way to represent the 'strange' images and pre-process the data.

As far as I'm concerned, there are 5 images for different label and 10 in total. Due to the number limit of images, it's really hard to avoid the overfitting problems. Moreover, the resolution for every single image is 5 * 5 pixels, and I can find that for every single pixel, there are only black ones and white ones. So, I choose to use 0 to represent white ones and 1 for the black ones. Then the whole image would be represented as a 2-dimensional array.

After I have data like that, for the first model, I would like to choose a clustering algorithm, because the question converts to that how to label a 2-dimensional array which only contains 0s and 1s. There are lots of unsupervised clustering algorithm, but it's improper for us to choose an unsupervised algorithm. Because in my view, there's no need for me to use a method using SVM or regression based on this tiny dataset. So, I would like to choose a supervised clustering. Thus, I selected the K-Nearest Neighbors algorithm (KNN).

In this part, I'm trying to use KNN to calculate the distances between all the data I have and select a parameter k that defines the threshold to determine which class should be the prediction. For instance, If I assume $k = 3$, what I'm going to do is to take the closest three labels into consideration and the majority as the prediction. About the detail implementation which is really easy because the main idea of KNN only needs few steps to finish.

For the implementation part:

1. The most first thing is to read matrix from text files where I save the matrix information for Class A, Class B and Mystery.

2. Because the size of dataset is larger than the size of target images, I make a copy for 10 times to match up the size of our dataset.

3. Then I calculate the difference by using the array subtraction and get difference in every single pixel.

4. Next, sum up every single square of the difference of the pixel and make a square root operation on the result. At this point, I've already got Euclidean distances between the target and my dataset on every single image.

5. Last but not least, sort the distances and take K results from the start of the list (because I sort the list by ascend order).

I test several cases by given different K, because my dataset only contains 10 different sets. So, I test the input K from 1 to 9 to compare the different outputs.

Table 1

K	Result
1	B, A, B, A, B
3	B, A, B, A, B
5	A, A, B, A, A
7	B, A, B, A, A
9	A, A, B, A, A

As we all know that when K is 9 which means only one is not considered in the 10 images, so, the result is meaningless sometimes. And also, I can't make an even K as an input, because there would be sometimes a tie which can't give a specify result. So, the results I can take to analyze should be the first four. As we can see from the Table 1, for the target images, the second, third and the fourth would be seen as well classified. But for the first and the last one, there are different results due to the different inputs of K. When K is 1 and 3, the prediction result for the last target is B, but it changes to A when K is 5, 7 and 9. I could estimate that there are several specific training sets (B) very close to the target. So, I prefer B as the final prediction result for the last one. As for the first, it's really hard to say from the results showed. In my view, I prefer to set K to near 3 or 5.

The result of the prediction seems to be close and the algorithm really makes sense, but it's really difficult for human beings to understand the inner relation between the dataset and the target images. As far as I'm concerned, that's one of the disadvantages of KNN algorithm. Since the KNN only compares the differences (Euclidean distances here) between the dataset and target image. But it's impossible for human beings to analyze the inner difference only with a huge amount of data. In my view, it's really hard for us to consider the result as a rational one.

b)

Due to the size of the data provided, there must be overfitting problems. Actually, it's a really hard question for me. Because the overfitting problem in KNN is really hard to avoid. The best way I can considered is to increase the size of data provided, but of course, in this case, it's impossible. So, the only other way may be changing the K values to decrease the influence of overfitting. As we all know that the K is related with the number of samples you have and the noise on these samples. For instance, in this case, if I choose K is equal to 1, I would have overfitting easily, even without any noise. So, I need to increase the K slightly and try to have a balance between overfitting and huge influence by noise. In other projects, when I'm going to use KNN, I would start with $K = log(number\ of\ samples)$, and I would increase K depending on the level of noise in my samples.

c)

In the second model, I'm trying to use the Naive Bayes Classifier to achieve the classification work. Just as the first one with KNN, I preprocess the data in the same way and store them into text files. Then the main idea is to compute the possibility of being black for every single pixel. So, if I introduce x_{ij} to represent the value of the pixel at (i, j), then I can have equations like that:

$$P(x_{ij}|Class = A) = \frac{\sum_{k=1}^{5} P(x_{ij}^{(k)}, Class = A)}{\sum_{k=1}^{5} P(Class = A)}$$

$$P(x_{ij}|Class = B) = \frac{\sum_{k=1}^{5} P(x_{ij}^{(k)}, Class = B)}{\sum_{k=1}^{5} P(Class = B)}$$

Then I would use the function above to assign possibility to every single value. Then I could get a possibility map. The whole process above could be seen as a pretraining process. Then, put the target image into the model (the possibility map). Then compare every single pixel with the possibility map and sum up all the possibility together to get the possibility $P(Like\ A)\ and\ P(Like\ B)$.

Then, as we can see from Table 2, the KNN and the Naive Bayes classifier have almost the same output "B, A, B, A, B". So, I can make a conclusion that both these two algorithms have a pretty good performance even the algorithms are such simple ones.

Table 2

Image	Class A	Class B	Result
1	0.0002286236854138088	0.0008573388203017832	B
2	0.00020322105370116337	4.7629934461210166e-06	A
3	2.3814967230605083e-06	0.0025720164609053494	B
4	0.0004572473708276176	4.762993446121017e-05	A
5	0.0005144032921810699	0.0020576131687242796	B