Q2. Name: Xinghao Wang NetId: xw354

(a) I apply SVM model to this classification problem.

1) Input: Since each image is a 5×5 grid of black or white cells, I represent it as a 5×5 matrix with elements 0 (white) and 1 (black). However, the input of SVM must be a vector, so I flatten the matrix into a 1×25 vector.

2) Output: According to the problem, I need to classify an image into class A or class B. So, the output is a value that I can use to classify the image.

3) Dataset: The dataset for training is all the images of class A and class B. The size of it is 10, including 5 images of A and 5 images of B. Since SVM is a supervised learning method and this is a binary classification problem, I label each image of class A as "1" and each image of B as "-1". Therefore, the inputs of training process is a list of 10 tuples. The first element of each tuple is a 1×25 vector representation of an image and the second element is the label (1 or -1) of the image. The Test data set is 5 unlabeled images. Each image is to be classified after the training process.

4) Loss Function: I choose the quadratic loss function.

$$Loss = \sum_{i=1}^{n} (y_i - \phi(\underline{x}^i))^2$$

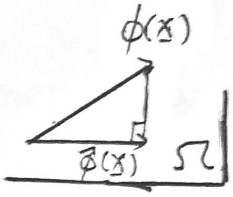
The reason I choose it is I want to consider the cost of classification errors instead of the size of margin. In other words, for class A ($y=1$), I want $\phi(\underline{x}) > 0$ and for class B ($y=-1$), I want $\phi(\underline{x}) < 0$. Thus, I use $(y_i - \phi(\underline{x}^i))^2$ to measure the error.

But how to represent $\phi(\underline{x}^i)$? I use $\hat{\phi}(\underline{x}^i) = \alpha_1 K(\underline{x}^i, \underline{x}^1) + \cdots + \alpha_n K(\underline{x}^i, \underline{x}^n)$ to represent it. The reason is as follows.

Assume $\phi(\underline{x}) \in V$ and $\Omega = \{\alpha_1 K(\underline{x}, \underline{x}^1) + \alpha_2 K(\underline{x}, \underline{x}^2) + \cdots + \alpha_n K(\underline{x}, \underline{x}^n)\}$ is a subspace of V. $\hat{\phi}(\underline{x}) \in \Omega$, so $\hat{\phi}(\underline{x})$ can be represented as:

$$\hat{\phi}(\underline{x}) = \alpha_1 K(\underline{x}, \underline{x}^1) + \alpha_2 K(\underline{x}, \underline{x}^2) + \cdots + \alpha_n K(\underline{x}, \underline{x}^n)$$

It's easy to know that when $\hat{\phi}(\underline{x})$ is the map of $\phi(\underline{x})$ in Ω, $\hat{\phi}(\underline{x})$ is the best representation of $\phi(\underline{x})$.



$$\therefore \phi(\underline{x}) - \hat{\phi}(\underline{x}) \perp K(\underline{x}, \underline{x}^i), \forall i \in \{1, \ldots, n\}$$

$$\therefore \langle \phi(\underline{x}) - \hat{\phi}(\underline{x}), K(\underline{x}, \underline{x}^i) \rangle = 0$$

$$\Rightarrow \langle \phi(\underline{x}), K(\underline{x}, \underline{x}^i) \rangle = \langle \hat{\phi}(\underline{x}), K(\underline{x}, \underline{x}^i) \rangle$$

$$\Rightarrow \phi(\underline{x}^i) = \hat{\phi}(\underline{x}^i) = \sum_{j=1}^{n} \alpha_j K(\underline{x}^i, \underline{x}^j)$$

Then the goal is to minimize the loss function as follows.

$$\min_{\alpha_1, \ldots, \alpha_n} \sum_{i=1}^{n} (y_i - \sum_{j=1}^{n} \alpha_j K(\underline{x}^i, \underline{x}^j))^2$$

Notice that the optimization is without constraints.

5) Kernel Function: The kernel I choose is Gaussian kernel.

$$K(\underline{x}^i, \underline{x}^j) = \exp(-\frac{1}{h} \cdot \|\underline{x}^i - \underline{x}^j\|^2)$$

The reason is that theoretically, it can be used to fit any non-linear problem.

6) Result: After training the model, I can simply substitute $\underline{x}^i$ into $\hat{\phi}(\underline{x}^i)$ and see whether $\hat{\phi}(\underline{x}^i)$ is bigger than 0. If $\hat{\phi}(\underline{x}^i) > 0$, then classify $\underline{x}^i$ as class A, else classify it as class B.
Name the unlabled images in the original order as C_1, C_2, C_3, C_4, C_5.
The result is $C_1 \to B$, $C_2 \to A$, $C_3 \to B$, $C_4 \to A$, $C_5 \to B$. The result is reasonable because I find that most of black cells of class A appear in the upper left part and most of black cells of class B appear in the lower right part.

(b) In order to avoid overfitting, I add a regularization into the objective function. So now the objective function is as follows.

$$\min_{\alpha_1,\dots,\alpha_n} \sum_{i=1}^{n} \left(y_i - \sum_{j=1}^{n} \alpha_j K(\underline{x}^i, \underline{x}^j)\right)^2 + \lambda \cdot \|\vec{\beta}(x)\|^2$$

By using the regularization term, the model is more likely to generate simple parameters. The reason I choose L_2 regularization is that it has derivatives everywhere, which is a good mathematical property. The other way to avoid overfitting is to assign a high value to h in the kernel function.

(c) I also apply K-Nearest Neighbor model to this problem. The input is same as previous model. However, the output is no longer a value. It is a list containing labels of the K nearest neighbors. In this model, we don't need a loss function or training process. Just compute the distances between the unlabeled point and any other labeled points. Pick K neareast neighbors and count the number of each class, then find which one is bigger and finally decide the classification.

The result is: $C_1 \to A$, $C_2 \to A$, $C_3 \to B$, $C_4 \to A$, $C_5 \to A$ when K is 5. Obviously C_1 and C_5 are different from the prediction of SVM. However, I think the result makes sense because black cells of C_1 and C_5 are kind of evenly distributed on the map. The difference I think is from the uncertainty of values of K. I changed K to 3, then it has the exactly same result of SVM.