CIS 419/519: Applied Machine Learning Fall 2024

# Homework 2

*Handed Out:* *Due: 7:59 pm October 23*

**Name:** Richard Zhang

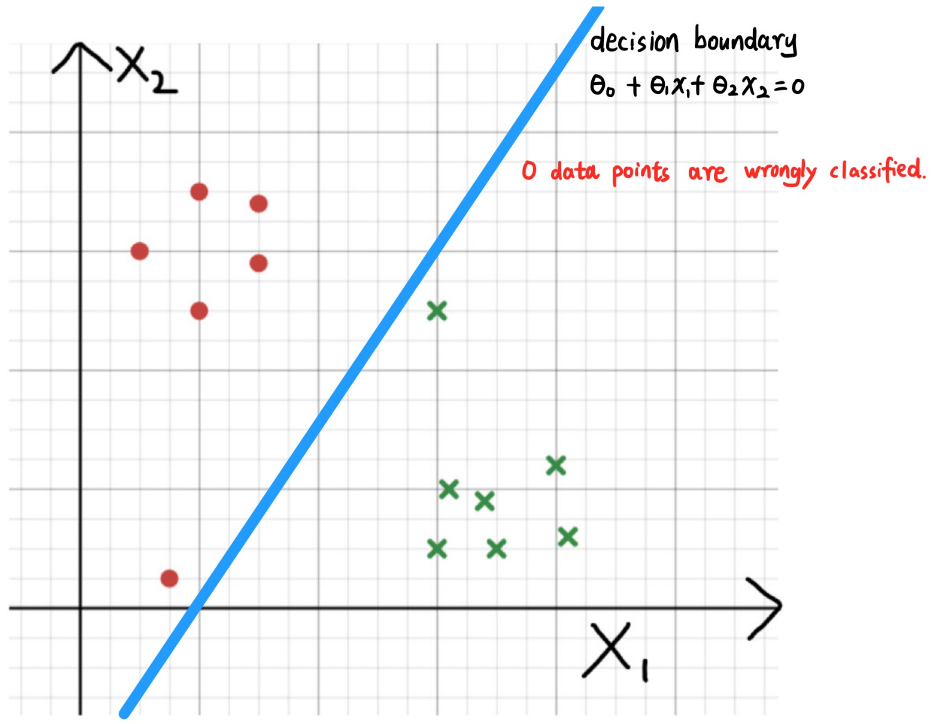
**PennKey:** zhank24

**PennID:** 19331985

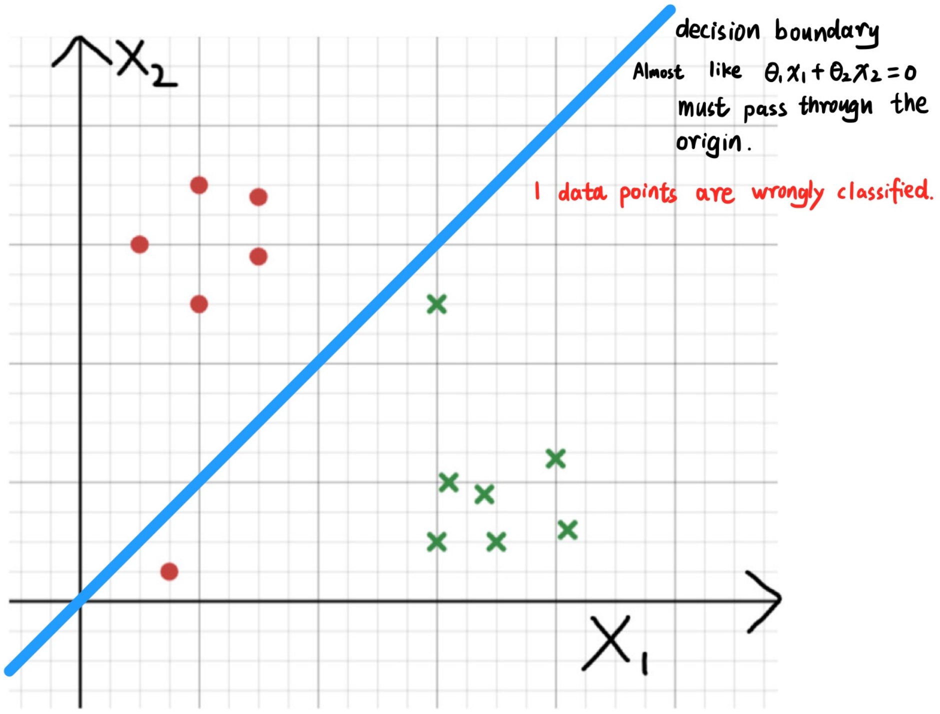
## 1 Declaration

- **Person(s) discussed with:** *N/A*
- **Affiliation to the course: student, TA, prof etc.** *student*
- **Which question(s) in coding / written HW did you discuss?** ***N/A***
- **Briefly explain what was discussed.** *N/A*

## 2 Question 1

1. 

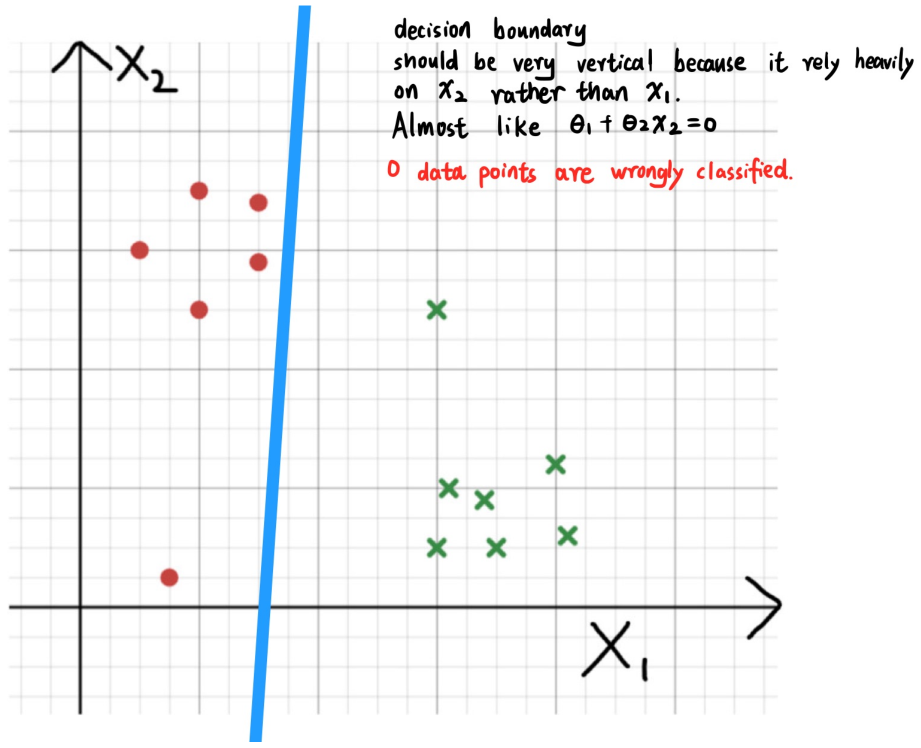
2.



decision boundary
Almost like $\theta_1 x_1 + \theta_2 x_2 = 0$
must pass through the origin.

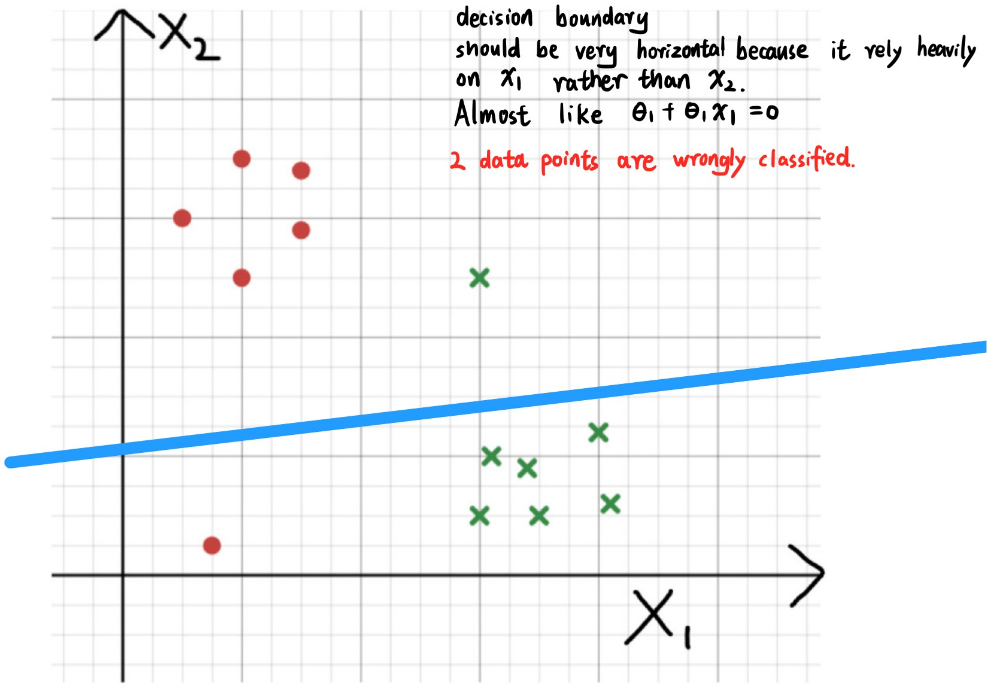
1 data points are wrongly classified.

3.



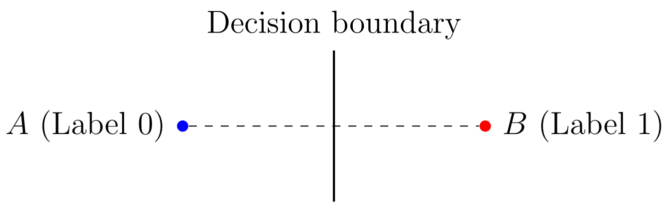
decision boundary
should be very vertical because it rely heavily on $x_2$ rather than $x_1$.
Almost like $\theta_1 + \theta_2 x_2 = 0$

0 data points are wrongly classified.

4. 

## 3 Question 2

1. There are only 2 points, so the boundary is the line perpendicular to the line segment joining the two points.



2. Yes, when k is 1, the prediction for every data point is based on the label of its closest neighbor. And, because the number of data points is infinite, we can put all the training points very close to any test point is appropriately labeled. In this way, every point would fall on each side of any complex (any even non-linear) decision boundary. In other words, we can control which class a test point would be assigned to by placing training points. For instance, if we have infinite data points and want to have a decision boundary whose shape is like a parabola, we can place training points all close to the parabola, assigning test points inside the parabola to one class and points above the parabola the other class. In this way, we can create a decision boundary.

3. Constant function. In such case, all the data points are neighbors, and the constant function would always predict the majority of the data labels, regardless of the inputs.

4. Based on question b and c, increasing $k$ would **increase bias** because the model would be less sensitive to local variations and may lead to under-fitting. Also, increasing $k$

would **decrease variance**, therefore reducing noises and the possibility of over-fitting.

5. I think we can have a weighted voting scheme to adjust the trade-off between TPR and TNR. Specifically, we can let the neighbors closer to test points have more weight (influence) on the prediction, including using square distance or cubic distance to be the voting scheme. In such case, the scheme gives more weight on closer training points so that we can adjust the trade-off based on our preference.

## 4 Question 3

1. Firstly, early stopping conditions can prevent a decision tree from growing dramatically without constraints. This can be computationally expensive and cost lots of time to train. Thus, adopting early stopping conditions can reduce the potential size of decision tree and save time and resources before the training stage starts.

   Also, large and deep trees without early stopping conditions may have nodes with few samples, and this would lead to over-fitting and higher variance, making the model's predictions unreliable. Early stopping conditions can prevent this situation by setting the minimum number of samples in a node. In such way, the data wouldn't be split into many small subsets and lose its reliability.

   Furthermore, post-pruning might be hard to achieve when a decision tree is very complex and deep because it requires significant amount of time and computational resources to evaluate lots of branches. Therefore, if we can restrict the tree's growth at the beginning, the post pruning stage would be more smooth and manageable.

2. (a) **increases variance**: if a tree grows deeper, its complexity increase, thus leading to potential over-fitting and higher variance.

   (b) **increases bias**: by increasing the number of samples needed to split a node, we reduce the tree's complexity and decrease bias (may lead to under-fitting).

   (c) **increases variance**: the purpose of post pruning is to decrease useless branches and avoid over-fitting. Thus, disabling post-pruning would lead to the increase of variance.

   (d) **increases variance**: having more features would increase a tree's capability to fit its samples and may therefore lead to over-fitting and the increase of variance.

## 5 Question 4

1. Based on the question, we can easily deduce that $P(Y = T) = \frac{3}{4}$ and $P(Y = F) = \frac{1}{4}$. Since we know $H(Z) = -\sum_y P(Y = y) \log_2 P(Y = y)$, we can deduce:

$$H(Z) = -(\frac{3}{4}\log_2\frac{3}{4} + \frac{1}{4}\log_2\frac{1}{4}) = 0.811$$

   Thus, the entropy of the collection is **0.811**.

2. We know $a_1 = +$: 3 and 4 (T, T), and $a_1 = -$: 1 and 2 (F, T), so:

$$H(Z|a_1 = +) = -(1\log_2 1 + 0\log_2 0) = 0$$

$$H(Z|a_1 = -) = -(\frac{1}{2}\log_2\frac{1}{2} + \frac{1}{2}\log_2\frac{1}{2}) = 1$$

$$H(Z|a_1) = (\frac{2}{4} \times 0) + (\frac{2}{4} \times 1) = 0.5$$

we know $IG(Z, j, t) = H(Z) - [H(Z \mid x_j = t)P(x_j = t) + H(Z \mid x_j \neq t)P(x_j \neq t)$, so $a_1$'s information gain is:

$$IG(Z, a_1) = 0.811 - 0.5 = 0.311$$

Similarly, $a_2 = +$: 1, 3, 4 (F, T, T), and $a_2 = -$: 2 (T).

$$H(Z|a_2 = +) = -(\frac{2}{3}\log_2\frac{2}{3} + \frac{1}{3}\log_2\frac{1}{3}) = 0.918$$

$$H(Z|a_2 = -) = -(1\log_2 1 + 0\log_2 0) = 0$$

$$H(Z|a_2) = (\frac{3}{4} \times 0.918) + (\frac{1}{4} \times 0) = 0.6885 \approx 0.689$$

$a_2$'s information gain is:

$$IG(Z, a_2) = 0.811 - 0.689 \approx 0.122 \text{ bits}$$

Information Gain for $a_1$: **0.311**

Information Gain for $a_2$: **0.122**

3.
```
if (a1 == +) {
    Classification = T;
} else {
    // the situation in which a1 == -
    if (a2 == +) {
        Classification = F;
    } else { // a2 == -
        Classification = T;
    }
}
```

## 6 Question 5

1. K Nearest Neighbors assumes the houses with similar feature values, such as similar square footage and the number of bedrooms, would have similar prices.

2. Euclidean distance is not suitable in this case because the numerical values of square footage and the number of bedrooms are quiet different (their scales aren't the same. i.e. 3,000 square feet and 3 bedrooms). This could lead to the situation where square footage would have much stronger influence on Euclidean distance calculation and make the number of bedrooms paly much less role in predicting houses' prices.

   To tackle this, we can simply apply normalization and standardization to normalize the two features so that both of them can have equal influence in the prediction.

3. Definitely no. The type of roof is just a categorical variables, and its numbers are just IDs. Using them would dramatically distort the calculation of Euclidean distance.

4. $O(n \cdot d)$ should be the complexity of 1-NN, because for every point that need to be predicted, we need to calculate its distance to all the n training examples, and every of them involves d dimensions. Thus, the complexity is $O(n \cdot d)$.

5. The curse of dimensionality: Adding more dimensions would hurt the performance of k-NN because the space (a high dimensional space) is now so large that all points in any finite dataset are likely to be very far apart. In such scenario, "closest points" are almost as far away as the farthest away points. When "nearest neighbors" are far away, predictions are poor.

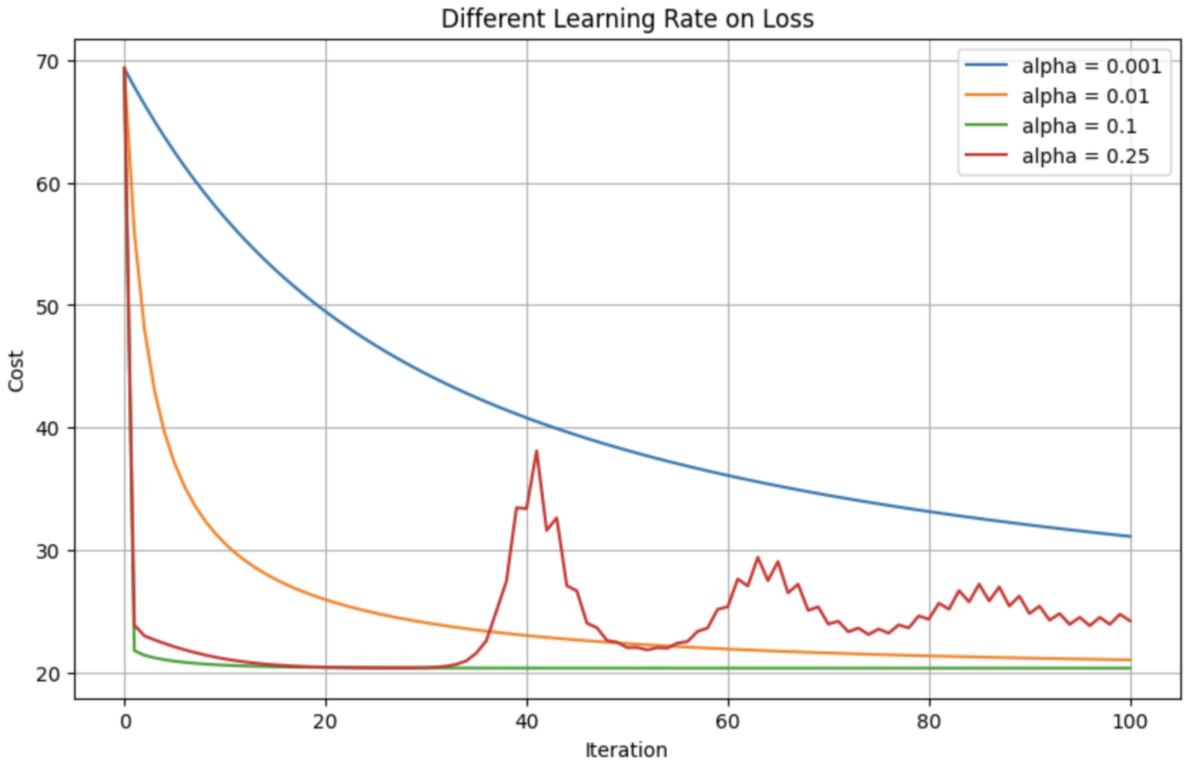
   However, k-NN can have good performance on datasets such as handwritten digits (MNIST) because useful data usually live in a lower-dimensional manifold within a high-dimensional feature space. In other words, although a high-dimensional space has lots of features, such as the pixels in an image, the actual variability of the data we want to investigate are living in a much smaller and manageable subset of dimensions. In such case, the local neighborhoods is still a very meaningful and distinct way to identify similar samples. In this case, k-NN can still predict a data point based on its proximity to other points in a low-dimensional setting. Using this technique, k-NN can detect local neighborhoods and have an effective similarity measure.

## 7 Question 6

1. False

2. (a) anywhere in between the above two choices.
   (b) $K^2$
   (c) False
   (d) False

## 8 Python Programming Question 1.2

1. 

2. When alpha = 0.001, the loss decreases steadily but super slowly, implying the learning process is stable but too slow. Although the model converges, it is too slow because the learning rate is too low. So, we won't choose it as the best one.

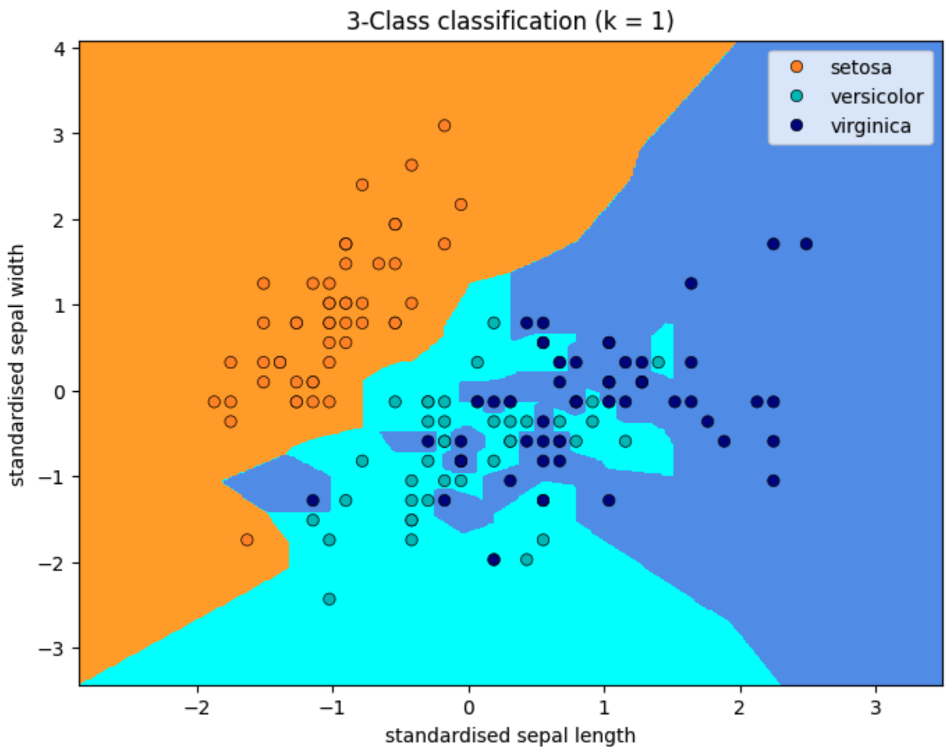
   When alpha = 0.01, the loss decreases faster, and it converges smoothly. This implies this model reaches the minimum efficiently without large oscillations. However, we can still improve it, so we won;t choose this either.

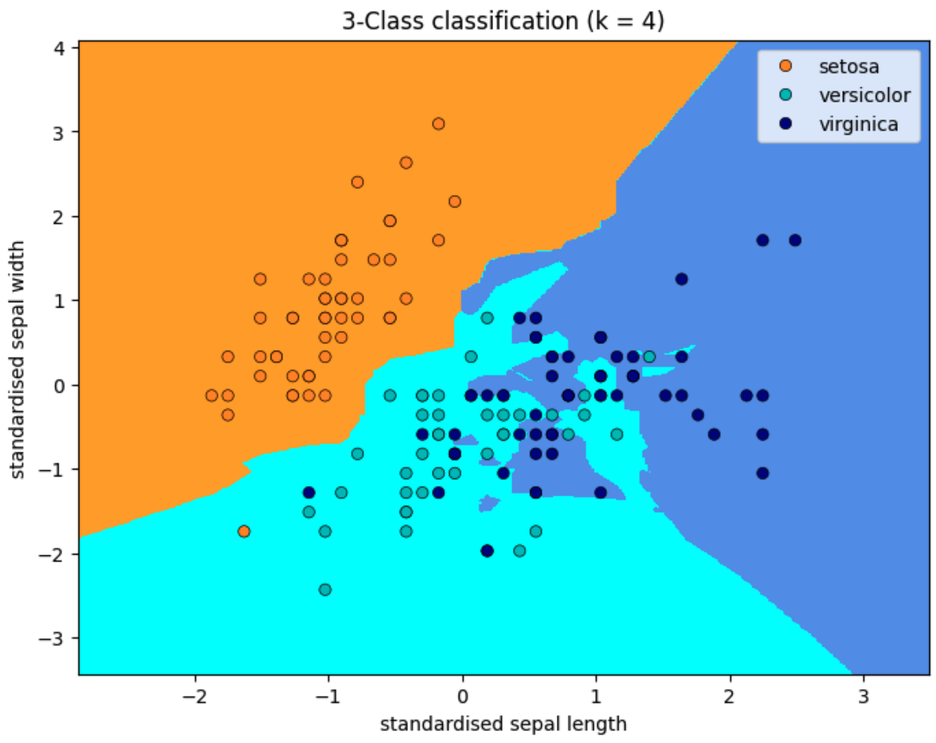
   When alpha = 0.1, the loss reduces quickly without any oscillation after some iterations. It converges very quickly and has stability without any oscillation. Thus, this is the best learning rate

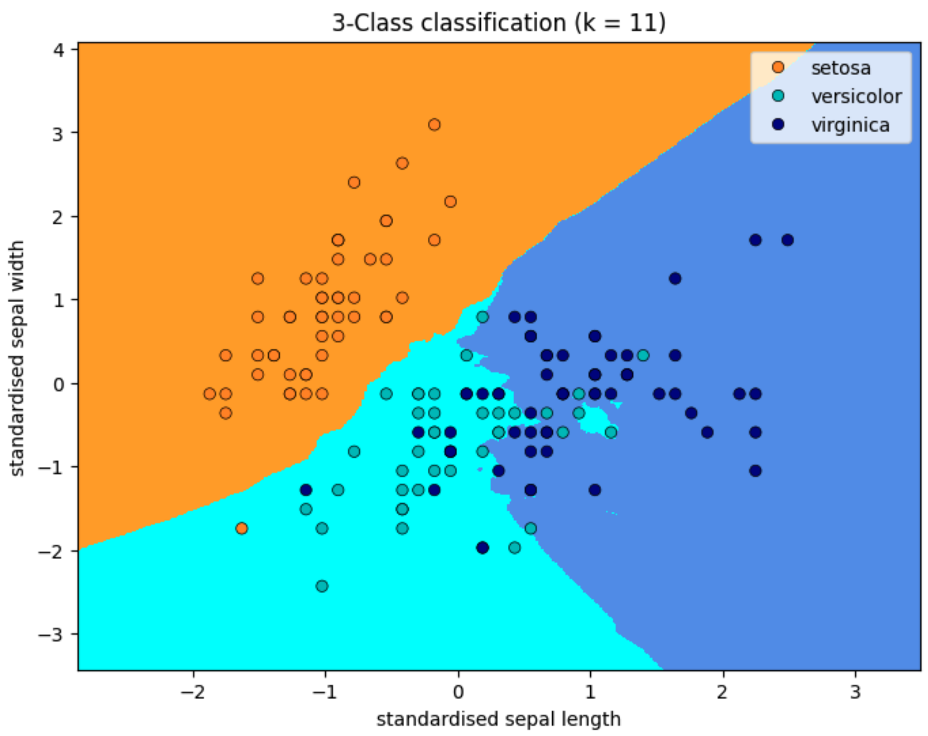
   When alpha = 0.25, the loss shows a significant oscillating pattern and fails to converge. This suggests the learning rate is too high and makes gradient descent to overshoot the optima. It's very unstable. Thus, we won't choose it.

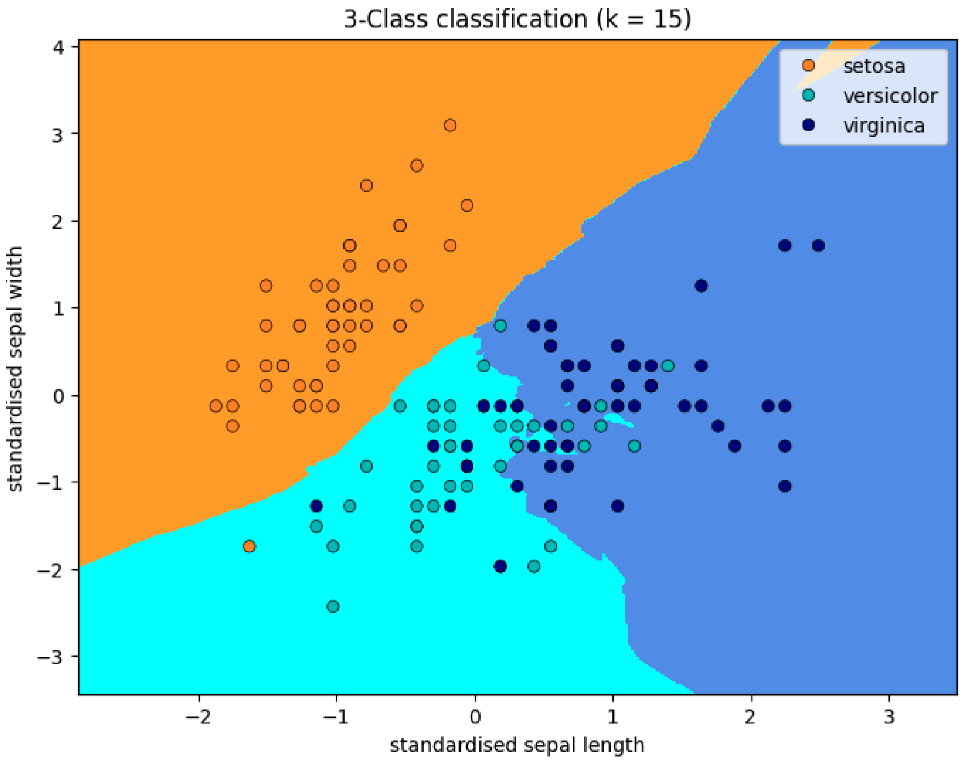
   **Therefore, alpha = 0.1 is the best learning rate**

## 9 Python Programming Question 2.3

1. 

2. 

3. 

4. 

5. When we have a small value of k, such as k=1, the model over-fit data. This is because it captures every local pattern and noise from the training data. As a result, each point influences the decision boundary and make it to a very irregular and complex boundaries that perfectly match the training set. In this case, the model cannot

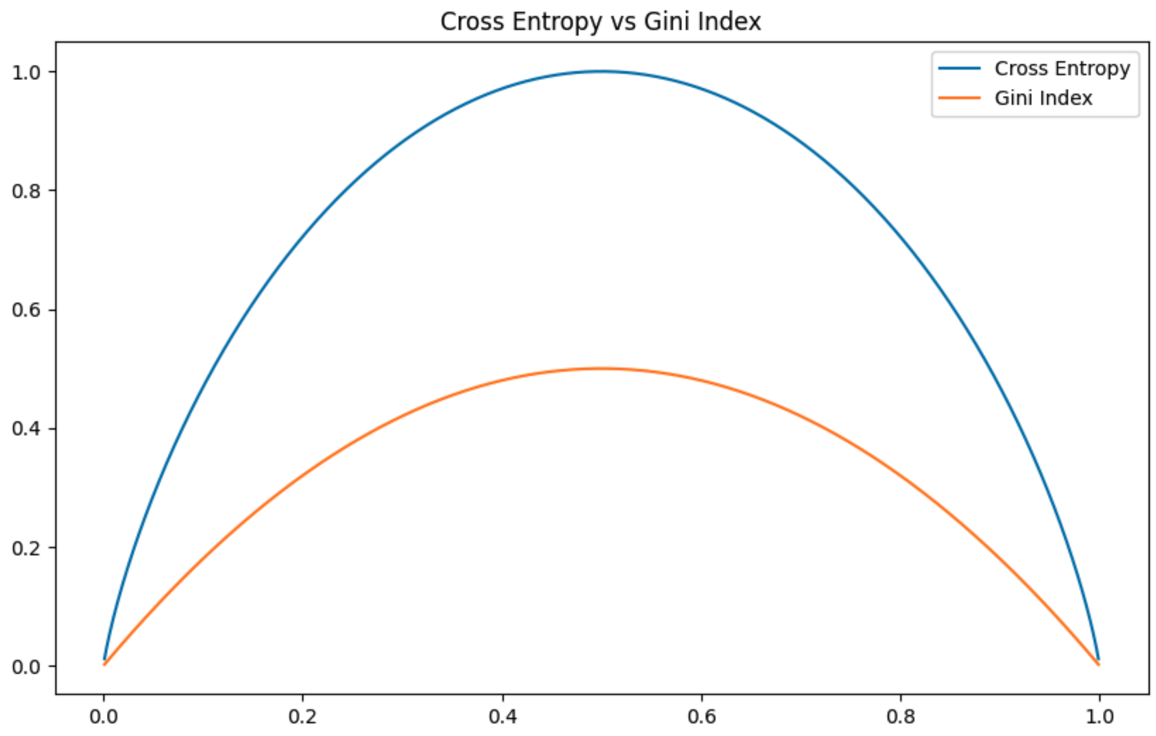
generalize well to new unseen test data, which can lead to poor performance on testing sets. The decision boundaries in this scenario very sharp and irregular.

When k is an intermediate value, such as k=4 or k=11, the model starts to perform better in terms of generalization. The decision boundaries become smoother, and the model is less influenced by individual noisy points. The boundaries is stable and exhibit a good balance between bias and variance. This is the perfect balance between variance and bias.

When k continues to increase, such as k=15, the model would have the issue of under-fitting. With too many neighbors being considered, the classifier may over-smooth the decision boundaries, leading to a loss of important details and distinctions between classes. In this case, the decision boundaries become very smooth, but the model could fail to capture important patterns within the data and cause under-fitting.

Therefore, we can say a small k leads to over-fitting, while a too large k can lead to under-fitting by oversimplifying lots of subtle patterns. Choose an intermediate value of k is the key to maintain the balance between variance and bias.

## 10 Python Programming Question 3.1.2

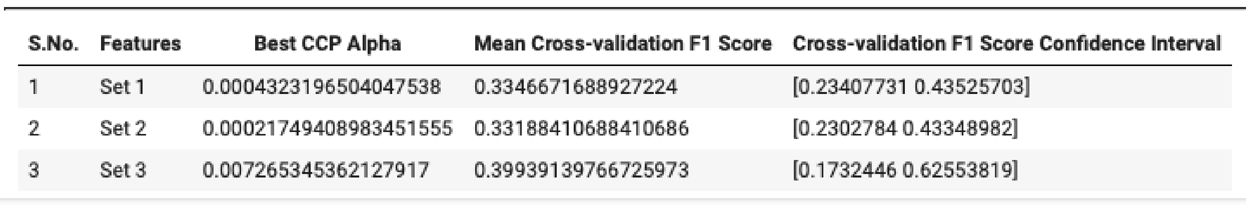
1. 

2. Cross entropy is sensitive to class imbalances. Based on the plot, the curve for cross entropy becomes steeper when the probability values are close to 0 or 1. This means cross entropy is even more sensitive to small changes in probability when one class is almost certain. Cross entropy reaches its maximum impurity at a 50-50 class split, which maximum uncertainty, just like the Gini index does.

Compared to cross entropy, the Gini index has a more smooth and less steep curve. Based on the plot, we can notice the Gini index is less sensitive to extreme probabilities (like 0 or 1). Like cross entropy, the Gini index also peaks at a probability of 0.5.

Both of them are very effective measures because they can capture the uncertainty (impurity) of a node and indicate how uncertain the classification at a given node is. Both cross entropy and the Gini index reach their maximum value when the probability of each class is equal. Also, they can show impurity is minimized when one class has a probability of 1, implying the node is completely pure without uncertainty. We can easily observe that cross entropy penalizes impurity more heavily than the Gini index does. Although they are different, both of them are suitable measures of impurity based on the plot.

## 11 Python Programming Question 4.5

1. 

| S.No. | Features | Best CCP Alpha | Mean Cross-validation F1 Score | Cross-validation F1 Score Confidence Interval |
|---|---|---|---|---|
| 1 | Set 1 | 0.0004323196504047538 | 0.3346671688927224 | [0.23407731 0.43525703] |
| 2 | Set 2 | 0.00021749408983451555 | 0.33188410688410686 | [0.2302784 0.43348982] |
| 3 | Set 3 | 0.007265345362127917 | 0.39939139766725973 | [0.1732446 0.62553819] |

2. **Set3 is the best set**

   Set 3 has the highest mean F1 score of 0.39939 with a larger confidence interval. This suggests that Set 3 has a strong average performance, even though the results fluctuate more significantly. This implies Set 3 provides a balanced measure between precision and recall, both of which are both critical in this question (medical predictions).

   Set 1 and Set 2 cannot compete with Set 3 in terms of F1 mean although they have more narrow confidence intervals. Thus, we choose Set 3 as the best set.

## 12 Python Programming Question 6.3

Logistic regression is simple, straightforward, and easy to use and interpret. We can quickly understand feature importance by using logistic regression. However, it assumes a linear relationship between features and may under-perform when non-linear patterns exist. When hyper-parameters are not tuned, it can still work well and generalize. Thus, we would do logistic regression when hyper-parameters are not tuned.

Neural networks are flexible and can model very complex and non-linear relationships. However, they require significant hyper-parameter tuning and could lead to over-fitting if if the dataset is pretty small (like the question we have here). So, we would do neural networks only if we can do proper tuning, have a large dataset, and need to handle complex interactions between variables.

Decision trees capture non-linear relationships and handle both numerical and categorical data without much pre-processing. They are easy to interpret but prone to over-fitting, particularly without pruning. It doesn't really require a large dataset or fine tuning. However, without some ensemble methods (like random forest), its performance may not be as good as the other two.

Therefore, I think when hyper-parameters are not tuned, logistic regression and decision trees are the best to predict diabetes (if I have to choose one from the two, I would say **logistic regression**). With proper tuning, **neural networks** is the best, especially if our data size is large and has complex feature interactions.