CS 520 Final Question 3 Solution

Guo Chen

December 2018

1 Solutions

1.1 Question a)

Solution:

In this part, as suggested by Prof. Cowan in his recent notification posted on Sakai, I utilized k Nearest Neighbors (kNN) algorithm to classify the unlabeled images. kNN algorithm is designed to classify a group of points and try to classify all of them into various categories. To do so, first of all, the algorithm takes a parameter k, which is often defined by the programmer and indicates at most how many possible neighbors close to a query point are considered as classification reference, then it takes some points $x_r, r = 1, ..., k$ labeled as $y_r, r = 1, ..., k$ as the training set and a unlabeled point (or a series of unlabeled points, here I merely take one point as an example) denoted as x'. Then it takes a look at x' and checks all of its neighbor points in a certain distance. If majority (over half) of these neighbor points are classified into a certain category denoted as y_a, then the query point is classified as y_a too.

Since I implemented the whole process with function knn in R, so before demonstrate the effect of running the algorithm to classify those unlabeled images, I would like to give a brief introduction to the inner mechanism of the function, which is based on implementation in its source code, and go back to the question afterwards. In the algorithm, all euclidean distances between the labeled points $x_{1:n}$ and the unlabeled point x' are calculated and all of them are arranged in an ascending order (from small to large), later then k labeled points are selected from this list based on the distance from smallest to smaller. At this time, the categories of these k point would be checked and a category to which the majority (over half) of them belong would be set as the one to the unlabeled point x'.

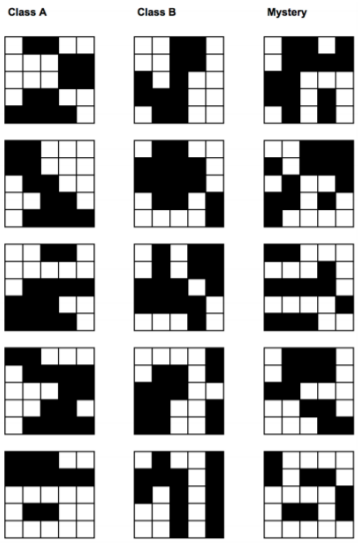
Now let's go back to the question and see how to process these images as inputs. In this question, 10 differently labeled (as class A or B) images are given and another 5 images need to be labeled. To deal with this problem, in order to apply kNN algorithm, all labeled images should be transformed into a set of point. Considering that each image is composed of 5 * 5 cells in white or black, an image can be regarded as a set of 25 cells where a white cell is denoted as value 0 and a black cell is denoted as 1. In this way a training set can be built.

At this point, kNN algorithm can take such training set as input and start to build its own classification model.

After running the kNN algorithm with different parameter k on all the images (of course, the number of given images is quite small and can cause overfitting problem easily, but I will put my solution to this issue in the question b)), a series of results are obtained and shown as below. From here it appears obviously that the best value for k is 3, when the model has the best performance while it can also keep simple.

Results	
Parameter k	Classification results (A means being in class A, B means being in class B)
2	B, B, A, B, B
3	B, B, A, A, B
4	B, B, A, A, B
5	B, B, A, A, B

In my opinion, I think this prediction makes sense to me. As shown in the picture below, intuitively, I notice that for images in class A, at least one white horizontal bar (composed of at least two consecutive cells) is longer than all vertical white bars (perhaps the second image from top in class A is an exception), while the situation is on the contrary in class B (the third image from top in class B may be an exception, too). Now when looking at those images categorized as "Mystery", you would find out that the first two images have the same feature of those in class A while the next two images have the same feature of those in class B. And for the final one, I think it has both features of those in class A and those in class B, but it makes sense to be classified to be class B.



1.2 Question b)

Solution:

Based on the situation of the question, I can think of three ways to avoid such an overfitting problem:

1. Introduce more training data.
2. Use a simpler model instead.
3. Cross validation technique.

Since I am given a limited number of images for training so far and there is no other ways to get more data, what comes to my mind firstly is to generate "new" images from given images. Specifically, based on the features of these images, I randomly pick up two images from each category of class A and B and make any two of the following data augmentation techniques:

1. exchange positions of a black cell and a white cell randomly;
2. randomly choose a white cell and make it as a black cell;
3. randomly choose a black cell and make it as a white cell.

By doing so, the total number of images in my training set increases by 4. And they works fine as the results shown in the last question.

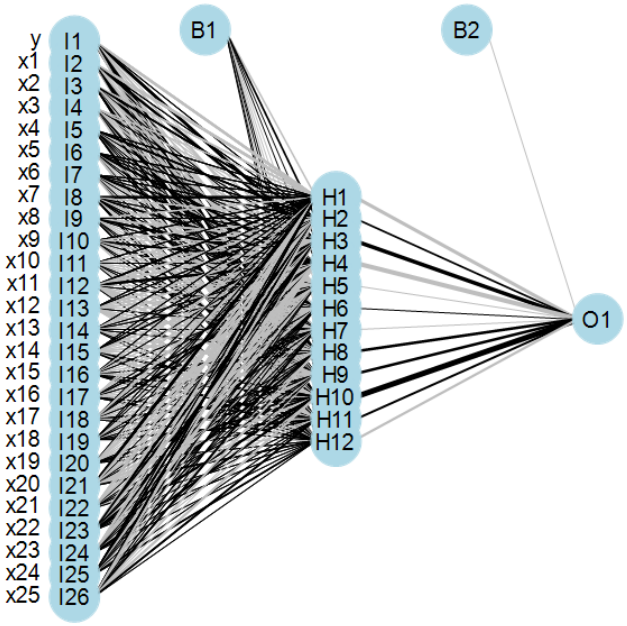
And for the next question, which requires me to construct and train a second type of model, I build a neural network with one hidden layer and use the two methods to avoid overfitting: introduce more training data and use cross validation technique. The process of introducing more data is the same as described above. And as for using cross validation technique, since I now have 14 groups of data sets in 2 different categories, I use 7-fold cross validation in the training process. Specifically, I wrote a while loop and in each iteration, two groups of data are selected from the 2 categories separately and act as a test set while the rest groups remain as the training sets. And each time as

the while loop moves around, all data in the two categories act as test sets in turn while the others remain as the training data still. These two methods work quite fine for me and the experiment results will be shown in solution to the next question.

1.3 Question c)

Solution:

In this part, I tried to build a neural network to classify those unlabeled images. Specifically, with the help of function *nnet* in R, I build a neural network with 26 inputs (corresponding to the already known category of an image plus 25 cells in the image), 1 output and only **1 hidden layer**. As for the activation function for each neuron in the network (including hidden layer and output), I set its activation function as **sigmoid function**. The whole structure of the neural network is shown below, which is drawn with the help of function *plot.nnet* in R:



Since I used *nnet* function in R to build a neural network, here I will give a brief introduction of how it trains the whole network in detail based on implementations in its source code. In order to train this neural network, backpropagation method is applied.

First of all, loss function is defined as the square of the difference between actual output denoted as y and theoretical output denoted as $f_\omega(x)$, where ω

represents all weights in the neural network, that is, $Loss(x, y) = \|f_\omega(x) - y\|^2 = \sum_{output\ k}(out_k - y_k)^2$, where k is one output neuron. At this point, a Stochastic Descent Gradient technique is applied to calculate how each weight parameter in the neural network should change to accommodate the loss. Specifically, first of all, for each output neuron k, the modified error is computed as $\Delta_k = (out_k - y_k)g'(in_k)$. Then for each prior neuron j that connects neuron k, the weight of its connection between k and j, denoted as $\omega_{j,k}(t+1)$, is updated in such a way: $\omega_{j,k}(t+1) = \omega_{j,k}(t) - \alpha\Delta_j out_i$. So far the loss at the output has been "pushed" backwards to prior nodes. And such a process would be repeated until all the weights in the neural network are updated.

When building the neural network, I tried different number of neurons for the hidden layer and the classification performance in terms of that is shown in the following form. After a series of experiments, I find out that **12 is the most proper number of the hidden layer neurons**, because it is the smallest number which can be used to build a neural network that performs the same effect of the previous kNN algorithm, the last image is classified in a different category though. However, based on the analysis in question a), since the last image has the features of both class A and B, it is acceptable if it is classified in class B as well.

Results	
Number of neurons in hidden layer	Classification results (A means being in class A, B means being in class B)
2	B, B, B, A, A
4	B, B, B, A, A
6	A, B, A, A, A
8	A, B, A, A, A
10	A, B, A, A, A
12	B, B, A, A, A
16	B, B, A, A, A
32	B, B, A, A, A

I will make a comparison between kNN alogrithm in question a) and neural network in question c) here. Here we can clearly see that both models have made classifications and performed the same effects in the end, with different parameters set. However, as for the interpretability of models, which is now a popular trend in AI research, kNN algorithm appears better than neural network in that at least it has potential ability to explain how the classification results come to be while neural network looks like a black box, because it is not that intuitively easy to see why the exact final outputs are generated from inputs.