Problem 1

Question 1

Notice that the risk is:

$$R(\alpha_i|x) = \sum_j \lambda(\alpha_i|c_j)P(c_j|x)$$

Substituting 0-1 loss function we get:

$$R(\alpha_i|x) = \sum_{j \neq i} P(c_j|x) = 1 - P(c_i|x)$$

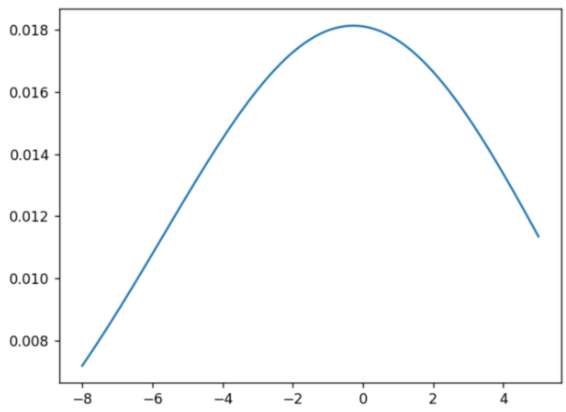
We want to minimize the risk, which means to maximize the probability $P(c_i|x)$. Notice that:

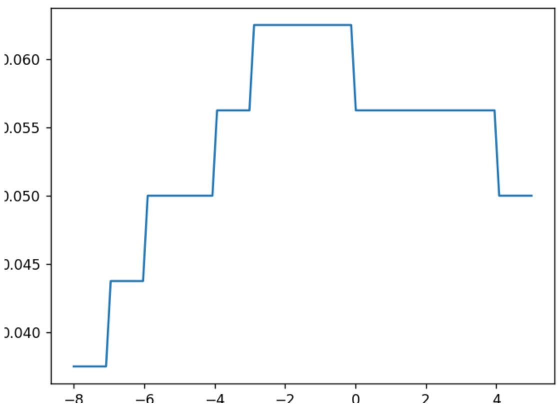
$$f_\varphi(x) = \frac{1}{n}\sum \frac{1}{h^d}\varphi\left(\frac{x - x_i}{h}\right)$$

Notice that when $y_i \notin c$, $\varphi\left(\frac{x-x_i}{h}\right) = 0$, and therefore we're simply counting the number of votes of each class, and choosing the one with the maximal votes.

Question 2

Here's a drawing of the probability estimation:





We assume the distribution is gaussian. Here are the MLE estimators:

$$\mu_{MLE} = \frac{1}{n}\Sigma x = -0.6$$

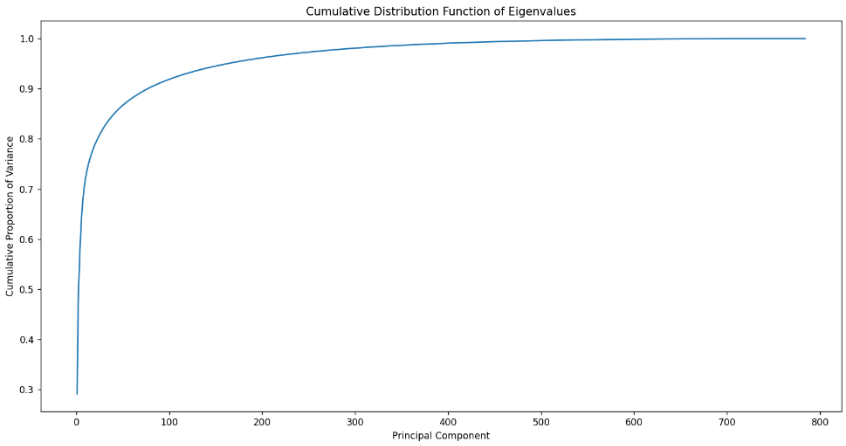
$$\sigma^2_{MLE} = \frac{1}{n-1}(x - \mu)^2 = 13.64$$

Problem 2

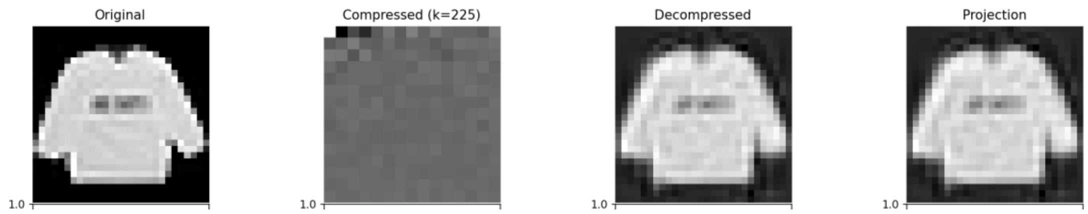
We built a function which loads data given a path, and then returns samples and labels separately as numpy arrays. We plotted a picture using the function plot_picture to ensure everything is going well.

Next, we defined PCA class which is initialized with the new dimension ($\sqrt{k}$), and the dataset. We found the eigenvalues and eigenvectors of the scatter matrix, and saved the k vectors with the k greatest eigenvalues in a matrix E. We defined a vectorized function that compresses each sample by multiplying it by the matrix E. In addition, we wrote two functions: one that calculates the projection (in dimension d), and one that decompresses images (back to dimension d).

Now, we plot the CDF of eigenvalues:



As observed in the picture, we can reduce to dimension 225 almost without losing data (0.975 CPV) - great! Now we plot a few pictures

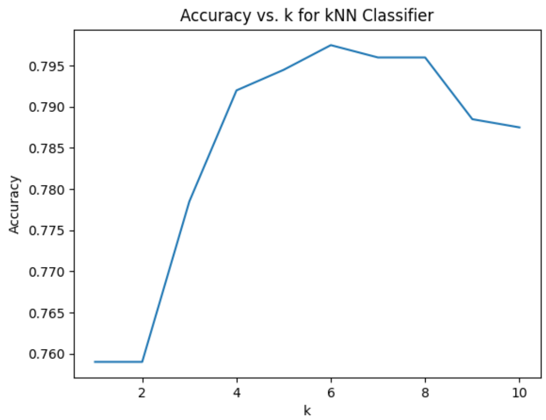


Now we are ready to run kNN. We defined a class, and its constructor, which receives training set, and calculates the square norms of them all (for a futuristic trick). Then, we defined a function "classify". This function gets a sample, and calculates its distance from all points, then does the vote within k closest samples. Notice that calculating the distance for each point would take 550 operations (subtraction and dot product), so instead we decided to find

$$\min_{y}(x-y)^2 = \min_{y} x^2 + y^2 - 2xy = \min_{y} 0.5y^2 - xy$$

And so we can use the square norms we preprocessed (y^2) and do 226 operations per sample instead (dot product 255, subtraction of scalars 1), and this way we have saved half of our time.

We then defined a function 'test' which takes the trained model and a compressed test-set, and calculates its accuracy. To find the optimal k, we ran over a small sample of test set and training set, and got the following graph:



The optimal k is 6, and so we chose it.

The final accuracy is 81.6 %, and running time is 140 seconds (about 2 minutes).

Problem 3

We will be doing logistic regression. For that purpose, we define a class which has 3 attributes: learning rate (etta), convergence constant, and LDF vector (w).

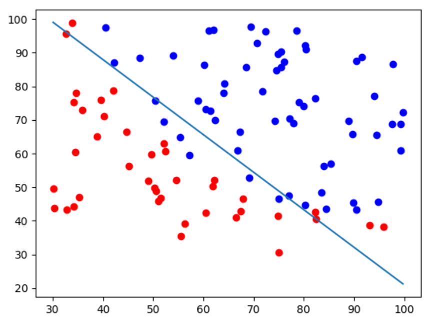
Then, we load the data from the CSV file, and split it into 90-10 train-test sets in the function train. Then, we iterate 28,000 iterations (or until convergance) with gradient descent. This gives us a value for w, which we use to test our model with the function test, which uses a vectorized function of the function classify, then calculates the mean. This whole thing is done within the scope of the function 'evaluate', which tests the model epoch_num times, and returns the average accuracy rate.

I split the train set into validation and training set (0.9-0.1), and ran over different values. I found accuracy rate as a function of learning rate running until convergance/until finishing 28,000 iterations, and plotted it (it took too much time and I forgot to take a picture so I won't be showing a picture)

The maximal accuracy is 90%, which can be accomplished for different values. The program returned 6.35, and so it is the learning rate in the code.

<u>Average running time</u> is 40 seconds.

<u>Average accuracy rate</u> is 90%. Data is not linearly separable. Best classifier for this dataset is some special kernel with SVM, although it is reasonably almost linearly separable. Here's the best separating line:



Problem 4

We run for each dataset.

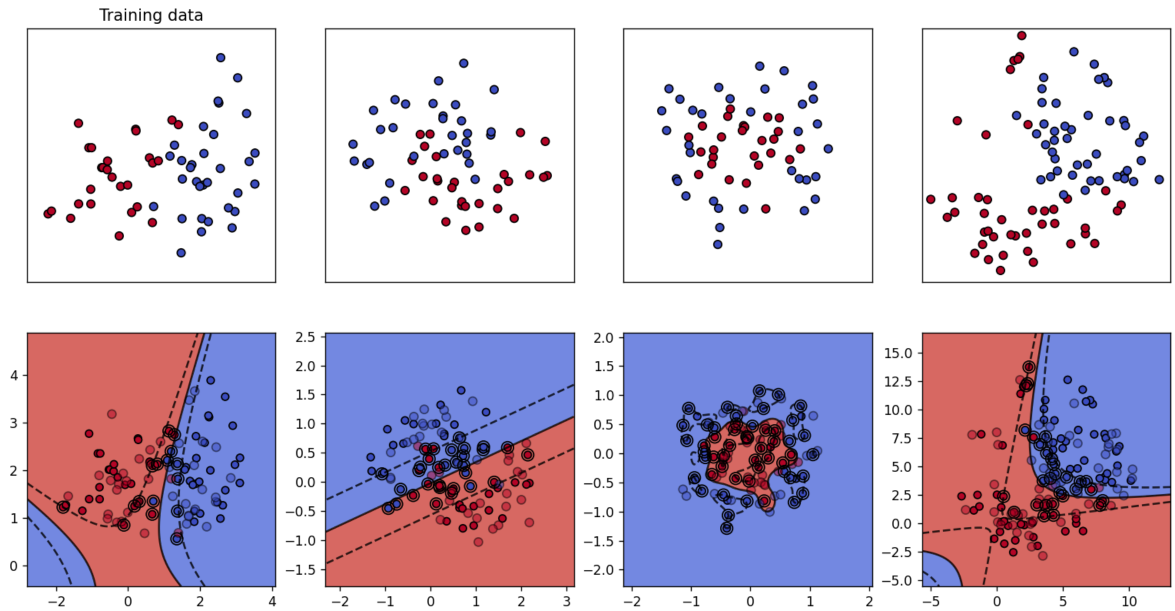
The first one is obviously not RBF(not radical), but could be separated by linear or polynomial kernels with 95% accuracy. The arguments are: $C = 0.6, K(x, y) = (1 + x^T y)^2$.

The second one is almost linearly separable, and so the best kernel is linear with 90% accuracy. The arguments are: $C = 1, K(x, y) = x^T y$.

The third one looks radical, so using RBF is reasonable, and gives 92.5% accuracy, which is good. The arguments are: $C = 0.9, \gamma = 15, K(x, y) = e^{-\gamma||x-y||^2}$

Last but not least, the forth one, obviously not linear, or radical, so by trying polynomial kernel we can get to 96.55% with the parameters: $C = 0.4, K(x, y) = (1 + x^T y)^2$.

Here are all 4 separations aligned in one picture:



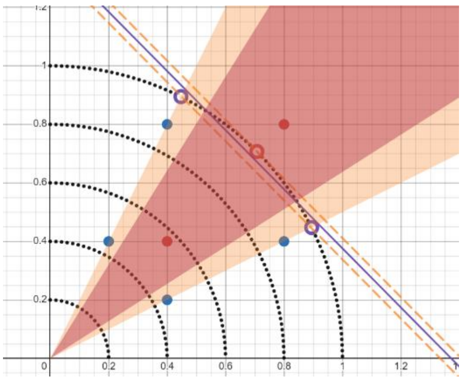
Problem 5

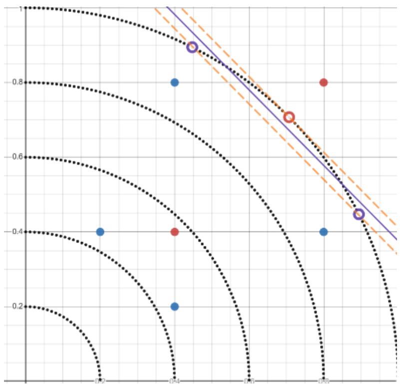
1) This new kernel maps points $(rcos\theta, rsin\theta)$ to points $(\cos\theta\,, sin\theta)$, and therefore we get the following mapping function:

$$\varphi(x) = \frac{x}{||x||_2}$$

$$K(x,y) = \varphi^T(x)\varphi(y) = \frac{x^T y}{||x||_2 ||y||_2}$$

2) In the following figure you could notice the mapping (empty points)
3) the separating line found by trial (blue line/read area borders), and the margin(shaded area/dashed lines). Notice that our separating line decides red for points between the intersection points, and blue for all the rest, which gives us the angular decision boundary:





4) Now, our data space is:

$$\mathcal{X} = \left\{\left(\cos\left(\frac{\pi}{4}\right), \sin\left(\frac{\pi}{4}\right)\right), (\cos(0.463)\,, \sin(0.463)), (\cos(1.107)\,, \sin(1.107))\right\}$$

With a simple substitution we get that blue is 1, red is -1. Now we substitute in:

$$\sum_{i=1}^{3} \alpha_i x_i y_i = w, \qquad \sum_{i=1}^{3} \alpha_i y_i = 0$$

We obtain:

$$\left(\begin{array}{ccc|c} 0.707 & -0.894 & -0.447 & -1 \\ 0.707 & -0.446 & -0.894 & -1 \\ 1 & -1 & -1 & 0 \end{array}\right)$$

Which is solved by:

$$\alpha = \begin{pmatrix} -27.21 \\ -13.59 \\ -13.62 \end{pmatrix}$$

5) The non-linear hyperplane is given by:

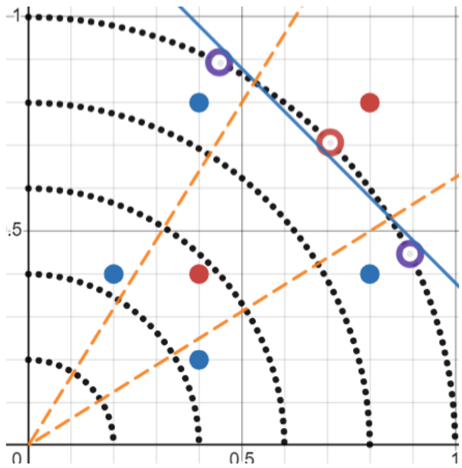
$$\sum_{i=1}^{3} \alpha_i y_i K(x, x_i) + b = 0$$

$$-27.21 \cdot \begin{pmatrix} 0.707 \\ 0.707 \end{pmatrix}^T \frac{x}{||x||_2} + 13.59 \cdot \begin{pmatrix} 0.894 \\ 0.446 \end{pmatrix}^T \frac{x}{||x||_2} + 13.62 \cdot \begin{pmatrix} 0.446 \\ 0.894 \end{pmatrix}^T \frac{x}{||x||_2} + 1.378 = 0$$

Which unsurprisingly simplifies to:

$$\frac{||x||_1}{||x||_2} = \frac{x + y}{\sqrt{x^2 + y^2}} = 1.378$$

Which is the dashed line in the following drawing:



For all equations used, click on the fish.