

Introduction to Machine Learning 236756

Assaf Alon, 207376807

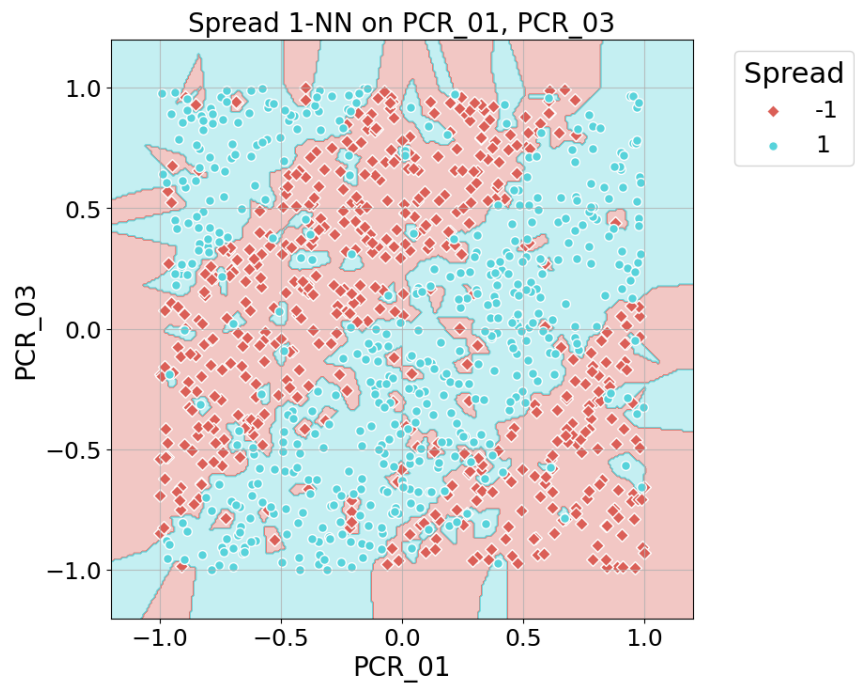
Daniel Gershkovich, 209088723

Major HW 2

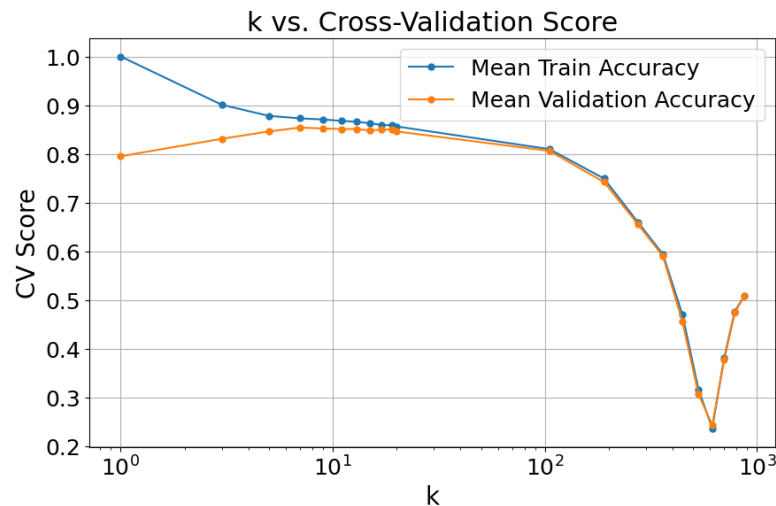
14/03/2024

Part 1 - Basic model selection with k-Nearest Neighbors

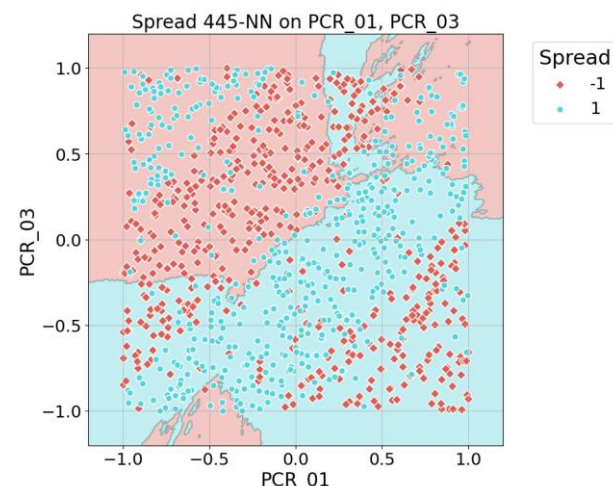
1. We did it. Here's the graph:



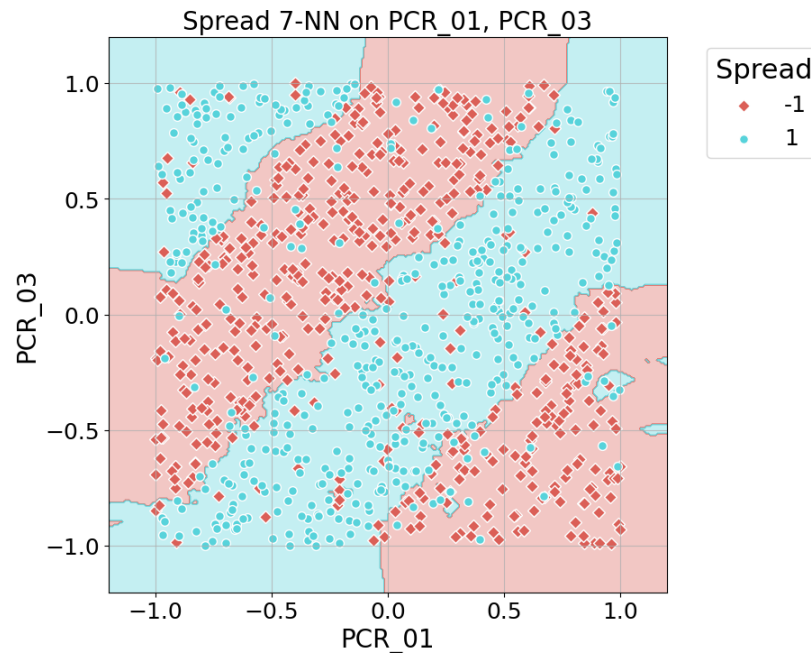
2. Here's the graph:



- Best k (number of neighbors): $k = 7$.
- (Estimated) Average **Training** Accuracy ≈ 0.873
- (Estimated) Average **Validation** Accuracy = 0.854
- k values that **overfit – Smaller values**, such as 1,3, and it's apparent from the above graph since there's a significant gap between the train accuracy (100% correct) and the test data (approx. 80%). They cause overfit because they divide the graph into very small, highly detailed regions, taking into consideration very few neighbors (the test accuracy will be high since the datapoint itself counts as its own neighbor, thus contributing to fitting the data)
A good visual example is the graph provided in Q1.
- k values that **underfit – Larger values**, such as 445,530. They cause underfitting because of overgeneralization: the graph will have few distinct regions, and kNN will ignore small details in the data. For example, for $k = 445$ (on the right), the bottom right region is classified as spread = +1, ignoring the obvious majority of datapoints labeled as -1.



3. The graph of our optimal kNN ($k = 7$)



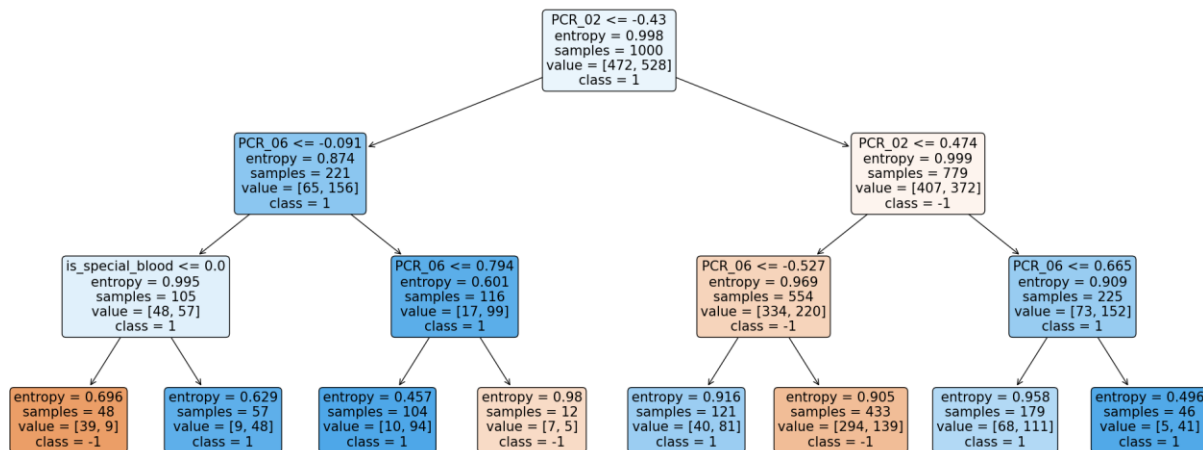
Test accuracy: **0.872**.

4. The graph has much simpler boundaries for $k = 7$ relative to $k = 1$. We can infer that kNN with $k = 7$ gives much less emphasis for individual datapoints. As discussed previously, this leads to **less sensitivity** in **mislabeled datapoints**, and **less sensitivity** to different **data partitions** to train and test sets. It is noteworthy that it could cause information loss, ignoring smaller features in the graph (for $k = 7$ the information loss isn't substantial)

Part 2 – Decision Trees

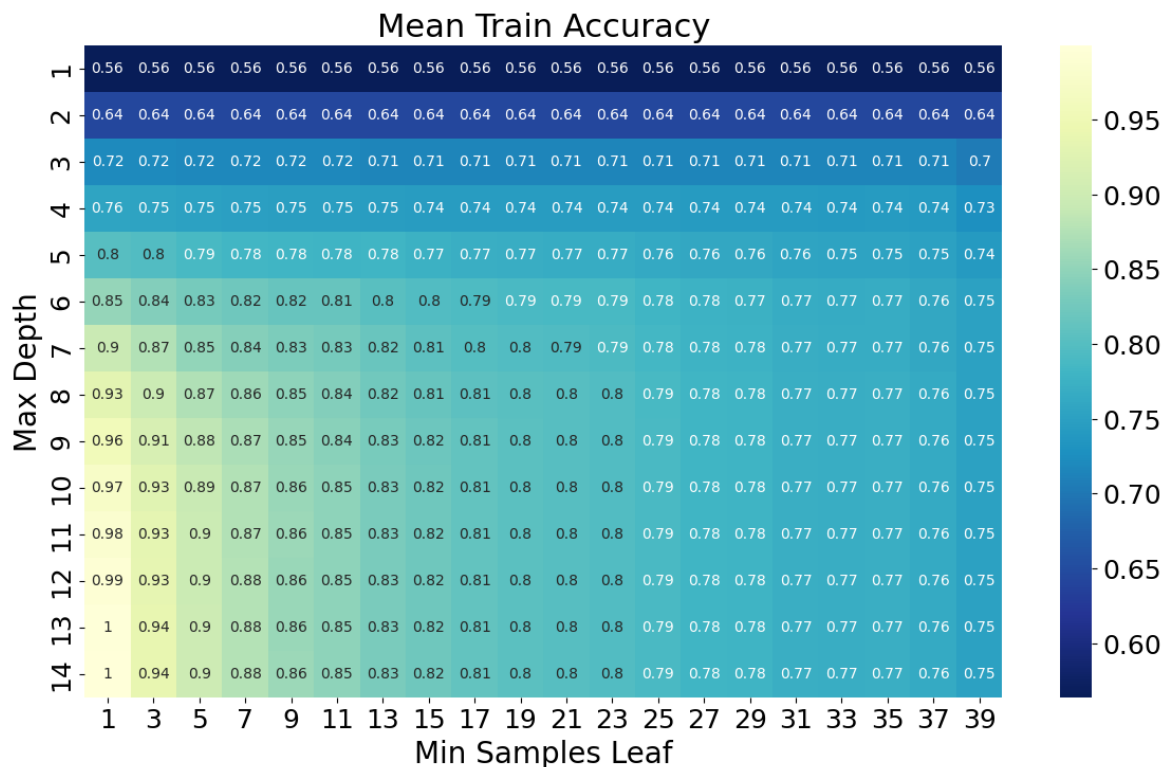
5. Visualization of the tree:

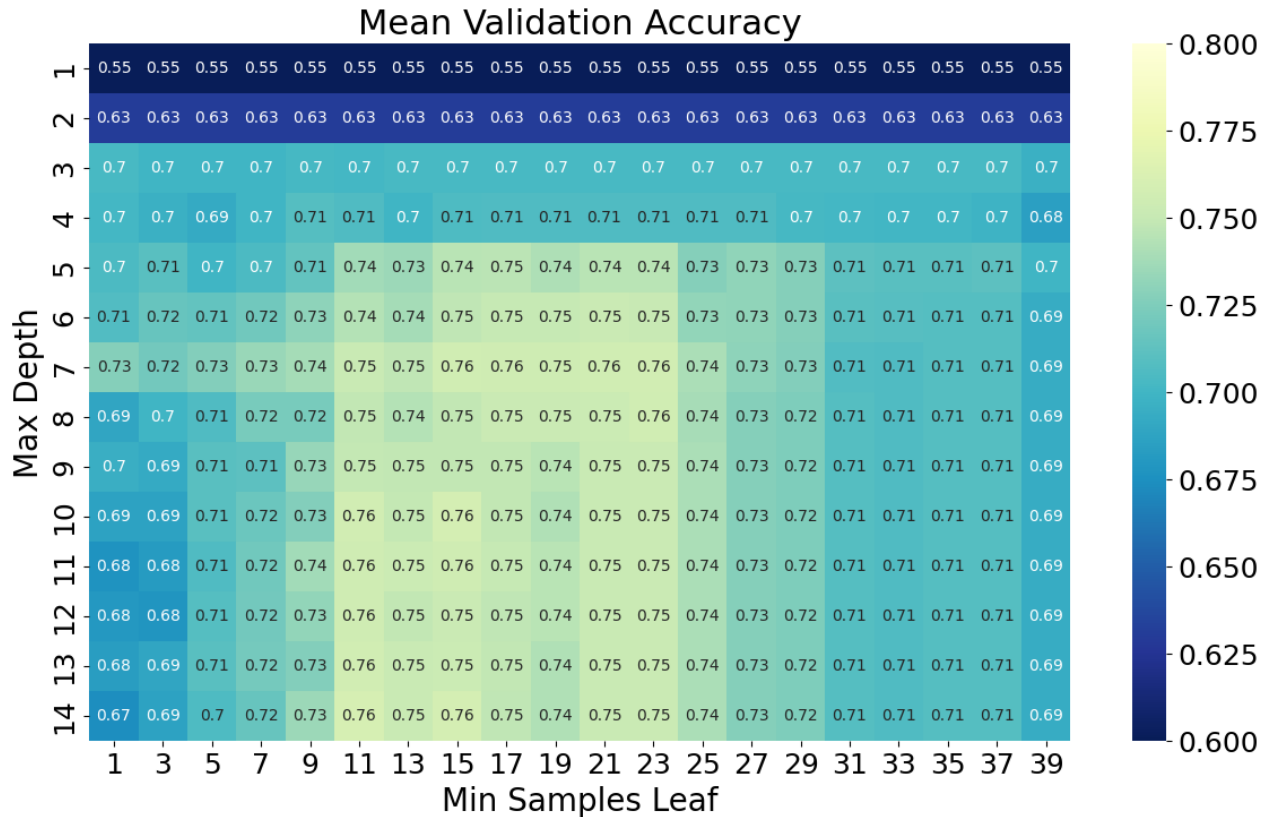
ID3 on training set with max depth 3



The **training** accuracy is **0.715**.

6. Here's the two plots (more on the next page):





- Optimal combination: **max depth = 7, min samples leaf = 17**, with test accuracy of **0.76** (not unique, there are other combinations with the same accuracy)
- **Underfitted** combination: any combination with **Small max depth** (for example max_depth=1, min_samples=39). It caused underfitting because there are very few possible trees that could form with these parameters, so the number of classifiers in the class is small, and thus it is unlikely that the data fits well to any one of them.
- **Overfitted** combination: **Large max depth, small min samples** (for example max_depth=14, min_samples=1). It caused overfitting because the tree is practically unlimited in complexity (for this specific task with specific data). In ID3, this will result in a perfectly fitted tree for the training data (given that it's consistent), at the expense of validation accuracy, a classic case of overfit.

7. We evaluated combinations of 14 different max_depth values, and 19 different min_samples values. Overall, $14 \cdot 19 = \mathbf{266}$ options. A third hyperparameter would multiply this number by the number of values for the third parameter we wished to test. If d is the number of parameters and n is the number of values for each parameter, then the number of combinations is $O(n^d)$ (**exponential growth with relation to added parameters**).

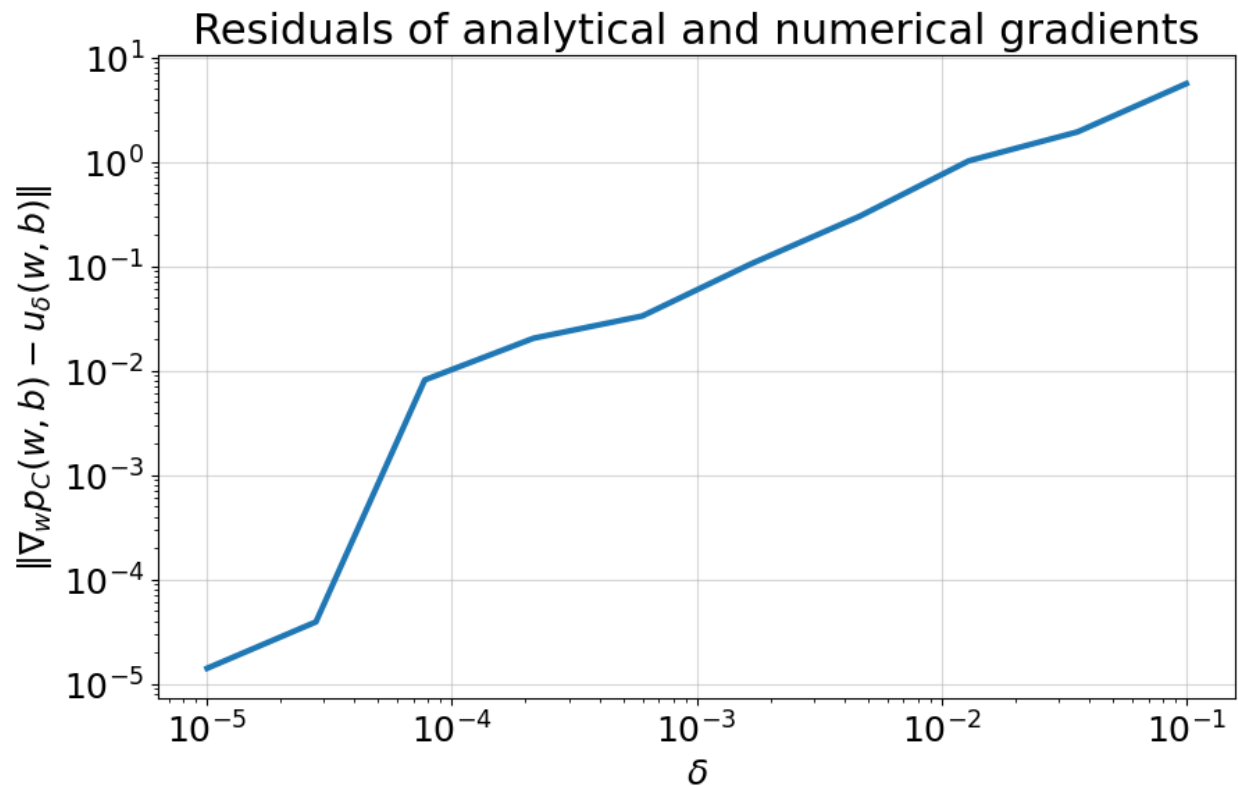
8. The optimal hyperparameter combination we found:

- Max depth = 7
- Min samples = 17

The retrained tree with these parameters had a test accuracy of **0.764**.

Part 3 – Linear SVM and the Polynomial Kernel

9.



It seems like that difference between the numerical gradient and the analytical gradient starts small and increases as δ increases.

This behavior is expected, as δ is the step parameter of the numerical gradient. Since the definition of gradient is $\lim_{\delta \rightarrow 0} \frac{f(x_0 + \delta) - f(x_0)}{\delta}$, it makes sense that for small δ values the difference between the actual gradient and $\frac{f(x_0 + \delta) - f(x_0)}{\delta}$ is relatively small, and as δ gets further away from 0, the calculation will get less precise.

10. In the given example, the values chosen were extreme:

C is ginormous, while lr is minuscule.

The small lr value means the learning rate is slow – that is reflected in the accuracy graph by the **relatively tame changes on each step** (the impact of smaller lr values can be understood better on the next question's graphs).

The **train loss** is somewhat reduced as iterations play out but stays very large (order of 10^{14}) since the data is not linearly separable and C is large. **The choice of a large C gives extreme emphasis to minimizing the loss** on the expense of regularizing w . Since the data is not linearly separable, there's probably a very high lower bound for the train loss.

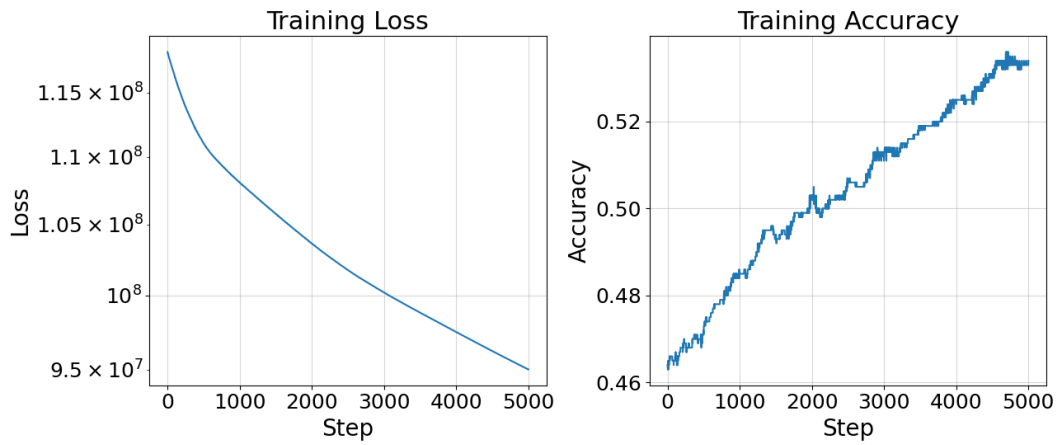
It is noteworthy that when the loss drops in the first iterations, the training accuracy increases, and as the loss-drop gets tamer, the training accuracy starts dropping. This may be counter-intuitive at first, since for this large C value the loss may be thought of as correlated to accuracy, but that's true mostly for 0-1 loss, and it's important to remember that in contrast, **hinge-loss can increase drastically due to incorrectly labeled outliers** without affecting the accuracy by much (as in, some wrong labels may cause high penalties).

In the end, it seems like the algorithm is on the verge of converging (this observation could be false), but the accuracy is quite bad (< 0.5)

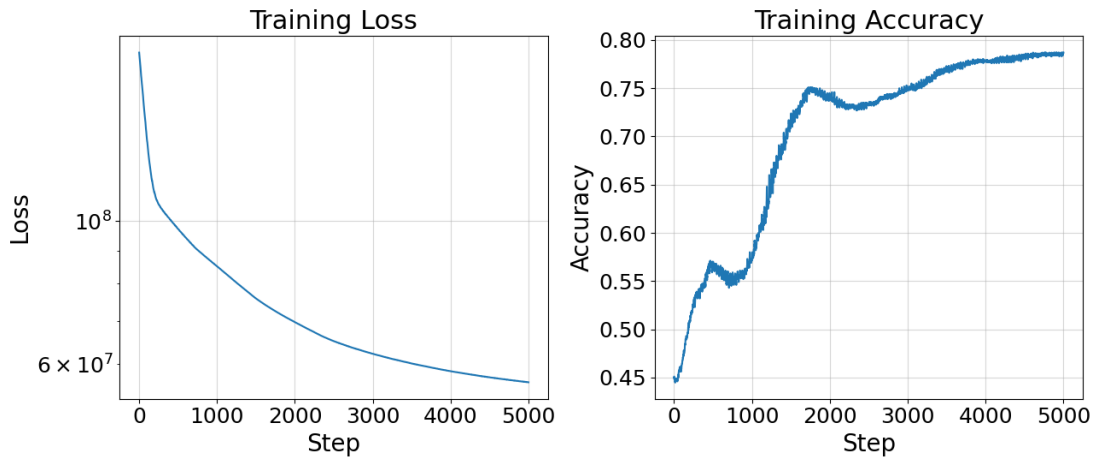
Overall, the initial phase of rapid loss reduction and accuracy improvement aligns with our expectations, while the latter parts are funkier, but we can justify them by the fact that the algorithm doesn't focus solely on reducing loss and is also stochastic.

11. All the following use $C = 10^5$:

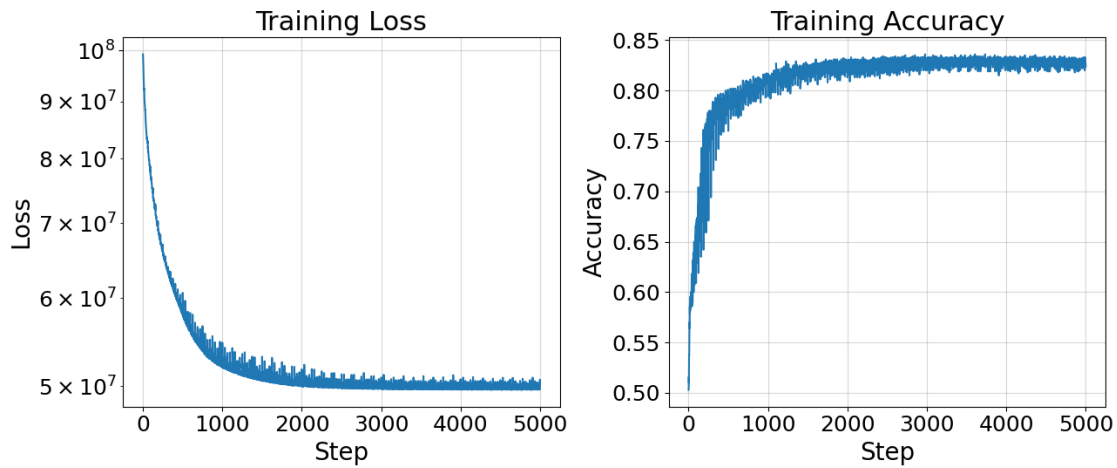
Learning rate: $1e-09$

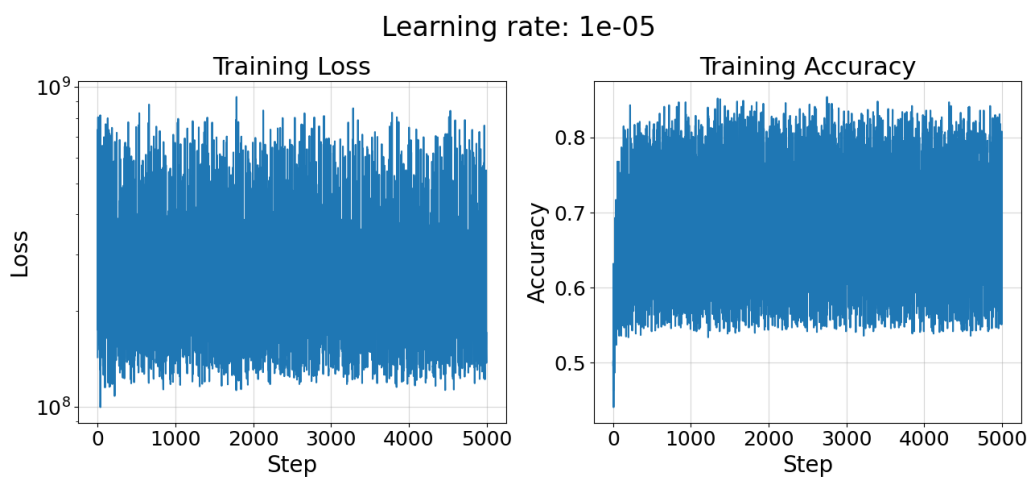
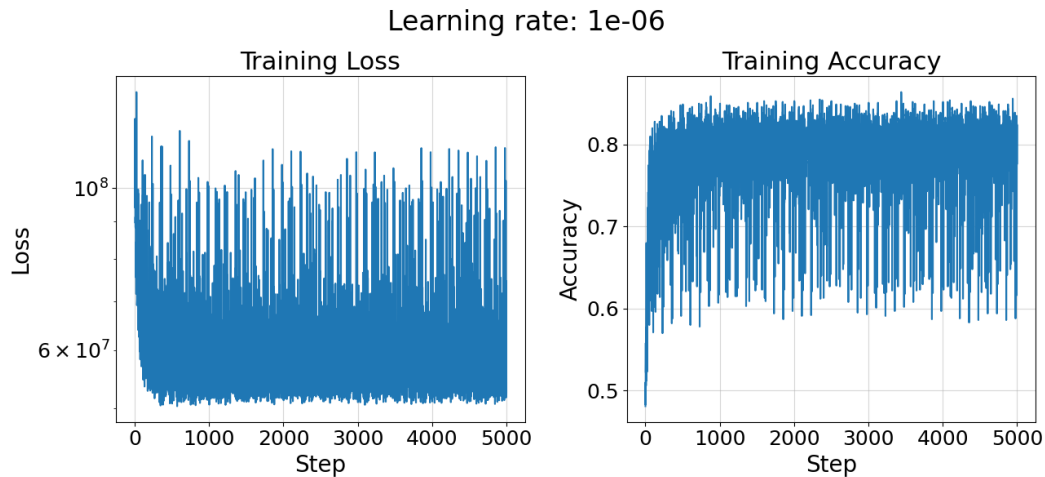


Learning rate: $1e-08$



Learning rate: $1e-07$





It is clear that $lr = 10^{-5}, 10^{-6}$ diverges, probably due to too big of a learning rate.

$10^{-9}, 10^{-8}$ don't converge quickly enough because of relatively small steps in each iteration. Note that 10^{-9} 's accuracy is low within the provided iteration limit.

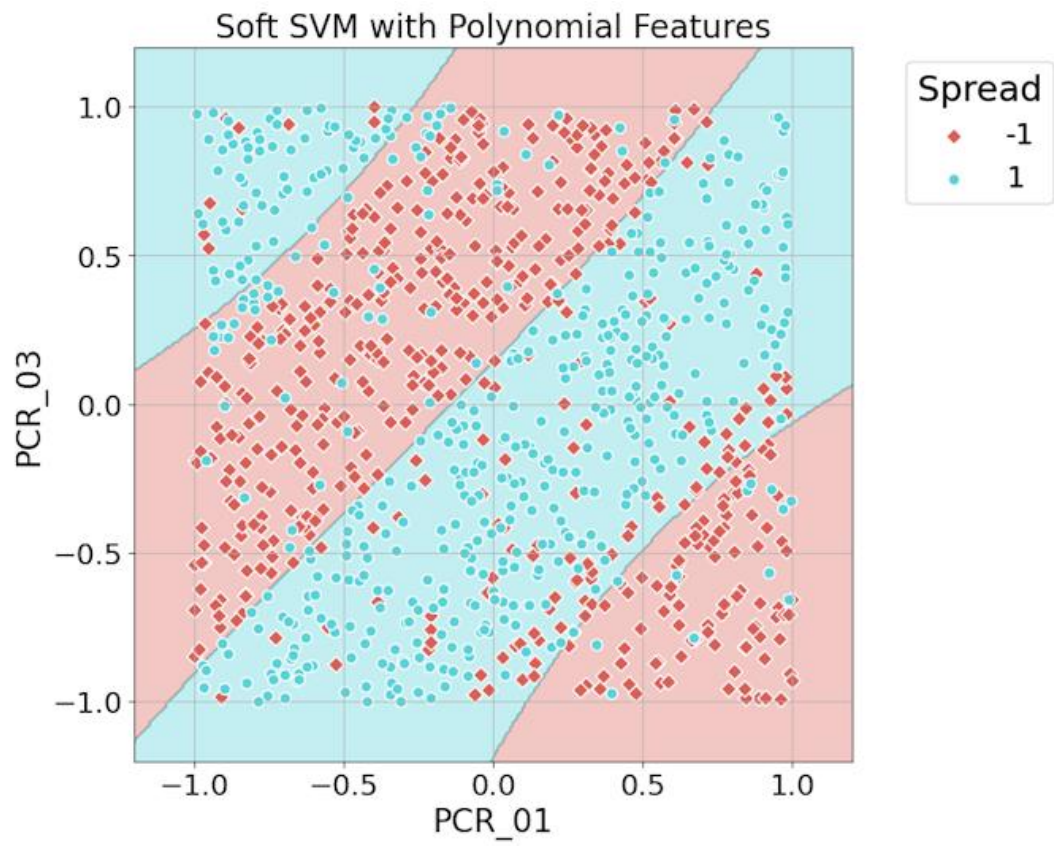
We would choose 10^{-7} as it seems to be close to the sweet spot.

On the one hand it converges faster than $10^{-8}, 10^{-9}$, and reaches a higher accuracy value within the provided iteration limit.

On the other hand, it seems small enough to avoid divergent behavior near the optimum, which in 10^{-6} case causes wild swings in the accuracy and loss values.

12.

a. The trained model:



b. **Training** accuracy: **0.819**.

Test accuracy: **0.824**.

Part 4 – The RBF Kernel

13.

Section a.

By assumption (i):

$$\lim_{\gamma \rightarrow 0} \left(\text{sign} \left(\sum_{i \in [m], \alpha_i > 0} \alpha_i \cdot y_i \cdot e^{-\gamma \|x - x_i\|_2^2} \right) \right) = \text{sign} \left(\lim_{\gamma \rightarrow 0} \left(\sum_{i \in [m], \alpha_i > 0} \alpha_i \cdot y_i \cdot e^{-\gamma \|x - x_i\|_2^2} \right) \right)$$

Fixed i value. It holds that:

- α_i is a bounded constant by assumption (iii)
- $y_i \in \{\pm 1\}$
- By assumption (ii) $\exists c_1$ such that all features are bounded by c_1 ,
therefore: $\|x - x_i\|_2^2 \leq \|x\|_2^2 + \|x_i\|_2^2 \leq 2c_1^2$


$$\Rightarrow -|\gamma| \cdot 2c_1^2 \leq -\gamma \cdot \|x - x_i\|_2^2 \leq |\gamma| \cdot 2c_1^2$$

e^x is monotonic, therefore $\forall \gamma \in \mathbb{R}: e^{-|\gamma| \cdot 2c_1^2} \leq e^{-\gamma \|x - x_i\|_2^2} \leq e^{|\gamma| \cdot 2c_1^2}$

e^x is continuous, therefore $\lim_{x \rightarrow x_0} e^x = e^{x_0}$, thus:

$$\lim_{\gamma \rightarrow 0} \alpha_i \cdot y_i \cdot e^{-|\gamma| \cdot 2c_1^2} = \lim_{\gamma \rightarrow 0^+} \alpha_i \cdot y_i \cdot e^{-\gamma \cdot 2c_1^2} = \alpha_i \cdot y_i \cdot e^{-0 \cdot c_1^2} = \underbrace{\alpha_i}_{\text{bounded}} \cdot y_i \cdot 1$$

$$\lim_{\gamma \rightarrow 0} \alpha_i \cdot y_i \cdot e^{|\gamma| \cdot 2c_1^2} = \lim_{\gamma \rightarrow 0^-} \alpha_i \cdot y_i \cdot e^{-\gamma \cdot 2c_1^2} = \alpha_i \cdot y_i \cdot e^{-0 \cdot c_1^2} = \alpha_i \cdot y_i \cdot 1$$

By the squeeze theorem () , we'll get that $\lim_{\gamma \rightarrow 0} \alpha_i \cdot y_i \cdot e^{-\gamma \|x - x_i\|_2^2} = \alpha_i y_i$.

Continu(ou)es the next page.

Since the limit exists for each i , the limit of the sum of all $i \in [m]$ also exists, and is equal to the sum of the limits, therefore:

$$\lim_{\gamma \rightarrow 0} \left(\sum_{i \in [m], \alpha_i > 0} \alpha_i \cdot y_i \cdot e^{-\gamma \|x - x_i\|_2^2} \right) = \sum_{i \in [m], \alpha_i > 0} \alpha_i \cdot y_i \cdot 1$$

And it holds that:

$$\sum_{i \in [m], \alpha_i > 0} \alpha_i \cdot y_i = \sum_{\substack{i \in [m], \\ \alpha_i > 0 \\ y_i = +1}} \alpha_i - \sum_{\substack{i \in [m], \\ \alpha_i > 0 \\ y_i = -1}} \alpha_i$$

And so, the above is greater than 0 $\Leftrightarrow \sum_{\substack{i \in [m], \\ \alpha_i > 0 \\ y_i = +1}} \alpha_i > \sum_{\substack{i \in [m], \\ \alpha_i > 0 \\ y_i = -1}} \alpha_i$, so the sign value

of the above can be rewritten as: $\operatorname{argmax}_{y \in \{\pm 1\}} \sum_{\{i | y_i = y\}} \alpha_i$

And therefore:

$$\operatorname{sign} \left(\lim_{\gamma \rightarrow 0} \sum_{i \in [m], \alpha_i > 0} \alpha_i \cdot y_i \cdot e^{-\gamma \|x - x_i\|_2^2} \right) = \operatorname{sign} \left(\sum_{i \in [m], \alpha_i > 0} \alpha_i \cdot y_i \right) = \operatorname{argmax}_{y \in \{\pm 1\}} \sum_{\{i | y_i = y\}} \alpha_i \quad \blacksquare$$

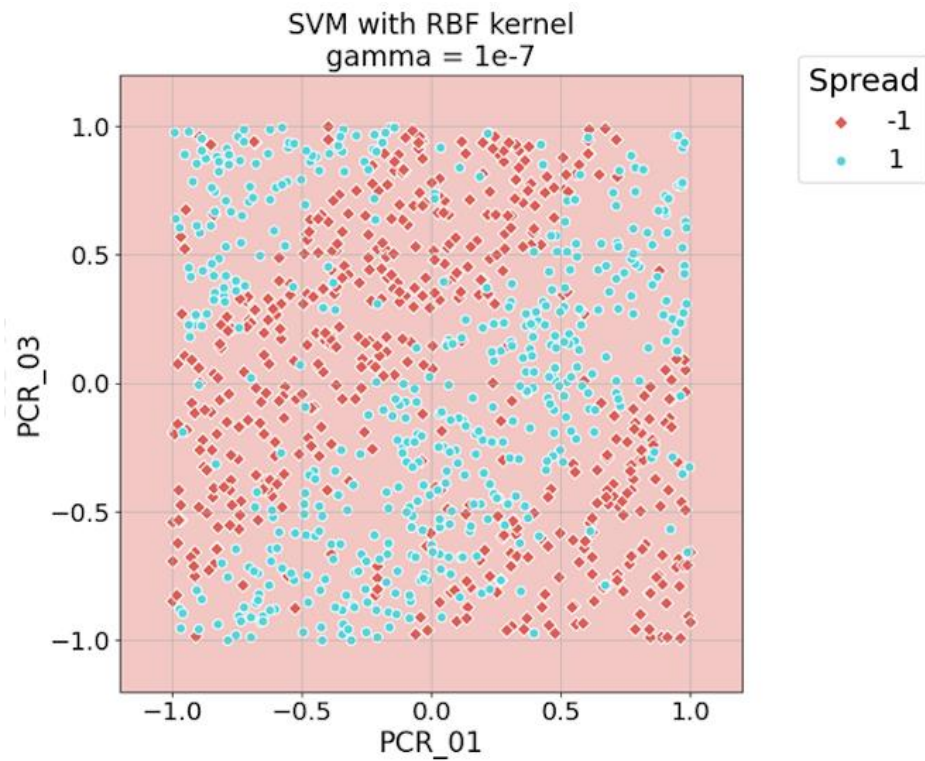
Section b.

The decision rule becomes rule by majority, or something like “ ∞ nearest neighbors”, since, as shown in the previous section, the classification rule is a constant label determined by the limit of the sign of the sums, which is equal to $\operatorname{argmax}_{y \in \{\pm 1\}} \left(\sum_{\{i | y_i = y\}} \alpha_i \right)$, and since $\alpha_i = 1$ it holds that:

$$\operatorname{argmax}_{y \in \{\pm 1\}} \sum_{\{i | y_i = y\}} \alpha_i = \operatorname{argmax}_{y \in \{\pm 1\}} \sum_{\{i | y_i = y\}} 1 = \operatorname{argmax}_{y \in \{\pm 1\}} (|\{y_i = y | i \in [m]\}|)$$

Which is equal to the majority label.

14. The decision boundary is:

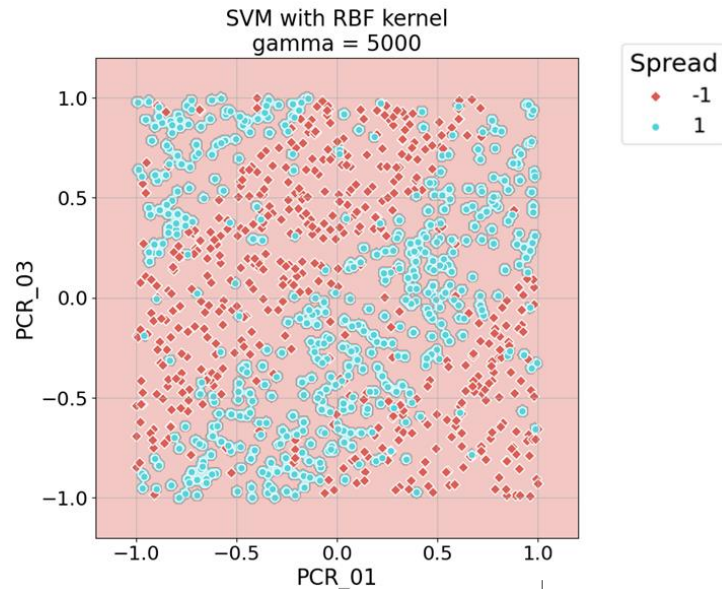


This matches the decision rule discussed in Q13 section b, since the classification is always spread = -1, and this is indeed the majority label:

```
spread
-1    508
 1    492
Name: count, dtype: int64
```

15. The decision boundary is →

The decision regions **vary** greatly from 1-NN, for example it can be observed that the upper left corner is classified as -1, even though the nearest neighbor's spread is labeled as +1. Generally, 1-NN chooses spread labels based on the nearest neighbor's label, while this model labels +1 if and only if the point is close to a datapoint labeled with +1, and otherwise -1.



We believe the difference stems from the fact that when γ is very large, the **distance between x, x_i must be very small for $e^{-\gamma \|x - x_i\|_2^2}$ to be non-negligible** (for example, even when the norm of the difference is as small as 0.05, the exponent is $e^{-5000 \cdot 0.05^2} \approx 0.000004$).

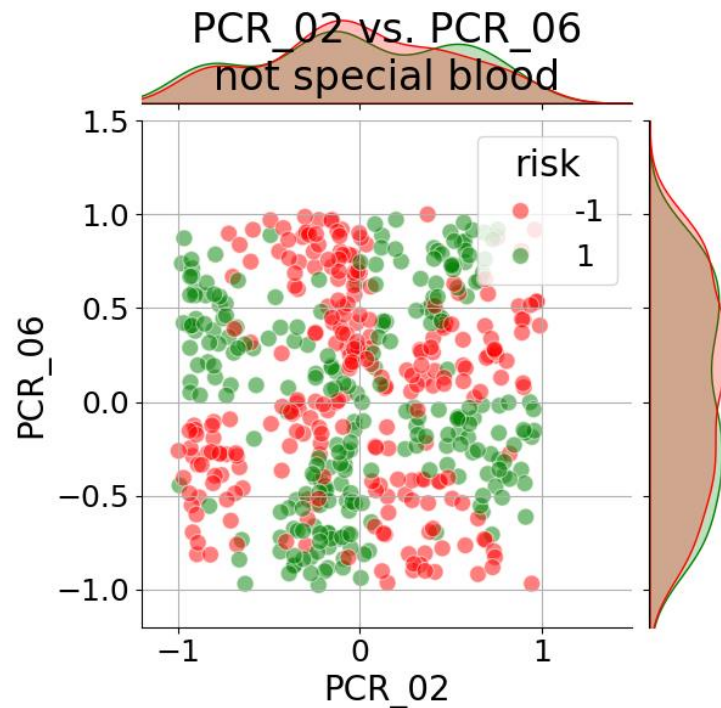
x	$e^{-x^2 5000}$
0	1
0.01	0.60653066
0.05	3.726653×10^{-6}
0.1	1.92875×10^{-22}
0.11	5.311092×10^{-27}
0.15	1.386343×10^{-49}
0.2	1.383897×10^{-87}

We believe that points that aren't extremely close to points that are labeled 1 are classified as -1 because of **a non-negligible negative bias**. Any coordinate that isn't extremely close to any datapoint will have the classifier sum negligible numbers that are orders of magnitude away from the bias, causing the classifier to classify these coordinates as -1.

Another possible explanation: we're dealing with numbers of magnitude of 10^{-20} , and as stated in Numerical Algorithms, these are "Matlab's zeroes". Numerical errors may apply.

Part 5 – Custom Feature Mapping

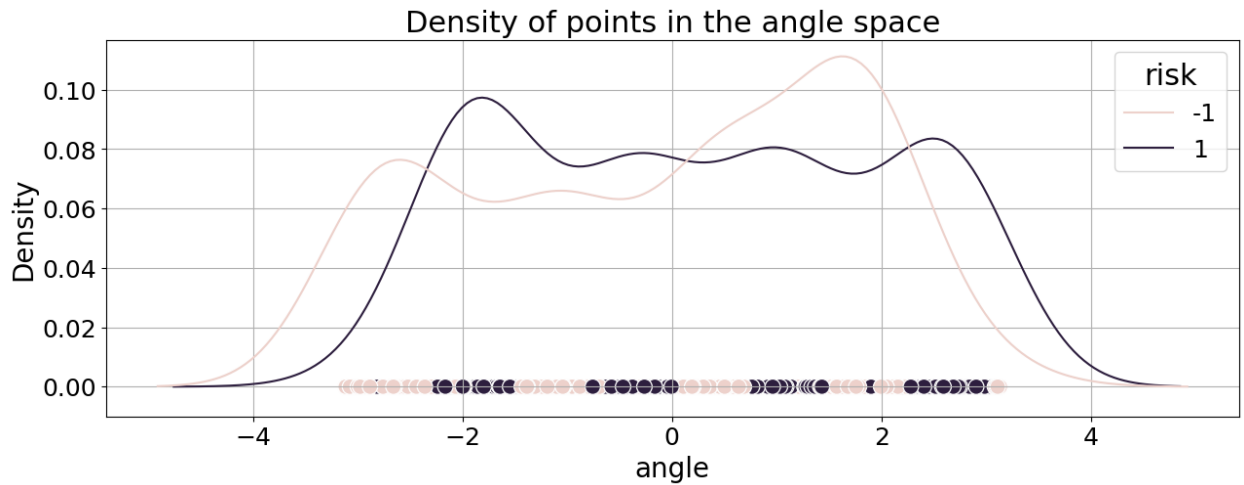
16.



The data seems to be divided in a manner not entirely dissimilar to a pizza 🍕, or a peppermint candy 🌀. Either way, seems delicious.

It seems like **the angle plays the most important factor** regarding the classification of each datapoint.

17. Below is a representation of the points in 1D-angle space:

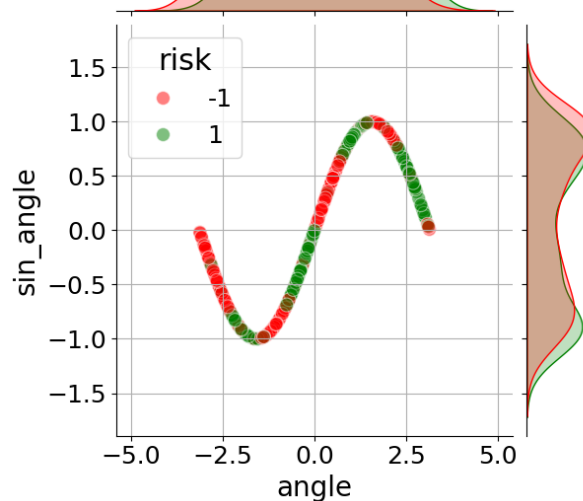


The graphs describe the density of each label as a function of the angle. It can be observed that there's a substantial difference in the densities of each label for most angles.

While the angle feature is separable by angle intervals, it is **not linearly separable**, as any possible threshold provides low classification accuracy.

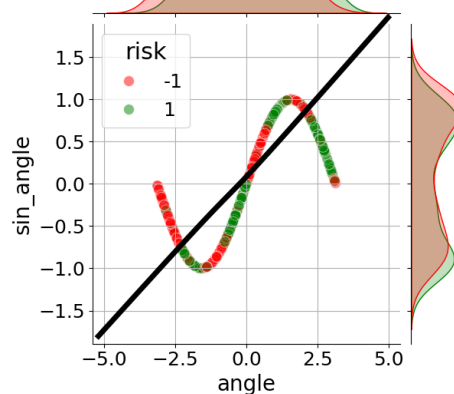
18. Below is the new graph:

Mapping of points to (angle, sin_angle) space



After applying the transformation, the two features (*angle*, *sin_angle*) seem closer to being linearly separable, as we can draw a linear line so that most red points are above the line, and most green points are below it.

Mapping of points to (angle, sin_angle) space



However, this line **does not provide us with great linear separation** (and more generally, the data is not linearly separable), as there are some misclassified points remaining (other than the obvious outliers).

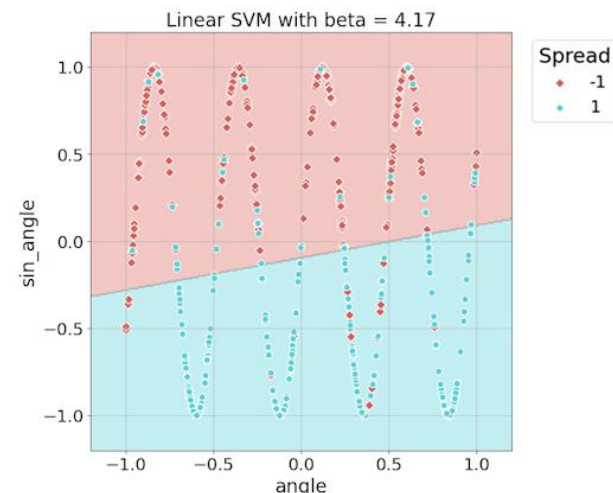
19. Our suggestion is to attempt to fit the sine curve such that the **intersection with the x axis (angle) will be where the dominating label changes**. It can be seen in the Q17 graph that there are 8 main areas, all of them spanning across the same “length” (angle ranges of approximately $\frac{\pi}{4}$).

We have two (close) suggestions for β values:

- 1) $\beta = 4$, **based on observation** of the data. We think that a round β value provides an accurate model of the data’s meaning and preserves the continuity between the minimum and maximum angles (which should probably represent “nearby” datapoints, since the angle $-\pi$ is the same as π in radians).
- 2) $\beta = 4.17$, **based on an exhaustive search**: we tested each β value between 1 and 6 with step size of 0.01, evaluating quality of different β values using 5-cross-validation.

We decided to use $\beta = 4.17$, as it gave significantly better results. We note that it’s possible that this value could be the product of overfitting, as it was acquired experimentally. Logic dictates that a value of 4 should perform better, as we discussed earlier (fits better with the underline meaning of the sin of the angle).

However, this value provides significantly better train accuracy and test accuracy compared to $\beta = 4$, and we wanted to demonstrate our extra effort :)



Training accuracy: 0.861.

Test accuracy: 0.863.

The model performs significantly **better than the given model** as expected, as it utilizes the observation that labels are mostly determined by the angle (and we transformed the data so that it’d be linearly separable using the transformed angles)