Question 3.

The following part (1) and (2) respectively describe KNN model and Perceptron model, training, results and ~~any~~ analysis

(1) In this part, we use KNN as our classification model. It's basic idea is to achieve classification by finding the closest neighbor. Below is the detailed steps of our model.

- Input: we are given a ~~set~~ bunch of 5×5 pictures. In order to make the pictures ~~computable~~ more easily computable, we ~~[illegible]~~ extract every row of the picture and reform rows into a row-vector. We use '-1' to represent 'black block' and '1' to represent 'white block'.

- steps: for every picture in unlabeled set:
    - Compute the Euclidean Distance to class A and class B;
    - Sort ~~the~~ all distance to find the k closest neighbors;
    - find the class with highest frequence among thoes k neighbors, let's say A has highest frequence;
    - Take A as the classification.

[code of Question3_1 and log_1.txt can be found in Question3's Folder]

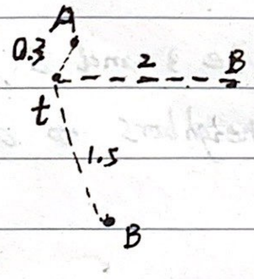
- Result; we ~~take~~ respectively set k to be 3 and 5, which means we choose 3 and 5 nearest neighbors to do classification. results are as follows:

| | $C_1$ | $C_2$ | $C_3$ | $C_4$ | $C_5$ | |
|---|---|---|---|---|---|---|
| k=3 | B | A | A | B | B | - classfication |
| | $B_3B_4A_5$ | $B_3A_5A_3$ | $A_1A_2B_2$ | $B_2B_3A_2$ | $A_3B_2B_1$ | - neighbors |
| k=5 | B | B | A | A | B | |
| | $B_3B_4A_5B_1A_1$ | $B_3A_5A_3B_1B_4$ | $A_1A_2B_2A_3B_1$ | $B_2B_3A_2A_4A_5$ | $A_3B_2B_1B_5A_1$ | |

- Some Analysis:

① In KNN classification, in order to keep away from overfitting, ~~the~~ we need to choose ~~a good~~ a relatively large value of 'k'. If 'k' is too small, let's say k=1, there is a chance that the neighbor is a 'noise points' which lead to a wrong classification. However, 'k' could not be too big as well. let's say there is a small cluster $a_1$ which contains 50 points. While you take ~~100~~ 150 as the value of k. When you take 150 neighbors, the neighbors may contain more than 50 ~~of~~ points of $a_2$ cluster, which also leads to a wrong classification.

② In ~~y~~our implement of KNN algorithm, we only take the number of neighbors into account, regardless of the distance value. ~~Let~~ And we can do improvement in this part. Let's look at a small example.



[ t represents target ]

The graph shows ~~the~~ a ~~possibility~~ possible situation of 'k=3'. As we can see, t is really close to $\overset{a}{}$point of class A, at the distance of 0.3, while the distance of t to points in B are 5 times larger, so in this case, are we confident to say t belongs to B?
In this case, we can use the ~~distanced-weighted~~ distance-weighed strategy. The comparison of number of neighbors is transformed into comparison of forms of distance. The closer the points are, the more important they are in classification:

$V(A) = 1 \times \frac{1}{0.3} = \frac{10}{3}$

$V(B) = 1 \times \frac{1}{2} + 1 \times \frac{1}{1.5} = \frac{7}{6}$

$V(A) > V(B)$. Then we classify 't' into class A.

(2) In this part, we choose Perceptron as our classification model. ~~The reason~~ It's a classic model used to classify data into two parts. Below is the detailed ~~steps~~ steps of this model.

- Input: like what we did in part (1), we transform the picture into a row-vector to convenience the following computation. And we use '-1' to represent 'black ~~bot~~ block' and '1' to represent 'white block'. ~~Finally~~ Then we label pictures in A as 0, pictures in B as 1

- ~~steps~~ model:

  input space: $\underline{x_i} = R^{25}$ outspace: $f(x) \in R^1$ label $\in \{0, 1\}$

  model: $f(\underline{x}) = \begin{cases} 1 & \text{if } \underline{w} \cdot \underline{x} \geq 0 \\ 0 & \text{else} \end{cases}$

  Error: error $(f(x), \text{label}) = \text{abs}(f(\underline{x}) - \text{label})$

  update: $\underline{W}(k+1) = \underline{W}(k) - \alpha[f_{\underline{w}}(\underline{x}) - \text{label}] \cdot \underline{X}$

- steps:
  - from training set, randomly pick one as input.
    generate random weights ~~W=R^25~~ in 0-1: $W = R^{25}$
  - Repeat until a relatively big number of steps.
    ```
    | compute f(x)
    | compute error
    | Compute update process for weights.
    ```
  - From ~~test set~~ unlabeled set, pick each one in order, feed it to the perceptron, and get f(x)

```
| if f(x) = 1.   label it as 'class B'.
| else           label it as 'class A'
```

[code of Question 3_2 and log_2.txt can be found in Question3's folder]

- Results: we feed each picture ~~of~~ from unlabeled set, and get the results as follows:

| $C_1$ | $C_2$ | $C_3$ | $C_4$ | $C_5$ |
|---|---|---|---|---|
| B | A | A | B | A |

- Some thoughts:

① Perceptron is only applicable when data is linear-seperable. If not, the learning process will continuously shift without ~~conver~~ convergence. In this situation, we need to add hidden layer in order to 'draw curve' to seperate data, which is refered to as neural network; While KNN algo is not limitted by such situation. KNN is always applicable whether or not the data is ~~lea~~ linear applicable. It only care about choosing good neighbors.

② Improvement could be made in this case by add bias ~~into~~ as a variable. Bias gives the 'seperating line' ability to shift, which will lead to better ~~se~~ results when the data ~~is~~ are more ~~dif~~ complex (still needs to be linear-seperable)