Zhuohang Li(zl299)

CS 520 Final: Question 3 - Classification

a) 1. *Representation:* I use a set of one-dimensional feature vectors to represent the images. Each vector has 25 features, corresponding to the 25 cells in a image. Use 0 for white cells, and 1 for black cells.
2. *Model:* I choose to use a logistic regression model to classify these images for the following reasons: 1) Logistic regression is good for binary classification problems. 2) The logistic regression model is simple and has few parameters, so it will still work for a small sample size.
3. *Implementation:* Learning parameter θ is a vector with dimension = number of features + 1, the extra coefficient represents the weights for the constant term, which is the bias. The hypothesis function of a given input is:

$$h_\theta(x) = g(\theta^T x) = \frac{1}{1 + e^{-\theta^T x}}$$

where $g(z) = \frac{1}{1+e^{-z}}$ is the sigmoid (logistic) function.
Since the images are represented in one-dimensional vector, and in '0's and '1's, linear decision boundary will be sufficient for my model, which looks like this:

$$h_\theta(x) = g(\theta_0 + \theta_1 x_1 + \theta_2 \theta x_2 + \cdots + \theta_n x_n)$$

where n is the number of features.
I choose to use a convex cost function to avoid being stuck into local minimums while trying to apply the gradient-descent algorithm to find the minimum cost:

$$J(\theta) = -\frac{1}{m}[\sum_{i=1}^{m} \log h_\theta\left(x^{(i)}\right) + (1 - y^{(i)})\log(1 - h_\theta\left(x^{(i)}\right))]$$

where m is the number of training examples.
To avoid overfitting, add a regularization term to the cost function:

$$J(\theta) = -\frac{1}{m}[\sum_{i=1}^{m} \log h_\theta\left(x^{(i)}\right) + (1 - y^{(i)})\log(1 - h_\theta\left(x^{(i)}\right))] + \frac{\lambda}{2m}\sum_{j=1}^{n} \theta_j^2$$

where λ is the regularization parameter. Noticed that we don't penalize the constant term.
Then compute the gradient for each θ_j to update the coefficients:

$$\theta_0 = \theta_0 - \alpha \frac{1}{m}\sum_{i=1}^{m} (h_\theta\left(x^{(i)}\right) - y^{(i)}) x_0^{(i)}$$

For j = 1, 2, …, n:

$$\theta_j = \theta_j - \alpha[\frac{1}{m}\sum_{i=1}^{m}(h_\theta(x^{(i)}) - y^{(i)})x_j^{(i)} + \frac{\lambda}{m}\theta_j]$$

Repeat for each iteration until the training is over.

To make a prediction about a given new x, apply the hypothesis to the x:

$$output = h_\theta(x) = \frac{1}{1 + e^{-\theta^T x}}$$

4. *Result:* The prediction result for the mystery dataset is: (from top to bottom) BBABA

For me, the images of class A are somewhat 'horizontal', and the images of class B are somewhat 'vertical'. Therefore, I would mark the mystery dataset from top to bottom as: BBAA, and the last one has no clear preference neither to A nor to B. So, we only disagree with the fourth image. The overall performance is acceptable.

b) As is mentioned in Question a), I used a regularized cost function for this model. Add a regularization term will prevent overfitting by restricting the coefficients at a small and reasonable value. Below is what θ looks like after training: [-0.050523320852894266, -0.13969902835130815, -0.044375244273006126, -0.04352408601193919, 0.04654935227220488, 0.13657290860698, 0.0031906505918611544, 0.04662933369797389, 0.09411708888295103, 0.004267500143227804, -0.0042771696440179925]
Another thing we can do is to stop early. In this case, I stop the training process just after the accuracy of the training set has turned 1 to prevent overfitting.

c) 1. *Model:* For the second implementation, I choose to use the k-Nearest-Neighbors algorithm. The reasons are: 1) kNN is a lazy learning algorithm. The training process do not involve complicated computations, all it needs to do is to simply keep the training data. Therefore, it can be implemented easily.2) It works well on small datasets as a classifier, especially for this problem where there are only two classes and 5 samples for each class.
2. *Implementation:* The algorithm is straight-forward. First, pick a distance function. There are many ways to compute distance, such as Euclidean distance, Manhattan distance, Hamming distance, which can all be generalized as Minkowski distance. But for this problem, since I'm representing those images with '1's and '0's, all those distance functions shall produce the same result. In the training part, just keep all the training data in a list. When making a prediction, take the new input, compute the distance from it to all data in the training set. Take the first k nearest labeled

data to vote, if most of which is mark as A, then the prediction result is A, otherwise B.
3. *Result:* For k=5, the model predicts the mystery data set as BBAAB, which is consistent with my views.
4. *Comparison:* The logistic regression marked the fourth mystery image as class B, while kNN marked it as class A. I think the reason is that the logical regression algorithm is trying to extract the common features out of each class and make predictions based on all the training data, while kNN is simply comparing the new image to each of the input images, and try to find which one looks alike, or, which k labeled images look most like the new image. There is no general good or bad regarding to the performance of these two algorithms, but in this specific case with a small data set, kNN performs better.