Assignment 4 - Machine Learning
Theoretical Questions

Mohammad Hossein Basouli

June 4, 2025

Question 1

(a) You have essentially removed the constraint for how great the magnitude of the parameters ϵ_i's can become. This causes the model to allow arbitrarily large errors, which leads to poor accuracy.

(b) I think so. If we set the w such that its norm is so small, then the margin would be so large, but also we have to allow arbitrarily large errors ϵ_i's.

Question 2

(a) It controls the effect of linear terms over the other higher order terms. If we set $c = 0$, we have essentially removed the linear terms in our feature space, whereas, if we set $c = 1$, we add linear terms based on our old feature space, to the new feature space. See the illustration in Figure 1.

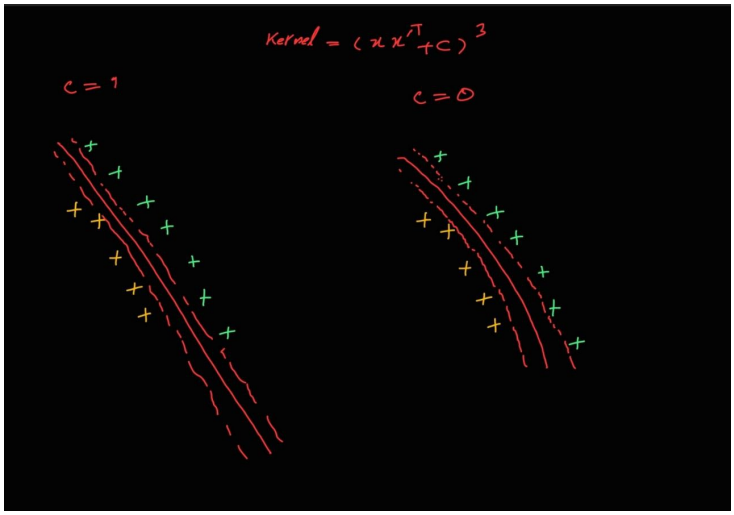

Figure 1: The figure illustrates the effect of inclusion and exclusion of constant term in the *polynomial kernel*; we can see that, by including the constant term (right section), the model has overfitted to the training data. But in the left side, we can see a model which has the constant term excluded, which leads to a better fit.

(b) We can decrease both (C and γ).

(c) Figure 2

$$K(x, x') = \exp\left(-\gamma \|x - x'\|^2\right)$$

$$f(x) = \sum_{i=1}^{n} \alpha_i y_i \exp\left(-\gamma \|x - x_i\|^2\right) + b$$

Solid lines (decision boundary): $f(x) = 0$

Dashed lines (margin boundaries): $f(x) = \pm 1$

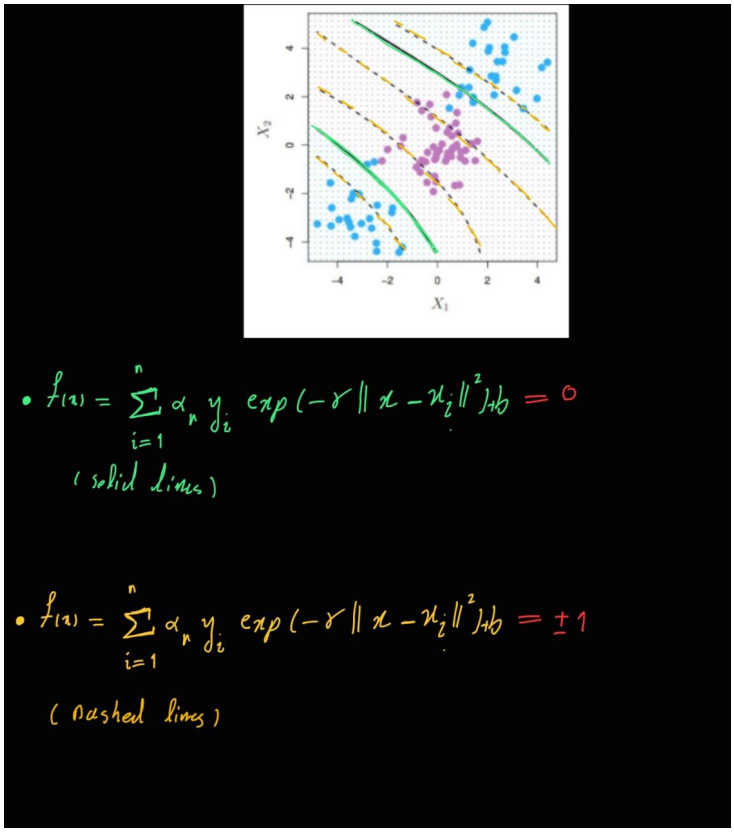

Figure 2: The figure shows the equation for solid and dashed lines in SVM using *RBF kernel*.

Question 3

In pre-pruning, we prune the tree, as it grows by checking some criterion such as:

- Is a minimum number of samples reached in a node?
- Is a maximum depth reached?
- Does Information Gain increase enough?

Whereas in post-pruning, we prune the tree, after it has fully grown, by removing the splittings that add low to no true splitting in the data. Some of the methods for this are:

- Reduced Errro Pruning
- Cost Complexity Pruning

Aspect	Pre-Pruning	Post-Pruning
Bias	High (due to early stopping)	Moderate (more flexible)
Variance	Lower (less overfitting)	Moderate (prunes overfit tree)
Computational Cost	Lower (stops tree early)	Higher (full growth + pruning)

Table 1: Comparison of Pre-Pruning and Post-Pruning

Question 4

Yes, the Gini impurity of an individual child node can be greater than that of its parent. However, the weighted average of the child nodes' Gini impurities will always be less than or equal to the Gini impurity of the parent node.
Parent Node:
Class A: 30, Class B: 20

$$p_A = \frac{30}{50} = 0.6, \quad p_B = 0.4$$

$$\text{Gini}_{\text{parent}} = 1 - (0.6^2 + 0.4^2) = 1 - (0.36 + 0.16) = 0.48$$

After Split:
Left Child:
Class A: 20, Class B: 10

$$p_A = \frac{20}{30} \approx 0.67, \quad p_B \approx 0.33$$

$$\text{Gini}_L = 1 - (0.67^2 + 0.33^2) \approx 1 - (0.4489 + 0.1089) = 0.4422$$

Right Child:
Class A: 10, Class B: 10

$$p_A = p_B = 0.5$$

$$\text{Gini}_R = 1 - (0.5^2 + 0.5^2) = 1 - (0.25 + 0.25) = 0.5$$

Weighted Gini after split:

$$\text{Gini}_{\text{split}} = \frac{30}{50} \cdot 0.4422 + \frac{20}{50} \cdot 0.5 = 0.264 + 0.2 = 0.464$$

Conclusion: Even though $\text{Gini}_R > \text{Gini}_{\text{parent}}$, the overall weighted Gini impurity decreased: $0.464 < 0.48$.

Question 5

Figure 3

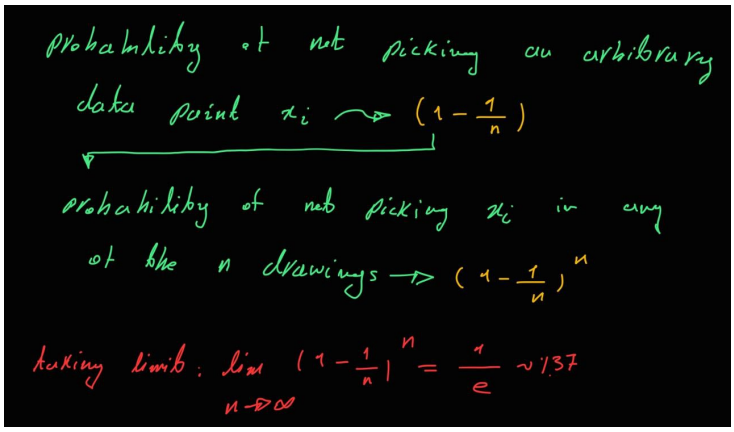

Figure 3: Justification for why, each bootstrapped dataset contains only 63 percent of the original samples.

Question 6

(a) Answer: **AdaBoost** in Short: AdaBoost first gives equal weights to each of the samples in the dataset, and then tries to fit the best stump to the data. Then it calculates the errors and updates the weights for each of the samples, based on how much error the model has made for that sample. Next, it will calculate how much error the model has made overall to give the model's prediction a weight. And it repeats the process for a certain number of iterations and then sum them up.

Gradient Boosting in Short: Unlike AdaBoost, in Gradient Boosting, instead of assigning weight to the data points, we will use the negative of the gradient of the loss function to build new models that correct the previous errors, made by the older models.

Advantages of the AdaBoost over Decision Trees & Random Forests:

- Lower Bias
- More Flexible (because it has more hyperparameters)
- Higher Accuracy
- Less Prone to Overfitting than Decision Trees

Algorithm 1 AdaBoost Algorithm

Require: Training set $\{(x_1, y_1), \ldots, (x_n, y_n)\}$ with $y_i \in \{-1, +1\}$
Require: Number of rounds T
1: Initialize weights $w_i^{(1)} = \frac{1}{n}, \quad i = 1, \ldots, n$
2: **for** $t = 1$ to T **do**
3: Train weak learner $h_t(x)$ using distribution $w^{(t)}$
4: Compute error: $\varepsilon_t = \sum_{i=1}^{n} w_i^{(t)} \cdot \mathbb{I}(h_t(x_i) \neq y_i)$
5: Compute model weight: $\alpha_t = \frac{1}{2} \ln\left(\frac{1-\varepsilon_t}{\varepsilon_t}\right)$
6: Update weights:

$$w_i^{(t+1)} = w_i^{(t)} \cdot \exp(-\alpha_t y_i h_t(x_i)) \quad \text{for } i = 1, \ldots, n$$

7: Normalize weights so that $\sum_{i=1}^{n} w_i^{(t+1)} = 1$
8: **end for**
9: **return** Final classifier: $H(x) = \text{sign}\left(\sum_{t=1}^{T} \alpha_t h_t(x)\right)$

Algorithm 2 Gradient Boosting Algorithm

Require: Training set $\{(x_1, y_1), \ldots, (x_n, y_n)\}$
Require: Differentiable loss function $L(y, F(x))$
Require: Number of iterations T
1: Initialize model: $F_0(x) = \arg\min_\gamma \sum_{i=1}^{n} L(y_i, \gamma)$
2: **for** $t = 1$ to T **do**
3: Compute pseudo-residuals:

$$r_i^{(t)} = -\left[\frac{\partial L(y_i, F(x_i))}{\partial F(x_i)}\right]_{F=F_{t-1}} \quad \text{for } i = 1, \ldots, n$$

4: Fit a weak learner $h_t(x)$ to the residuals $r_i^{(t)}$
5: Compute step size:

$$\gamma_t = \arg\min_\gamma \sum_{i=1}^{n} L(y_i, F_{t-1}(x_i) + \gamma h_t(x_i))$$

6: Update model:

$$F_t(x) = F_{t-1}(x) + \gamma_t h_t(x)$$

7: **end for**
8: **return** Final model: $F_T(x)$

Advantages of the GradientBoosting over Decision Trees & Random Forests:

- Lower Bias
- More Flexible (because it has more hyperparameters)
- Higher Accuracy
- Less Prone to Overfitting than Decision Trees

Where to Use AdaBoost over GradientBoosting:

- Faster Training
- Data is Not Extremly Noisy

Where to Use GradientBoosting over AdaBoost :

- We Want More Flexibility; Like Working with Different Loss Functions
- The Data is Nosiy (it can handle it by regularization)

(b) Answer:

- **AdaBoost**: Increase the *number of estimators* - Increase the *learning rate*.
- **GradientBoosting**: Decrease the *number of estimators* - Decrease the *learning rate* - Decrease *max depth* - Increase *min samples leaf* - Increase *min samples split*

(c) Answer: Some methods such as **XGBoost**, **LightGBM** and **CatBoost** handle categorical features natively, whereas the others, required them to be converted via *OneHotEncoding*, *LabelEncoding*, etc.