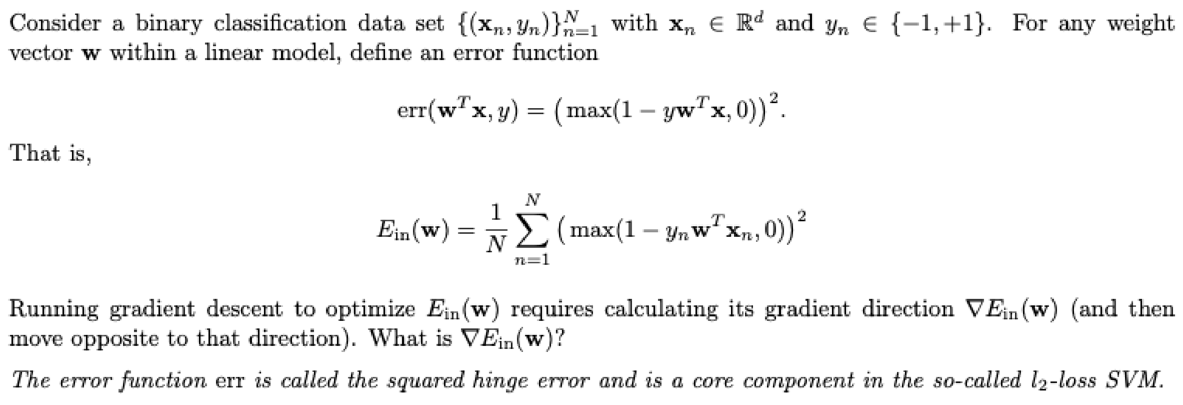
FAI 2024 HW3 Report

EE4, B09602017, Tsung-Min Pai

May 15, 2024

1 Written Part

1.1 Problem 1



Consider a binary classification data set $\{(\mathbf{x}_n, y_n)\}_{n=1}^N$ with $\mathbf{x}_n \in \mathbb{R}^d$ and $y_n \in \{-1, +1\}$. For any weight vector $\mathbf{w}$ within a linear model, define an error function

$$\text{err}(\mathbf{w}^T\mathbf{x}, y) = \big(\max(1 - y\mathbf{w}^T\mathbf{x}, 0)\big)^2.$$

That is,

$$E_{\text{in}}(\mathbf{w}) = \frac{1}{N}\sum_{n=1}^{N}\big(\max(1 - y_n\mathbf{w}^T\mathbf{x}_n, 0)\big)^2$$

Running gradient descent to optimize $E_{\text{in}}(\mathbf{w})$ requires calculating its gradient direction $\nabla E_{\text{in}}(\mathbf{w})$ (and then move opposite to that direction). What is $\nabla E_{\text{in}}(\mathbf{w})$?

The error function err *is called the squared hinge error and is a core component in the so-called l_2-loss SVM.*

Answer: Since we need to do the partial derivative to $\mathbf{w}$. The gradient of the squared hinge error:

$$\nabla_{\mathbf{w}}\text{err}(\mathbf{w}^T\mathbf{x}, y) = -2\mathbf{x}y\max(1 - y\mathbf{w}^T\mathbf{x}, 0)$$

Therefore, the gradient of the in-sample error is:

$$\nabla E_{\text{in}}(\mathbf{w}) = \frac{1}{N}\sum_{n=1}^{N}\nabla_{\mathbf{w}}\text{err}(\mathbf{w}^T\mathbf{x}_n, y_n) \tag{1}$$

$$= -\frac{2}{N}\sum_{n=1}^{N}\mathbf{x}_n y_n\max(1 - y_n\mathbf{w}^T\mathbf{x}_n, 0) \tag{2}$$

1.2 Problem 2

Consider a process that generates d-dimensional vectors $\mathbf{x}_1, \mathbf{x}_2, \ldots, \mathbf{x}_N$ independently from a multivariate Gaussian distribution $\mathcal{N}(\mathbf{u}, \mathrm{I})$, where $\mathbf{u} \in \mathbb{R}^d$ is an unknown parameter vector and $\mathrm{I} \in \mathbb{R}^{d \times d}$ is an identity matrix. The maximum likelihood estimate of $\mathbf{u}$ is

$$\mathbf{u}^* = \arg\max_{\mathbf{u} \in \mathbb{R}^d} \prod_{n=1}^{N} p_{\mathbf{u}}(\mathbf{x}_n),$$

where $p_{\mathbf{u}}$ is the probability density function of $\mathcal{N}(\mathbf{u}, \mathrm{I})$. Prove that

$$\mathbf{u}^* = \frac{1}{N} \sum_{n=1}^{N} \mathbf{x}_n.$$

The same derivation can be used to obtain the cross-entropy error function of logistic regression.

Answer: The probability density function of $\mathcal{N}(\mathbf{u}, \mathbf{I})$ is:

$$p_{\mathbf{u}}(\mathbf{x}) = \frac{1}{\sqrt{(2\pi)^d}} \exp\left(-\frac{1}{2}(\mathbf{x} - \mathbf{u})^T (\mathbf{x} - \mathbf{u})\right).$$

And the maximum likelihood estimation is then:

$$\begin{aligned}
\mathbf{u}^* &= \arg\max_{\mathbf{u} \in \mathbb{R}^d} \prod_{n=1}^{N} p_{\mathbf{u}}(\mathbf{x}_n) && (3)\\
&= \arg\max_{\mathbf{u} \in \mathbb{R}^d} \sum_{n=1}^{N} \log p_{\mathbf{u}}(\mathbf{x}_n) && (4)\\
&= \arg\min_{\mathbf{u} \in \mathbb{R}^d} \sum_{n=1}^{N} (\mathbf{x}_n - \mathbf{u})^T (\mathbf{x}_n - \mathbf{u}) && (5)\\
&= \arg\min_{\mathbf{u} \in \mathbb{R}^d} \sum_{n=1}^{N} f(\mathbf{u}), && (6)
\end{aligned}$$

where

$$f(\mathbf{u}) := \sum_{n=1}^{N} (\mathbf{x}_n - \mathbf{u})^T (\mathbf{x}_n - \mathbf{u}).$$

And to find $\mathbf{u}^*$, we need to equate the gradient of f computed at $\mathbf{u}^*$to zero:

$$\nabla_u f(\mathbf{u}^*) = -2 \sum_{n=1}^{N} (\mathbf{x}_n - \mathbf{u}^*) = 0.$$

As a result, the solution to the above equation is $\mathbf{u}* = \frac{1}{N} \sum_{n=1}^{N} \mathbf{x}_u$.

1.3 Problem 3

A classic binary classification data set that cannot be separated by any line is called the XOR data set, with

$\mathbf{x} = [x_1, x_2]$	y
$\mathbf{x}_1 = [+1, +1]$	$y_1 = -1$
$\mathbf{x}_2 = [-1, +1]$	$y_2 = +1$
$\mathbf{x}_3 = [-1, -1]$	$y_3 = -1$
$\mathbf{x}_4 = [+1, -1]$	$y_4 = +1$

You can see why it is called XOR by interpreting $+1$ as a boolean value of TRUE and -1 as FALSE. Consider a second-order feature transform $\Phi_2(\mathbf{x}) = (1, x_1, x_2, x_1^2, x_1x_2, x_2^2)$ that converts the data set to

$\mathbf{z} = \Phi_2(\mathbf{x})$	y
$\mathbf{z}_1 = \Phi_2(\mathbf{x}_1)$	$y_1 = -1$
$\mathbf{z}_2 = \Phi_2(\mathbf{x}_2)$	$y_2 = +1$
$\mathbf{z}_3 = \Phi_2(\mathbf{x}_3)$	$y_3 = -1$
$\mathbf{z}_4 = \Phi_2(\mathbf{x}_4)$	$y_4 = +1$

Show a perceptron $\tilde{\mathbf{w}}$ in the $\mathcal{Z}$-space that separates the data. That is,

$$y_n = \text{sign}(\tilde{\mathbf{w}}^T \mathbf{z}_n) \text{ for } n = 1, 2, 3, 4.$$

Then, plot the classification boundary

$$\tilde{\mathbf{w}}^T \Phi_2(\mathbf{x}) = 0$$

in the $\mathcal{X}$-space. Your boundary should look like a quadratic curve that classifies $\mathbf{x}_1, \mathbf{x}_2, \mathbf{x}_3, \mathbf{x}_4$ perfectly.

Answer: $\tilde{\mathbf{w}} = (0, 1, -1, 1, -3, 1)$ could be a solution of the perceptron. And the corresponding classification boundary would be

$$\tilde{\mathbf{w}}^{\mathbf{T}} \phi_{\mathbf{2}}(\mathbf{x}) = x_1 - x_2 + {x_1}^2 - 3x_1x_2 + {x_2}^2 = 0$$

And Figure 1 is the result of the boundary, which meets the requirement of the problem.

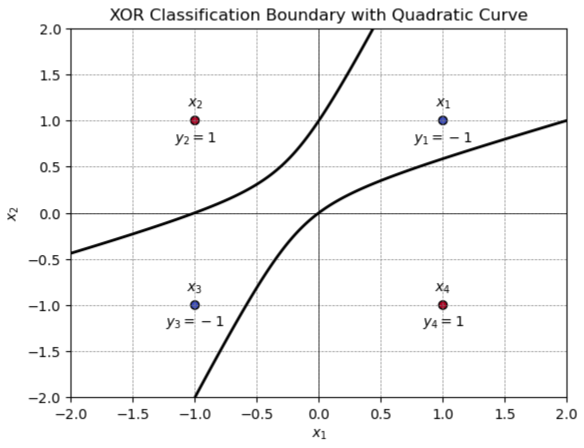
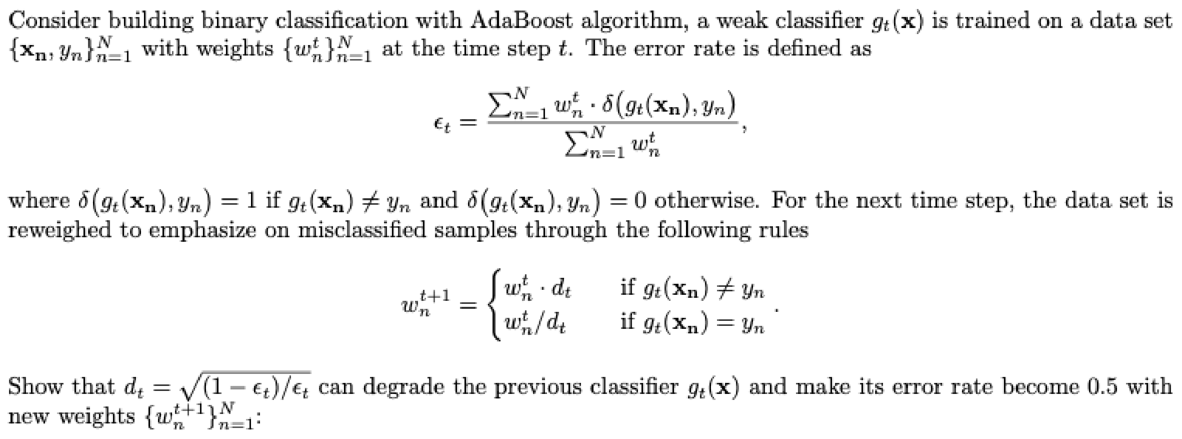


Figure 1: XOR Classification

1.4 Problem 4



Consider building binary classification with AdaBoost algorithm, a weak classifier $g_t(\mathbf{x})$ is trained on a data set $\{\mathbf{x_n}, y_n\}_{n=1}^N$ with weights $\{w_n^t\}_{n=1}^N$ at the time step t. The error rate is defined as

$$\epsilon_t = \frac{\sum_{n=1}^N w_n^t \cdot \delta\big(g_t(\mathbf{x_n}), y_n\big)}{\sum_{n=1}^N w_n^t},$$

where $\delta\big(g_t(\mathbf{x_n}), y_n\big) = 1$ if $g_t(\mathbf{x_n}) \neq y_n$ and $\delta\big(g_t(\mathbf{x_n}), y_n\big) = 0$ otherwise. For the next time step, the data set is reweighed to emphasize on misclassified samples through the following rules

$$w_n^{t+1} = \begin{cases} w_n^t \cdot d_t & \text{if } g_t(\mathbf{x_n}) \neq y_n \\ w_n^t / d_t & \text{if } g_t(\mathbf{x_n}) = y_n \end{cases}.$$

Show that $d_t = \sqrt{(1-\epsilon_t)/\epsilon_t}$ can degrade the previous classifier $g_t(\mathbf{x})$ and make its error rate become 0.5 with new weights $\{w_n^{t+1}\}_{n=1}^N$:

$$\frac{\sum_{n=1}^N w_n^{t+1} \delta\big(g_t(\mathbf{x_n}), y_n\big)}{\sum_{n=1}^N w_n^{t+1}} = 0.5.$$

Answer: Consider building binary classification with the AdaBoost algorithm, a weak classifier $g_t(x)$ is trained on a data set $\{(\mathbf{x}_n, y_n)\}_{n=1}^N$ with weights $\{w_n^t\}_{n=1}^N$ at the time step t. The error rate is defined as

$$\epsilon_t = \frac{\sum_{n=1}^N w_n^t \cdot \delta(g_t(\mathbf{x}_n), y_n)}{\sum_{n=1}^N w_n^t},$$

where $\delta(g_t(\mathbf{x}_n), y_n) = 1$ if $g_t(\mathbf{x}_n) \neq y_n$ and $\delta(g_t(\mathbf{x}_n), y_n) = 0$ otherwise. For the next time step, the data set is reweighted to emphasize on misclassified samples through the following rules

$$w_n^{t+1} = \begin{cases} w_n^t \cdot d_t & \text{if } g_t(\mathbf{x}_n) \neq y_n, \\ w_n^t / d_t & \text{if } g_t(\mathbf{x}_n) = y_n, \end{cases}$$

We need to show that $d_t = \sqrt{\frac{1-\epsilon_t}{\epsilon_t}}$ can degrade the previous classifier $g_t(x)$ and make its error rate become 0.5 with new weights $\{w_n^{t+1}\}_{n=1}^N$:

$$\frac{\sum_{n=1}^N w_n^{t+1} \cdot \delta(g_t(\mathbf{x}_n), y_n)}{\sum_{n=1}^N w_n^{t+1}} = 0.5.$$

First, let's calculate the new weights after reweighting:

$$\sum_{n=1}^N w_n^{t+1} = \sum_{g_t(\mathbf{x}_n)\neq y_n} w_n^t \cdot d_t + \sum_{g_t(\mathbf{x}_n)=y_n} w_n^t / d_t$$

Let's separate the sums for misclassified and correctly classified samples:

$$\sum_{g_t(\mathbf{x}_n)\neq y_n} w_n^t \cdot d_t = \epsilon_t \cdot \sum_{n=1}^N w_n^t \cdot d_t$$

$$\sum_{g_t(\mathbf{x}_n)= y_n} w_n^t / d_t = (1-\epsilon_t) \cdot \sum_{n=1}^N w_n^t / d_t$$

Combining these, we have:

$$\sum_{n=1}^N w_n^{t+1} = \epsilon_t \cdot \sum_{n=1}^N w_n^t \cdot d_t + (1-\epsilon_t) \cdot \sum_{n=1}^N w_n^t / d_t$$

Now, using $d_t = \sqrt{\frac{1-\epsilon_t}{\epsilon_t}}$:

$$\sum_{g_t(\mathbf{x}_n)\neq y_n} w_n^t \cdot d_t = \epsilon_t \cdot \sum_{n=1}^{N} w_n^t \cdot \sqrt{\frac{1-\epsilon_t}{\epsilon_t}}$$

$$\sum_{g_t(\mathbf{x}_n)= y_n} w_n^t / d_t = (1-\epsilon_t) \cdot \sum_{n=1}^{N} w_n^t / \sqrt{\frac{1-\epsilon_t}{\epsilon_t}}$$

Combining these, we get:

$$\sum_{n=1}^{N} w_n^{t+1} = \epsilon_t \cdot \sum_{n=1}^{N} w_n^t \cdot \sqrt{\frac{1-\epsilon_t}{\epsilon_t}} + (1-\epsilon_t) \cdot \sum_{n=1}^{N} w_n^t / \sqrt{\frac{1-\epsilon_t}{\epsilon_t}}$$

Factor out the common term:

$$\begin{aligned}\sum_{n=1}^{N} w_n^{t+1} &= \sum_{n=1}^{N} w_n^t \left(\epsilon_t \cdot \sqrt{\frac{1-\epsilon_t}{\epsilon_t}} + (1-\epsilon_t) / \sqrt{\frac{1-\epsilon_t}{\epsilon_t}} \right) \\ &= \sum_{n=1}^{N} w_n^t \left(\sqrt{\epsilon_t(1-\epsilon_t)} + \sqrt{\epsilon_t(1-\epsilon_t)} \right) \\ &= \sum_{n=1}^{N} w_n^t \cdot 2\sqrt{\epsilon_t(1-\epsilon_t)}\end{aligned}$$

Next, we calculate the new error rate ϵ_{t+1}:

$$\epsilon_{t+1} = \frac{\sum_{g_t(\mathbf{x}_n)\neq y_n} w_n^{t+1}}{\sum_{n=1}^{N} w_n^{t+1}}$$

Substituting the weights:

$$\begin{aligned}\epsilon_{t+1} &= \frac{\epsilon_t \cdot \sum_{n=1}^{N} w_n^t \cdot \sqrt{\frac{1-\epsilon_t}{\epsilon_t}}}{2\sqrt{\epsilon_t(1-\epsilon_t)} \cdot \sum_{n=1}^{N} w_n^t} \\ &= \frac{\sqrt{\epsilon_t(1-\epsilon_t)}}{2\sqrt{\epsilon_t(1-\epsilon_t)}} = \frac{1}{2}\end{aligned}$$

Thus, we have shown that $d_t = \sqrt{\frac{1-\epsilon_t}{\epsilon_t}}$ can degrade the previous classifier $g_t(x)$ and make its error rate become 0.5 with new weights $\{w_n^{t+1}\}_{n=1}^{N}$.

2 Programming Part

2.1 Compare the performance of the linear, nonlinear, and random forest models on the classification and regression tasks. Based on the evaluation metrics (accuracy for classification and mean squared error for regression), which model performs better for each task, and why do you think this is the case?

All the results are in Table 1, the results of python script and .ipynb are different in MSE. I am really confused about this result since we fix the random seed and I use the same conda environment, hoping that TA could help me figure this out. And since the requirement is to submit the python script, I would focus on the result of python script. We can see that in both tasks, **Random Forest is the best** and followed by Decision Tree and Linear Model is the worst, which is pretty valid and aligning with my knowledge.
For **classification**, I think if there exist some outlier with some very different feature, a simple linear model could not perfectly finding the solution. But the random forest and decision tree are non-linear models who can handle this situation. Not to mention that the decision tree can utilize this phenomenon to split them with its algorithm's characteristic.
As for **regression**, I think it's valid that the non-linear could have a better performance in regression task since that the linear model could only generate the linear solution. But the non-linear models could generate more solutions options to support the tasks that linear model could not fulfill.
But I am so confused that why the results of the python script and jupyter notebook are different. And I think the result of jupyter notebook is more reasonable since the target of regression task is **'medv'** and the scale seems not that big to cause the MSE like 43, 28, 24.

Model	Acc	MSE
Linear Model	0.667	43.414
Decision Tree	0.889	28.342
Random Forest	0.933	24.322

(a) **Results of python script.**

Model	Acc	MSE
Linear Model	0.667	0.070
Decision Tree	0.889	0.111
Random Forest	0.933	0.472

(b) **Results of jupyter notebook.**

Table 1: **Comparison of results.**

2.2 Apply both normalization and standardization techniques to the datasets and compare their impact on the performance of one of the models (e.g., logistic regression). Explain the rationale behind each technique, and discuss their advantages and disadvantages in the context of the given tasks.

Normalization (Min-Max Scaling): This technique scales all the numeric features into the range [0, 1] or [-1, 1] (although [0, 1] is more common). This is done by subtracting the minimum value of each feature and then dividing by the range of that feature.

- **Rationale:** Helps to scale down the effect of large numerical values in features where the algorithm is sensitive to the scale of variables (like logistic regression which uses distance calculations).
- **Advantages:** Ensures that all features contribute equally to the result.
- **Disadvantages:** Can be sensitive to outliers, and the presence of a single outlier in the data can reduce the range of all other normal data points.

Standardization (Z-score Normalization): This technique involves rescaling the features so that they'll have the properties of a standard normal distribution with $\mu = 0$ and $\sigma = 1$, where μ is the mean (average) and σ is the standard deviation from the mean.

- **Rationale:** Standardization does not bound values to a specific range, which may be useful for some algorithms e.g., logistic regression, which do not require input to be bounded but still benefit from standardization.
- **Advantages:** Less affected by outliers and if the data already has a normal distribution, standardization does not skew the data.

- **Disadvantages:** Does not scale the data into a fixed range, which might be necessary for other algorithms.

Though the problem asks us to choose one model to discuss, I found some interesting facts that the trend is quite different between logistic regression and decision tree as well as random forest. And Table 2 shows the results of 3 models in classification task.
As we can see, linear model performs better while using standardization and both non-linear perform better while using normalization(but quite similar to be honest). I think this is an interesting result that I have never expected.
I think it's because logistic regression is quite sensitive about the scale of the given data so that the standardization could enhance the performance of it. However, decision tree and random forest is basically decided on the rule so they are not so sensitive about the scale of the data.

Model	Normalization	Standardization
Linear Model	0.667	0.889
Decision Tree	0.889	0.882
Random Forest	0.933	0.889

Table 2: **Results of both techniques.**

2.3 Train your logistic or linear regression model using two different configurations of learning rate and number of iterations. Compare the performance of these configurations using the evaluation metrics (accuracy for classification and mean squared error for regression). Discuss how the choice of learning rate and the number of iterations affect the convergence and overall performance of the models.

Table 3 shows the results of different configuration. We can see that no matter learning rate or iterations, 0.01 and 1000 iterations is not the point it converge. So both method could make the performance better. I don't think there is any way to ensure the improvement of model performance. We just need to trial and error, for me myself, I would set the learning rate bigger and iteration larger as well if the performance seems could be better. Like I did in the Table 3. But if the learning rate is too large, that might be a too big jump for the model to getting the best point.

Configuration	Accuracy
lr=0.01, iterations=1000	0.667
lr=0.03, iterations=1000	0.800
lr=0.01, iterations=20000	0.933
lr=0.03, iterations=20000	0.978

Table 3: **Results of both techniques.**

2.4 Discuss the impact of hyperparameters, such as the number of trees and maximum depth, on the performance of the random forest model for both classification and regression tasks. How do these hyperparameters affect the model's complexity, generalization, and potential for overfitting? Include a brief explanation of how you selected or tuned these hyperparameters in your implementation.

For random forest's number of trees (number of estimators)

- **Effect on Performance:**
  - **More Trees:** Increasing the number of trees generally improves the model's performance by reducing variance. More trees lead to better averaging of the predictions, which can enhance the stability and accuracy of the model.
  - **Fewer Trees:** With too few trees, the model might not capture the underlying patterns adequately, resulting in higher variance and potentially poor performance.

- **Effect on Complexity and Overfitting:**
  - More trees do not typically cause overfitting, as the averaging process helps to mitigate this risk. However, it increases computational cost and memory usage.
  - A sufficient number of trees ensures that the model generalizes well to unseen data.
- **Selecting and Tuning Hyperparameters:** Formally, to select and tune the hyperparameters for a random forest model, follow these steps. But I actually did not do these implementation in this HW since it's not needed (I only tuned the number of trees to 200 to check the performance):
  - **Grid Search:** Use grid search to systematically explore combinations of hyperparameters. Define a range of values for number of estimators and maximum depth and evaluate the model performance for each combination. (I did this in my undergraduate research in WMNLab.)
  - **Cross-Validation:** Incorporate cross-validation during grid search to ensure that the hyperparameter selection is robust and generalizes well to unseen data. This process helps in selecting the best hyperparameters based on the average performance across multiple folds of the data.

2.5 Analyze the strengths and weaknesses of the implemented models. In what scenarios would you choose to use a linear model over a nonlinear model, or a random forest model, and vice versa? Discuss the trade-offs between model complexity, interpretability, and performance.

- **Linear Model**
  - **Strengths:** Linear models are straightforward to implement, train, and predict with, making them very fast compared to more complex models. And they require fewer parameters, which generally makes them less prone to overfitting, especially with fewer data points.
  - **Weaknesses:** Linear models assume a linear relationship between features and the target variable, which can limit their accuracy and applicability in cases where this assumption does not hold. And they often perform poorly on datasets where feature interactions are important or where the relationship between features and target is non-linear.
  - **Scenarios:**
    - The relationship between the features and target is approximately linear.
    - Need a quick, simple solution that is easy to interpret.
    - Have a very large number of features, and data is sparse (linear models can be more effective in these scenarios, especially with regularization).
- **Decision Tree**
  - **Strengths:** Can model non-linear relationships between features and the target, which can provide more accurate predictions for complex problems. And automatically captures interactions between features without needing explicit feature engineering. Not to mention it's highly interpretable with its expert-like characteristic.
  - **Weaknesses:** More complex models are more prone to overfitting, especially if not properly regularized or if the data set is not large enough.
  - **Scenarios:**
    - Choose a nonlinear model when the dataset exhibits complex, nonlinear relationships that a linear model cannot capture
    - Such as in image recognition, speech recognition, or complex pattern recognition in time series data.
- **Random Forest**
  - **Strengths:** Generally provides very high accuracy and performance across a wide range of classification and regression tasks. And due to the averaging of multiple decision trees, it is less likely to overfit than individual decision trees.

  - **Weaknesses:** More computationally intensive than linear models, requiring more memory and processing power. And the size of the model (number of trees) can become quite large, and managing such models in production can be challenging.
  - **Scenarios:**
    * Use a random forest model when you deal with heterogeneous features and complex datasets where performance outweighs the need for model simplicity
    * Such as in bio-information for gene classification or complex regression tasks in ecological data.

- **Trade-offs**
  - **Complexity vs. Interpretability**: More complex models generally offer higher performance but at the cost of interpretability. Linear models remain among the most interpretable, while models like random forests and neural networks sacrifice some interpretability for better handling of complex data patterns.
  - **Complexity vs. Performance**: While complex models can capture intricate patterns in the data, they require more data to train effectively without overfitting and are computationally more expensive. This can be a significant trade-off when resources are limited or when decisions need to be made in real-time.
  - **Performance vs. Simplicity**: High performance is critical in applications where predictions must be accurate and reliable, such as in medical diagnosis or financial forecasting. However, simpler models can suffice and even excel in scenarios where the underlying data relationships are well understood and not overly complex.

References

OpenAI and the slides of the lecture are all I need
https://chatgpt.com/c/c3b1be0d-abb3-41b6-8139-2b2bf11303d9