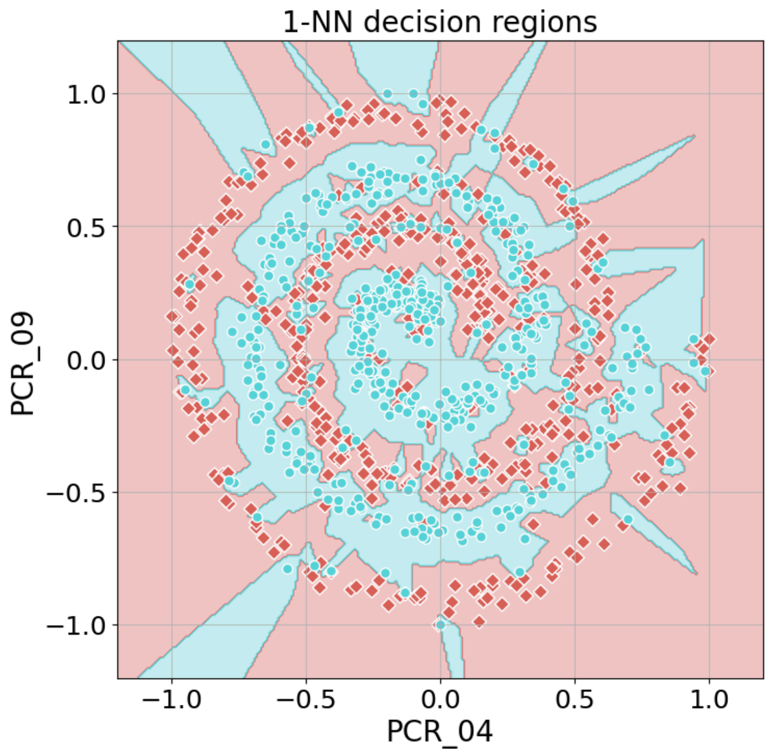
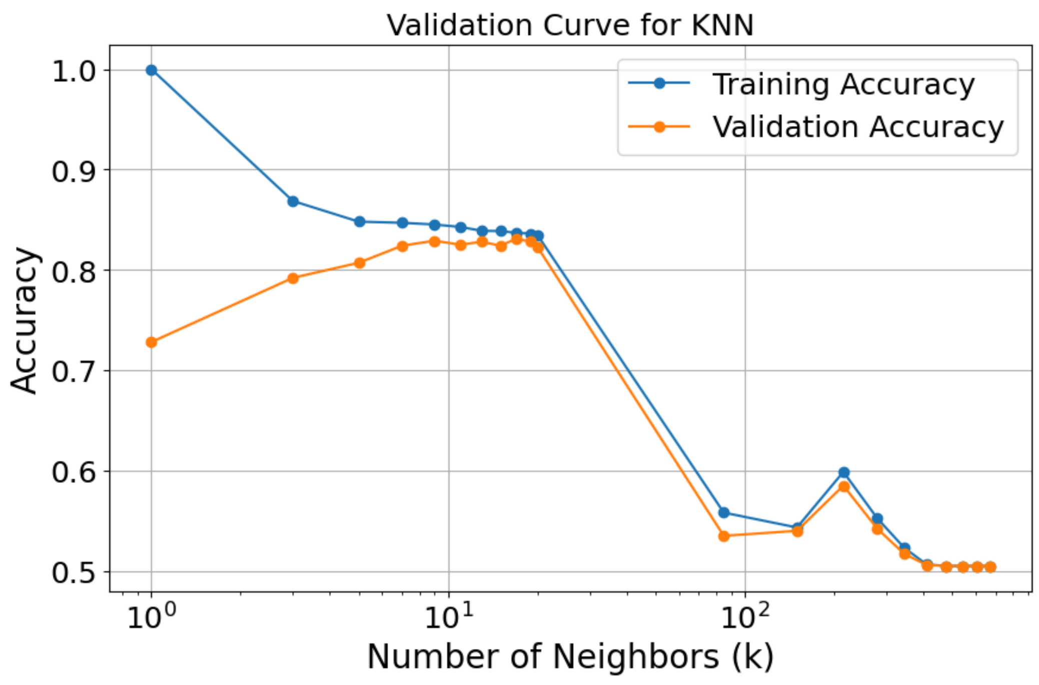
HW2 Report

Q1



Q2

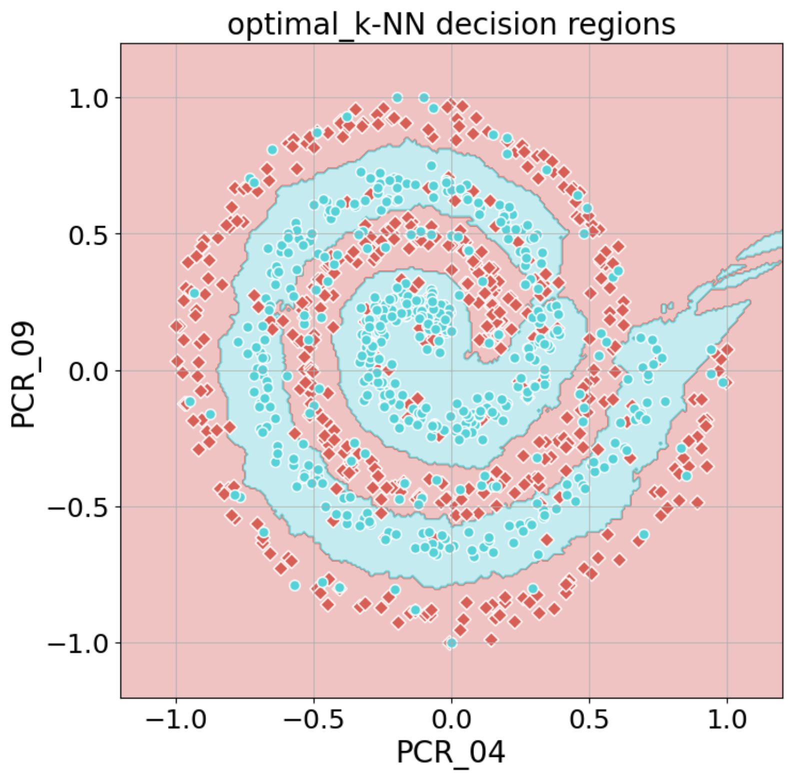
The best k is **17**.
The average training accuracy is: 83.69%
The average validation accuracy is: 83.1%



The k values in the range [1,8] cause overfitting because the training accuracy is high and the validation accuracy is low, which indicates that the kNN model does not generalize enough - it is too sensitive to the noise in the training set (hence more accurate on the training set).

The k values in the range [85,670] cause underfitting because the training and validation accuracies are low (around 0.5-0.55), which indicates that the kNN model generalizes too much, essentially becoming a majority vote on the training set.

<u>Q3</u>
The final model (optimal_k=17 on all the training set) decision regions:

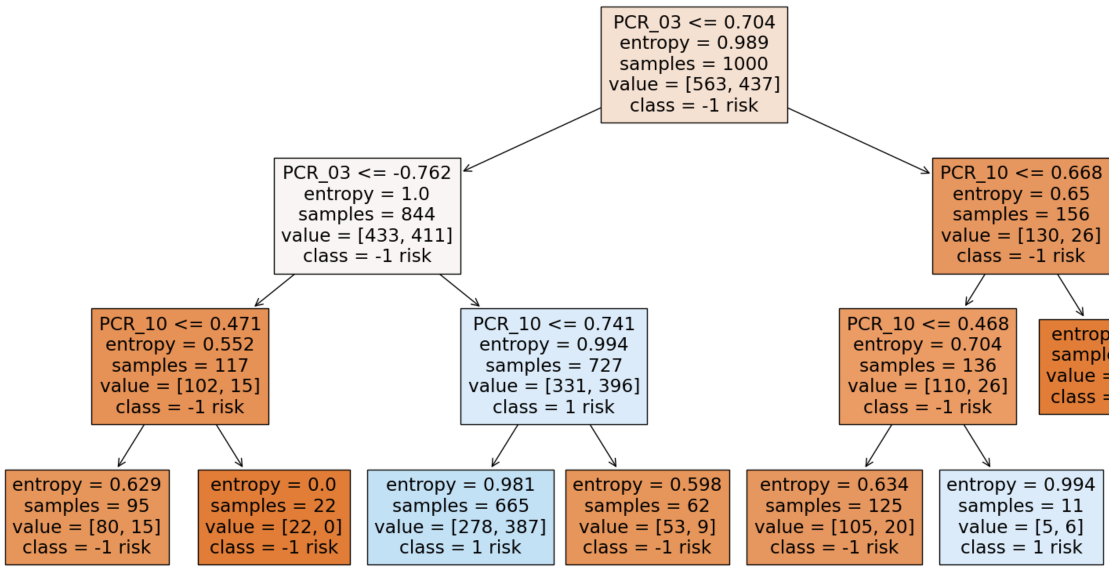


Test accuracy: 84.4%

<u>Q4</u>
We can see that the 17-NN decision regions are smoother than the 1-NN decision regions, and the 17-NN regions are fewer. The 1-NN decision regions were skewed towards noisy training samples, creating choppier regions.

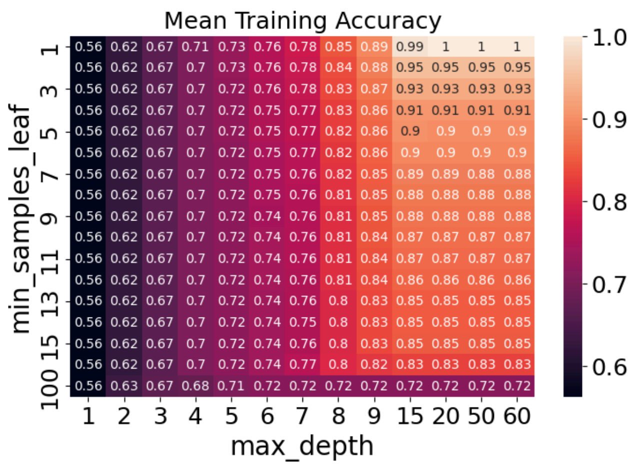
Q5

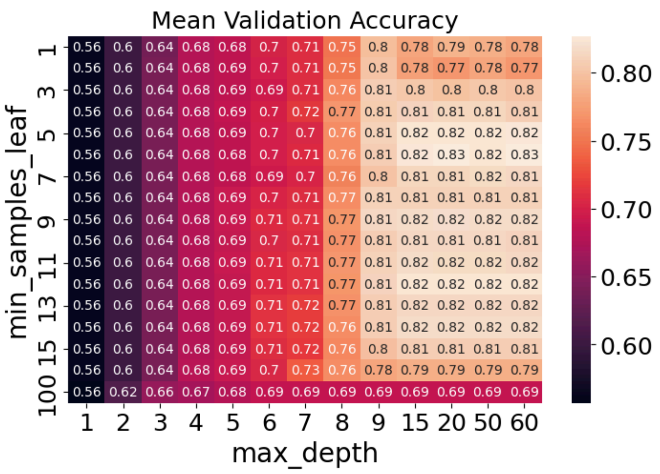
The training accuracy of ID3 with max_depth=3 is: 67.3%



Q6:

c.





The optimal hyperparameter combination is:

max_depth=20, min_samples_leaf=6

d. A hyperparameter combination that causes underfitting:
max_depth=**1**, min_samples_leaf=**100**
(As we can see it has both low train and validation accuracies)

e. A hyperparameter combination that causes overfitting:
max_depth=**60**, min_samples_leaf=**1**

(As we can see it has a high train accuracy but low validation accuracy)

f.
The combination max_depth=**1**, min_samples_leaf=**100** causes underfitting because the tree makes only one decision (because max_depth=1) based on one feature, thus not utilizing the other features, leading to poor performance on both train and validation sets.
Note that when max_depth=1, the underfitting is as bad for any min_samples_leaf value because the value doesn’t matter if it is under 1000 (number of training samples) since there will always be one split only. So we could choose any other min_samples_leaf value with max_depth=1 as an example for underfitting. We chose 100 because in general, a high min_samples_leaf value can cause underfitting and a low value causes overfitting (will be explained bellow).

The combination max_depth=**60**, min_samples_leaf=**1** causes overfitting because the high depth and the low number of samples in a leaf allow the tree to capture the training set perfectly, including the noise. The tree is very complex (many branches) and does not generalize, leading to perfect train accuracy and low validation accuracy.

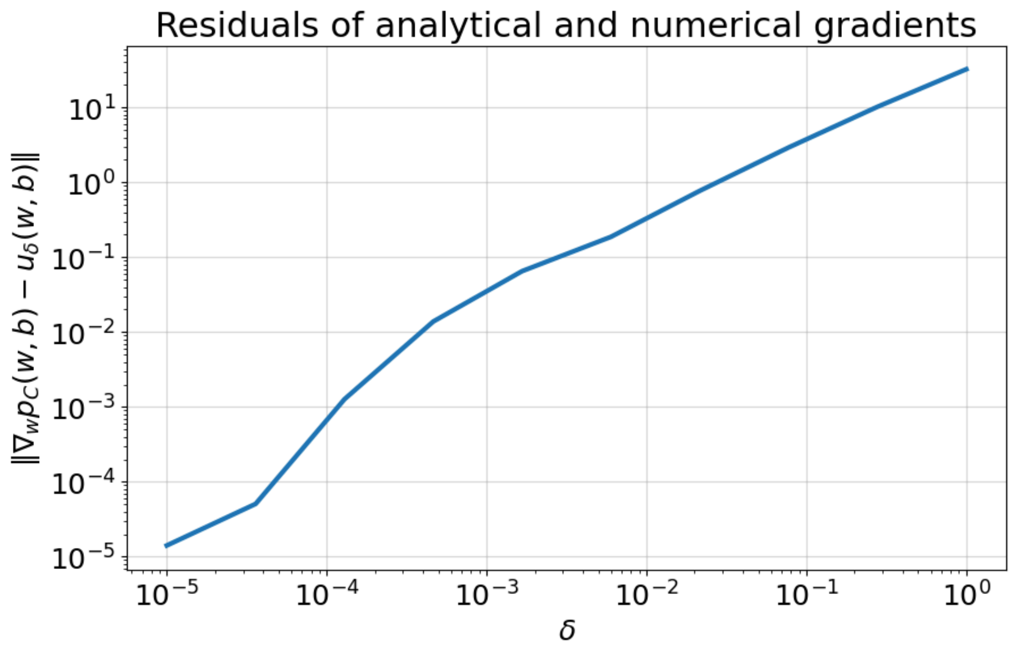
Q7:
The number of hyperparameter combinations that were evaluated is:
13 * 17 = **221**
because there are 13 different values for max_depth and 17 for min_samples_leaf.

If there were a third parameter with X possible values in its range, then the original number of hyperparameter combinations would be multiplied by X.

If the value range of the new parameter is large, it increases the number of combinations significantly, causing the grid search to take longer. We should take that into account when we want to add a parameter.

Q8:
Test accuracy of decision tree trained on all train set with best hyperparameter combination max_depth=20, min_samples_leaf=6:
80.4%

Q9:



As we can see, the smaller the delta is, the better the approximation (as expected by the Lagrange Residue theorem for Taylor polynomials).

Q10:

We can observe different loss/accuracy interactions in several 'step' segments:

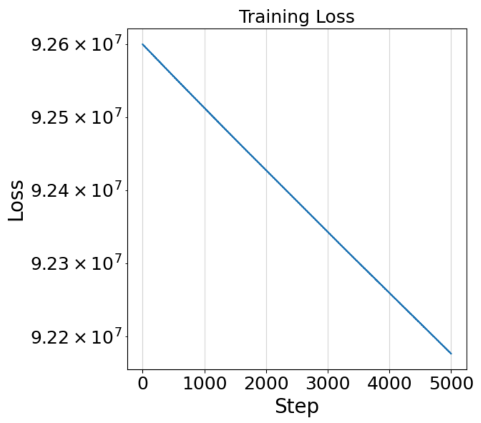
- Iterations 0-200 (approx.): loss decreases rapidly and accuracy increases.
- Iterations 200-2200 (approx.): loss decreases slowly and accuracy decreases.
- Iterations 2200-3400 (approx.): loss continues to decrease slowly and accuracy increases.
- Iterations 3400-4500: loss continues to decrease slowly and accuracy decreases.
- Iterations 4500-5000: loss continues to decrease slowly and accuracy increases a bit.

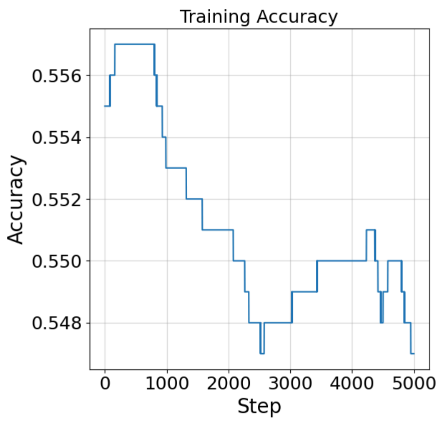
The behavior of the loss mostly matches expectations. We expect the initial loss to be high because we initialize with random weights, and then the loss should decrease thanks to GD updates.
Note that overall the loss values are of order 10^4 which is very high, but this is due to the C value being very high (10^11).
The sharp decrease in loss in iterations 0-200 and the slow decrease in loss in iterations 100-5000 are somewhat expected due to the very low learning rate (2*10^-14). The GD improves the model greatly at first but then continues to improve it very little. The low learning rate could also cause numerical issues, making the GD updates non-significant.

The accuracy fluctuations however do not match expectations. Initially, the accuracy increases as expected, but then the decrease is not expected. We would expect the accuracy to have an overall increasing trend. The two decreasing segments can be explained by severe overfitting due to noisy samples. The second increase segment can be explained by the GD making a recovery. Overall the fluctuations can also be due to the fact that the data isn't linearly separable and the C value is high, which heavily penalizes misclassifications, leading to unstable classifications over the iterations which leads to fluctuations in accuracy.

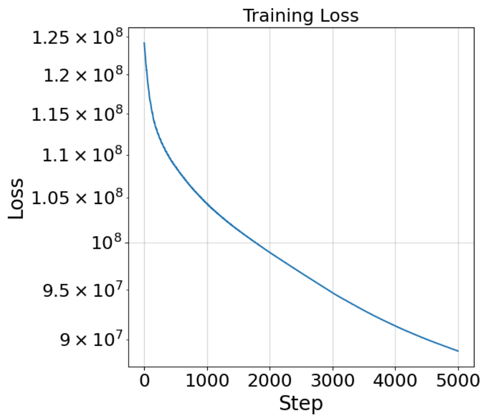
<u>Q11:</u>

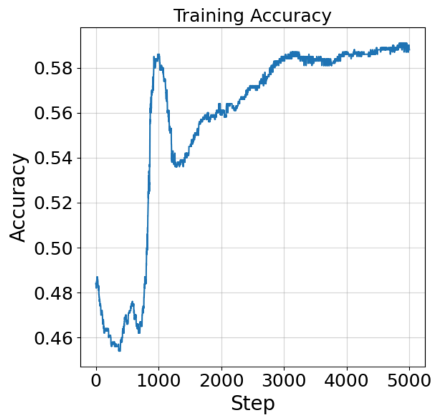
lr=1e-11:



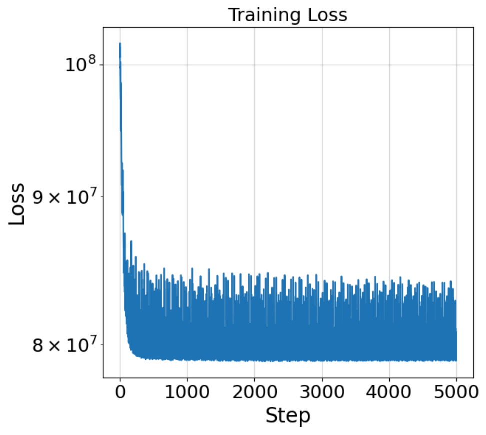


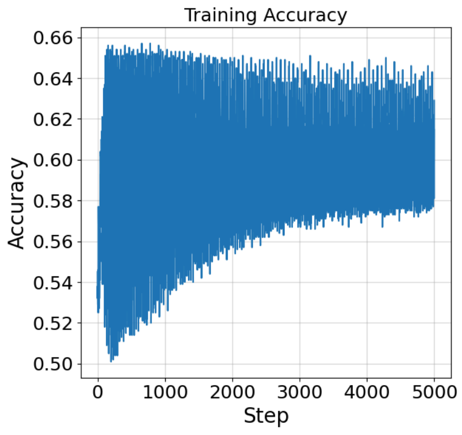
lr=1e-9:



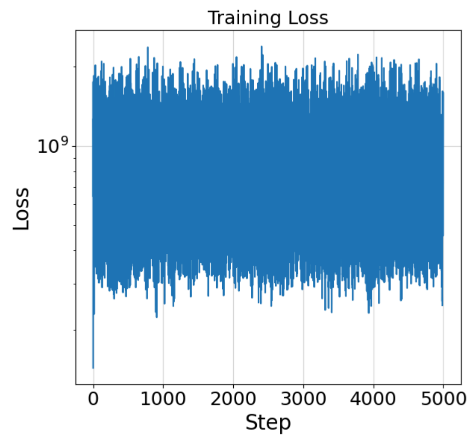


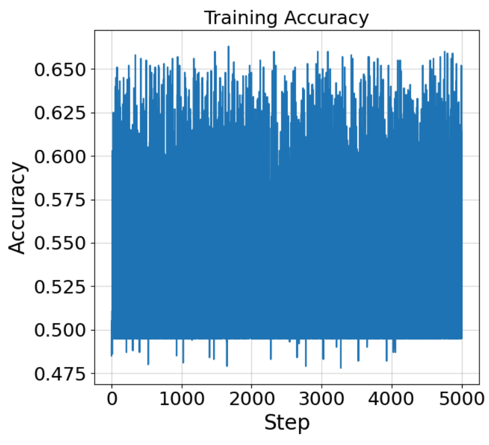
lr=1e-7:



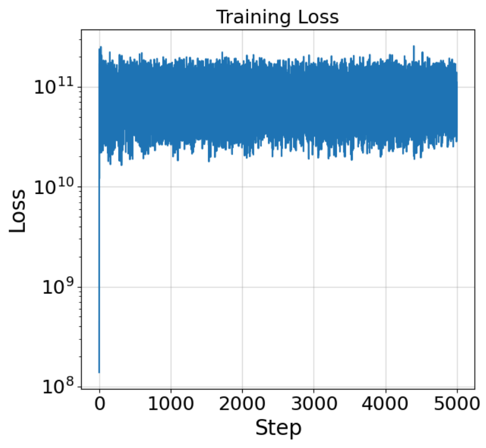


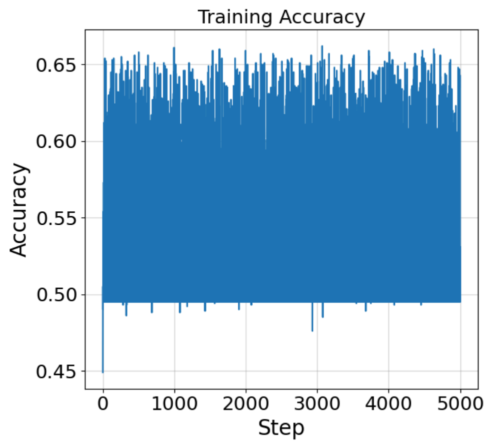
lr=1e-5:





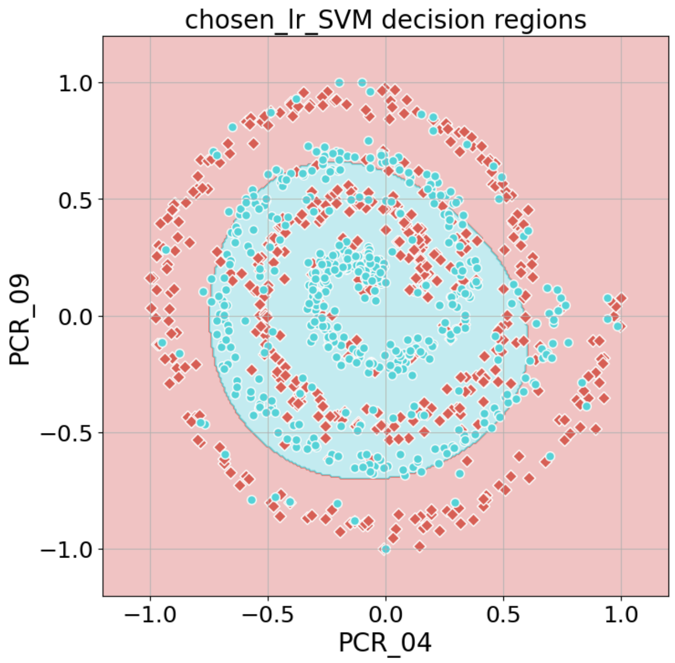
lr=1e-3:





We’d Choose **lr=1e-7**. This is because it reaches low loss and good average accuracy (60%) in a very short time, and then jumps between a bit better and a bit worse accuracy. This is better than lr=1e-9 where it takes a long time to reach the same accuracy, even though it is a bit more stable.
Of course, lr > 1e-5 is too much which we can see from the jumps in the loss and the fact that it is not declining, and lr < 1e-9 is too little and the accuracy barely increases (<0.5% for a very long time).

Q12:



Train accuracy: 60.7%
Test accuracy: 64.8%

Q13

a. The prediction rule: $h(x) = sign(\sum_{i=1}^{m} y_i K(x, x_i)) = sign(\sum_{i=1}^{m} y_i e^{-\gamma||x-x_i||})$

b. Since $y_i \in \{-1, 1\}$ we can split the sum according to y_i value:

$$h(x) = sign(\sum_{i=1}^{m} y_i K(x, x_i))$$

$$= sign(\sum_{j\in\{i\in[m]|y_i=1\}}^{m} 1 \cdot K(x, x_j) + \sum_{j\in\{i\in[m]|y_i=-1\}}^{m} (-1) \cdot K(x, x_j))$$

$$= sign(\sum_{j\in\{i\in[m]|y_i=1\}}^{m} K(x, x_j) - \sum_{j\in\{i\in[m]|y_i=-1\}}^{m} K(x, x_j))$$

c. From the given assumptions, there exists a tuple (x_i, y_i) such that $y_i = y = 1$ and $||x - x_i|| < \delta - 1$. In the sum $\sum_{j\in\{i\in[m]|y_i=1\}}^{m} K(x, x_j)$, one of the added values is $K(x, x_i)$ since $y_i = 1$. In addition, $K(x, x_j) > 0$ for any x_j (from the definition of the Laplacian kernel). Thus:

$$\sum_{j\in\{i\in[m]|y_i=1\}}^{m} K(x, x_j) > 0 +\ldots+ K(x, x_i) +\ldots+ 0 = e^{-\gamma||x-x_i||} > e^{-\gamma(\delta-1)}$$

($||x - x_i|| < \delta - 1 \wedge \gamma > 0 \Rightarrow -\gamma||x - x_i|| > -\gamma(\delta - 1)$ and the exponential function is increasing).

d. From the given assumptions, for every $j \in [m], j \neq i$ we have: $||x_i - x_j|| > 3\delta$ (the same x_i from earlier). For any $(x_j, y_j) \in S$ such that $y_j = -1$, we have that $||x_i - x_j|| > 3\delta$ (because any sample from S with the opposite label of x_i is different from x_i) and from the triangular inequality we have:

$||x_i - x_j|| = ||x_i - x + x - x_j|| \leq ||x_i - x|| + ||x - x_j||$

$\Rightarrow ||x - x_j|| \geq ||x_i - x_j|| - ||x_i - x|| > 3\delta - (\delta - 1) = 2\delta + 1 > 2\delta - 1$

$\Rightarrow -\gamma||x - x_j|| < -\gamma(2\delta - 1) \Rightarrow e^{-\gamma||x - x_j||} < e^{-\gamma(2\delta-1)}$

this is true for any $(x_j, y_j) \in S$ such that $y_j = -1$, so we can conclude that:

$$\sum_{j\in\{i\in[m]|y_i=-1\}}^{m} K(x, x_j) = \sum_{j\in\{i\in[m]|y_i=-1\}}^{m} e^{-\gamma||x-x_j||} \leq \sum_{j\in\{i\in[m]|y_i=-1\}}^{m} e^{-\gamma(2\delta-1)} = \frac{m}{2}e^{-\gamma(2\delta-1)}$$

e. given $\gamma = ln(\frac{m}{2})$:

$$\sum_{j\in\{i\in[m]|y_i=1\}}^{m} K(x, x_j) > e^{-\gamma(\delta-1)} = (e^{ln(\frac{m}{2})})^{-(\delta-1)} = (\frac{m}{2})^{1-\delta}$$

$$\sum_{j\in\{i\in[m]|y_i=-1\}}^{m} K(x, x_j) \leq \frac{m}{2}e^{-\gamma(2\delta-1)} = \frac{m}{2}(e^{ln(\frac{m}{2})})^{-(2\delta-1)} = (\frac{m}{2})^{1}(\frac{m}{2})^{-2\delta+1} = (\frac{m}{2})^{2-2\delta}$$

$$\Rightarrow \sum_{j\in\{i\in[m]|y_i=1\}}^{m} K(x, x_j) - \sum_{j\in\{i\in[m]|y_i=-1\}}^{m} K(x, x_j) > (\frac{m}{2})^{1-\delta} - (\frac{m}{2})^{2(1-\delta)} = (\frac{2}{m})^{\delta-1} - (\frac{2}{m})^{2(\delta-1)}$$

$$= (\frac{2}{m})^{\delta-1}(1 - (\frac{2}{m})^{\delta-1}) > 0$$

The last inequality holds because $m \geq 3 \Rightarrow 0 < \frac{2}{m} < 1 \Rightarrow 0 < (\frac{2}{m})^{\delta-1} < 1$ because $\delta > 1$. So we have that both $(\frac{2}{m})^{\delta-1}$ and $1 - (\frac{2}{m})^{\delta-1}$ are positive.

Note that $m \geq 3$ because $\gamma = \frac{1}{2\sigma^2}$ by definition so $\gamma > 0$ but also $\gamma = ln(\frac{m}{2})$ so if $m = 2$ then $\gamma = 0$ and if $m = 1$ then $\gamma < 0$ so it must be that $m \geq 3$.

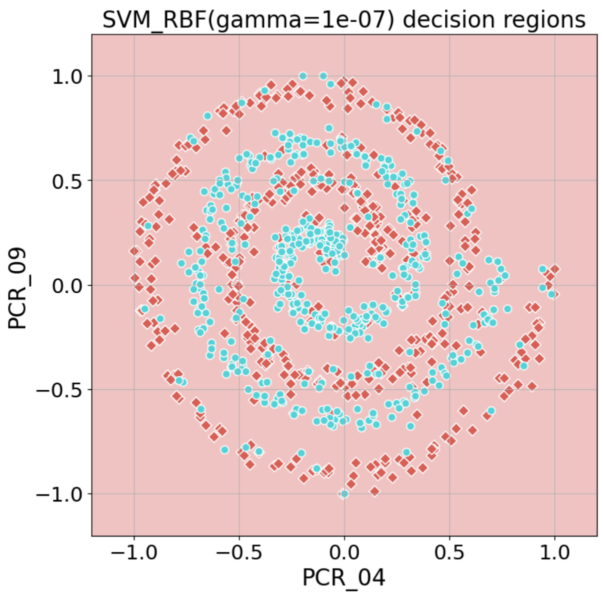
Finally, we have proved that $\sum_{j\in\{i\in[m]|y_i=1\}}^{m} K(x, x_j) - \sum_{j\in\{i\in[m]|y_i=-1\}}^{m} K(x, x_j) > 0$

which means that its sign is +1:

$$h(x) = sign(\sum_{j\in\{i\in[m]|y_i=1\}}^{m} K(x, x_j) - \sum_{j\in\{i\in[m]|y_i=-1\}}^{m} K(x, x_j)) = +1$$

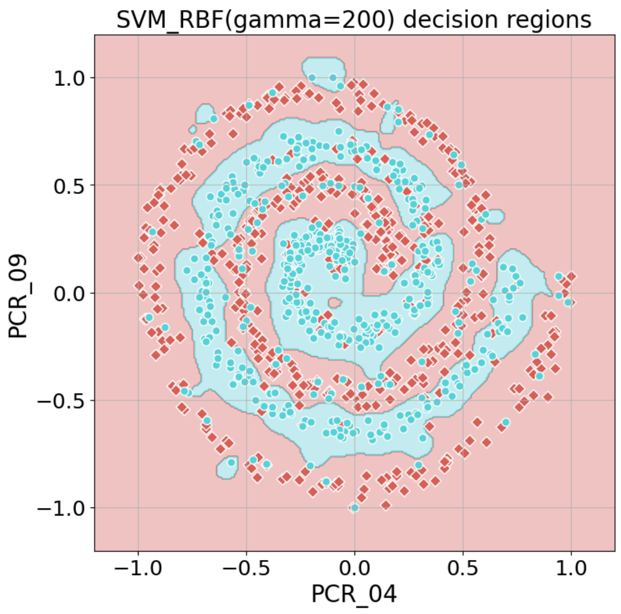
f. Given the assumption over the distribution, we have proved in c-e that if $x \in D$ has a class $y = 1$ then this kernel will always result in $h(x) = 1$. However the difference between $y = \pm 1$ is arbitrary, and the kernel would work the same way if we flipped the sign of y. Mathematically speaking, this symmetry proves that the same proof from c-e would result in the fact that if $x \in D$ has a class $y = -1$ then the kernel will always result in $h(x) = -1$. Together we have proved that in general, for every $x \in D$ this model will predict the correct y. This is the **General Prediction Rule**.

Q14:

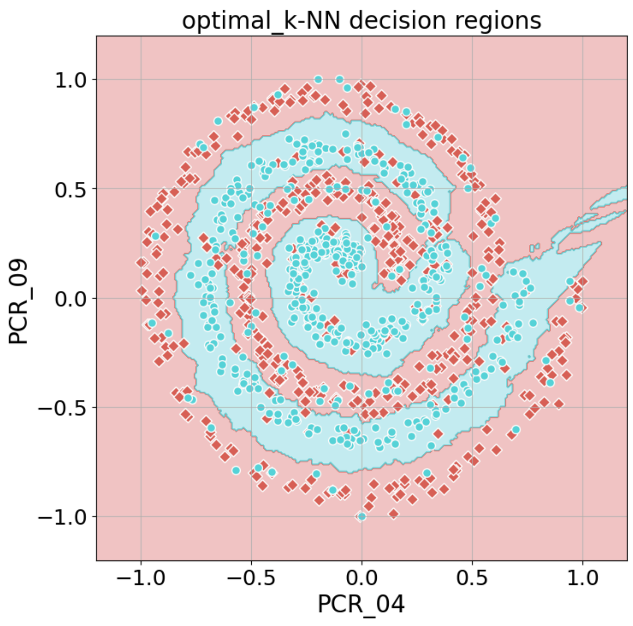


Train accuracy: 50.5%
Test accuracy: 48.0%
We would classify it as an **underfit**. This model just guesses that everyone is red. While that is ok on average (50-50 split), in reality, this shows that the model hasn't learned anything from the training sample but simply chooses a majority vote (=red, because there's slightly more red than blue). This is a sign of underfitting.

Q15

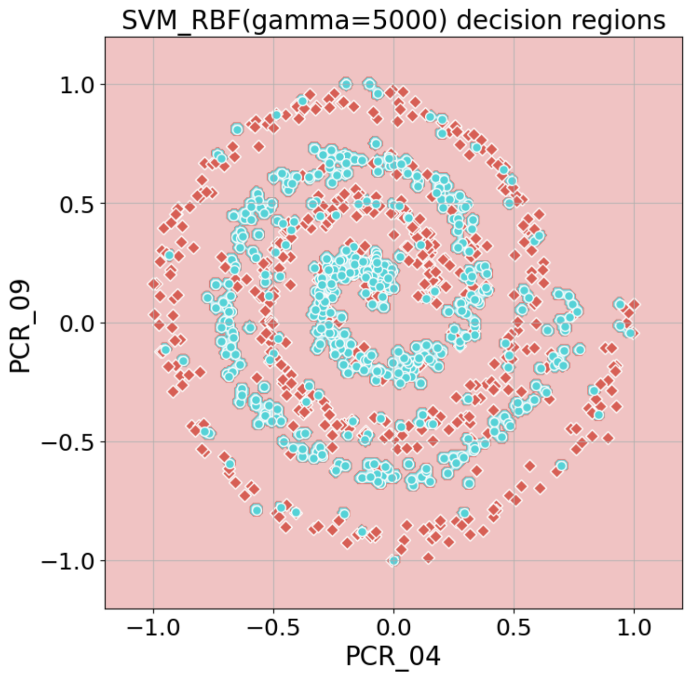


Train accuracy: 85.9%
Test accuracy: 82.0%
We would classify it as a **reasonable fit.**
We should note that gamma is pretty large, which makes $K(x, x_i) = exp\{-\gamma||x - x_i||_2^2\}$ a very narrow Gaussian. This effectively means that each training point is represented by a narrow Gaussian, hence the largest contributions to the kernel decision come from the very nearest neighbors (further points decay exponentially faster). With that in mind, the RBF kernel gives similar decision regions to the 17-NN but with a significant difference.
Like we described, RBF treats every data-sample as a narrow gaussian and hence we can see “blobs” of blue in the “sea” of red in the RBF model, because if the RBF sees a few points together it will give a strong prediction there. This is opposed to the 17-NN which checks all 17 neighbors and then decides, which gives smooth and less noisy decision regions (i.e ignoring small blobs of blue). This means that RBF can sense good detail with the price of overfitting, and the 17-NN gives smoother but less detailed regions. In this distribution, the 17-NN works better than the RBF.



(This is the 17-NN decision regions from part 1 for comparison, which got 84.4% Test Accuracy)

<u>Q16</u>



Train accuracy: 98.5%
Test accuracy: 68.4%
We would classify it as **overfitting**. This model simply predicts blue only in very close proximity to the blue training samples, and red otherwise (the blue regions cannot even be seen since they are the size of the sample dot itself). This allows the model to achieve high train Accuracy but clearly does not generalize well. This is a sign of overfitting.