Question3:

I programmed in python3 to implement the model. And I denoted the black pixel as 1 and white pixel as 0 in a 5 * 5 numpy array to represent each picture.

Class

A	B
0	1

a)

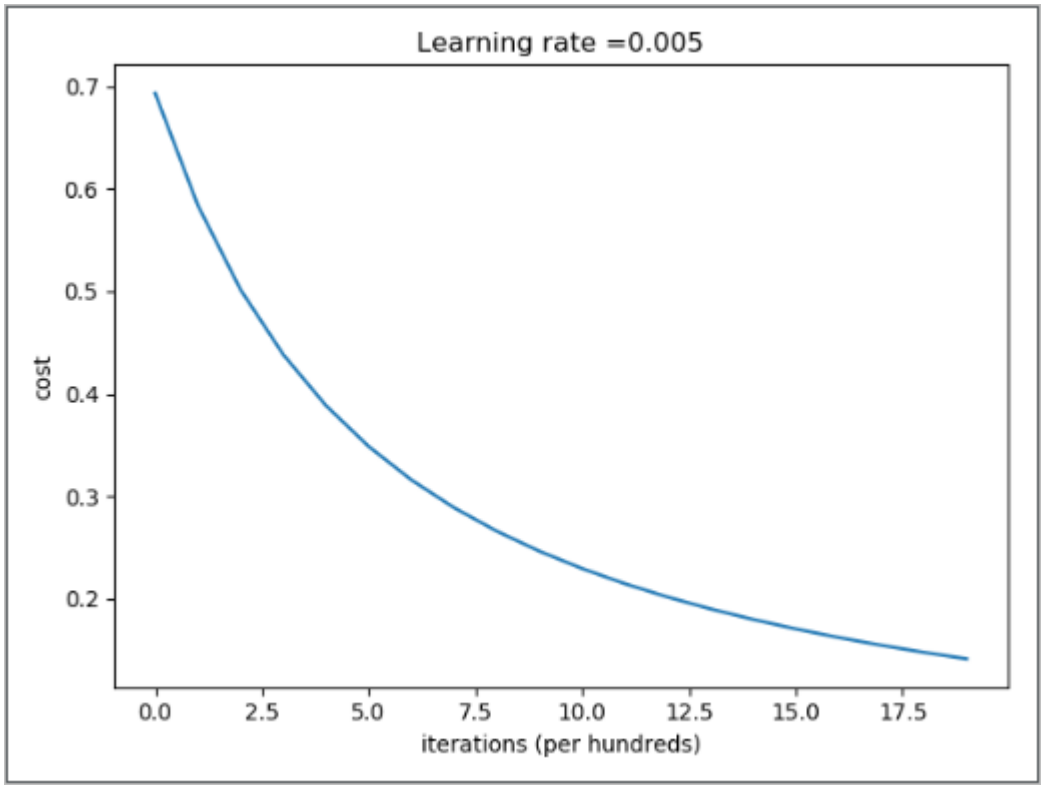
I construct a logistic regression for this part and most of the idea about how to construct the model I have written in the comment in the code logistic_regression.py. And which I want to add is that I use all the 10 AB pictures to train the model and chose 2 of them as the test set, so the train and test accuracy can't be used as a scale for how successful our model is.

The core of this model is the sigmoid function, but I don't want to talk too much about it because it seems that I have copied a lot from the wiki even though I didn't. So, most of the explanation I want to say is in the comment in the python code. And the parameters such as learning rate and number of iterations are set as usual, mostly learned them from the website.

The result of mystery and the learning curve is below.

```
train accuracy: 100.0 %
test accuracy: 100.0 %
[[1.  1.  0.  1.  0.]]
```

PREDICT: [B, B, A, B, A]



b)

The data is small so obviously there would be huge overfitting, that is the reason why I choose a simple model. Because is we use a model like neural network, it will much more match the train set which could cause larger overfitting and the model will not be useful in the real use when input other data.

So, I introduce some methods to expand the training set.

1. As a picture, the most recent method for editing it without losing information is to do the rotation or reflection, because it won't change the content of the picture but for computer they are different picture. So, we can use rotation or reflection to expand the set.
2. Another method is flipping. Most of the time flipping doesn't change the content of a picture but there is some exception. For example, for the traffic sign turn right, if we flip it, we will get a sign states turn left, that will totally change the meaning of the picture. But in this problem, the flipping does work.
3. There is also a useful method to augmenting the data called Cross-validation. It is any of various similar model validation techniques for assessing how the results of a statistical analysis will generalize to an independent data set. It is mainly used in settings where the goal is prediction, and one wants to estimate how accurately a predictive model will perform in practice. In a prediction problem, a model is usually given a dataset of known data on which training is run (training dataset), and a dataset of unknown data (or first seen data) against which the model is tested (called the validation dataset or testing set). The goal of cross-validation is to test the model's ability to predict new data that was not used in estimating it, to flag problems like overfitting or selection bias and to give an insight on how the model will generalize to an independent dataset (i.e., an unknown dataset, for instance from a real problem).

   In our problem, we have 10 samples for training, for one iteration we can randomly choose 2 of the 10 as the test set and remaining 8 of the 10 as training set, and we can repeat this step to update weights if and only if any two iterations have different training set and test set.

   In summary, cross-validation combines (averages) measures of fitness in prediction to derive a more accurate estimate of model prediction performance.
4. There are other methods like random crop, scale, shift, normalization and so on. But it may not match our problem, so I don't want to talk too much about them.

c)

I do the same strategy to divide the train set and test set and load the data as in a). And because A and B is each have 5 samples, I think that I just choose k up to 5. Because if k is larger than 5 there is both A and B in the kth nearest points. Because of that I just think k up to the biggest possible number that can make all kth nearest points have a chance that they are in the same class.

Knn is an algorithm that when we input the unknown samples, we compute the distance with the points in the train set, and choose the kth nearest points, and choose the most often label in those kth points as the label of the unknown sample. The common methods we use to compute distance are Euclidean Distance and Manhattan Distance.

Other details are basically in the comments of the code knn.py. The results are below.

```
k = 1
Correct 2/2: The test accuracy: 1.000000
[1. 1. 0. 1. 0.]
k = 2
Correct 2/2: The test accuracy: 1.000000
[1. 0. 0. 1. 0.]
k = 3
Correct 2/2: The test accuracy: 1.000000
[1. 1. 0. 1. 1.]
k = 4
Correct 2/2: The test accuracy: 1.000000
[1. 0. 0. 0. 1.]
k = 5
Correct 2/2: The test accuracy: 1.000000
[1. 1. 0. 0. 1.]
```

We easily know that the smallest k is, the more errors or overfitting occur, because there may be a chance that the unknown sample match with some special points or a noise points.

Difference: LR use a sigmoid function to fit the samples, knn just calculate the distance (difference) between samples. So, usually for small number of samples, knn would be more likely to get overfitting than LR. For example, we have some samples, best model is in the middle graph, LR is the left and knn is the right graph below. Because comparing to the LR, knn is more likely to be influenced by outlier or noise.