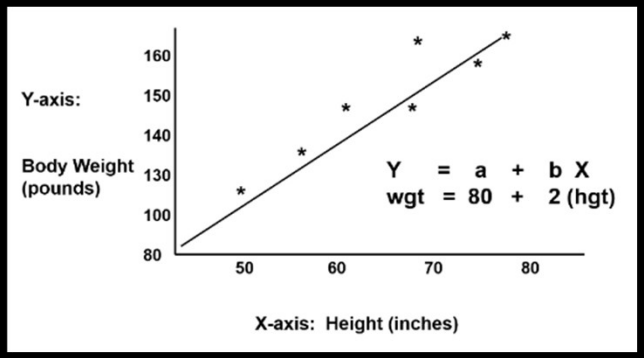
Generative AI Academy

Module 2: Machine Learning

End to end – predict weight of a person based on height

1. Start with training dataset

2. Determine the model to use
   Linear regression in this case



y = mx + c

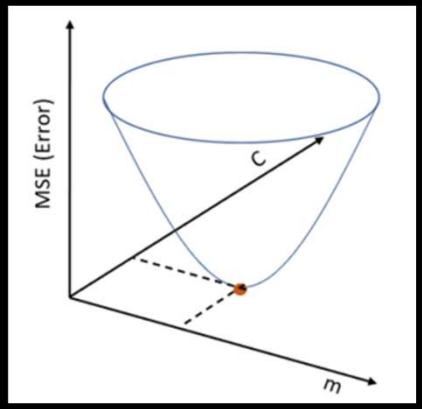
3. Measure error / accuracy

$$Cost\ Function(MSE) = \frac{1}{n}\sum_{i=0}^{n}(y_i - (mx_i + c))^2$$

Cost function (or loss function) measures the error.
Our goal is to minimize the cost function to find the best fit line

4. Tune the model - Minimise error (improve accuracy)

Gradient Descent is a mathematical function that can help compute the model parameters (m, c) that will minimize the loss function in minimum number of iterations.



Relies on derivatives and calculus

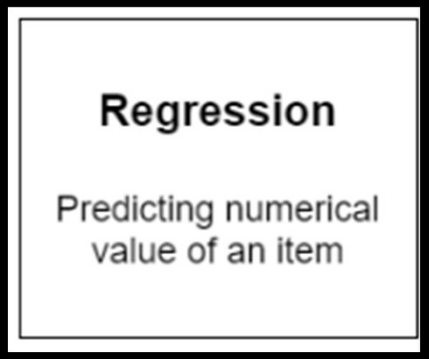
$$D_m = \frac{\partial(Cost\ Function)}{\partial m} = \frac{\partial}{\partial m}\left(\frac{1}{n}\sum_{i=0}^{n}(y_i - y_{i\,pred})^2\right)$$

$$D_c = \frac{\partial(Cost\ Function)}{\partial c} = \frac{\partial}{\partial c}\left(\frac{1}{n}\sum_{i=0}^{n}(y_i - y_{i\,pred})^2\right)$$

$$m = m - LD_m$$

$$c = c - LDc$$

Types of ML problems



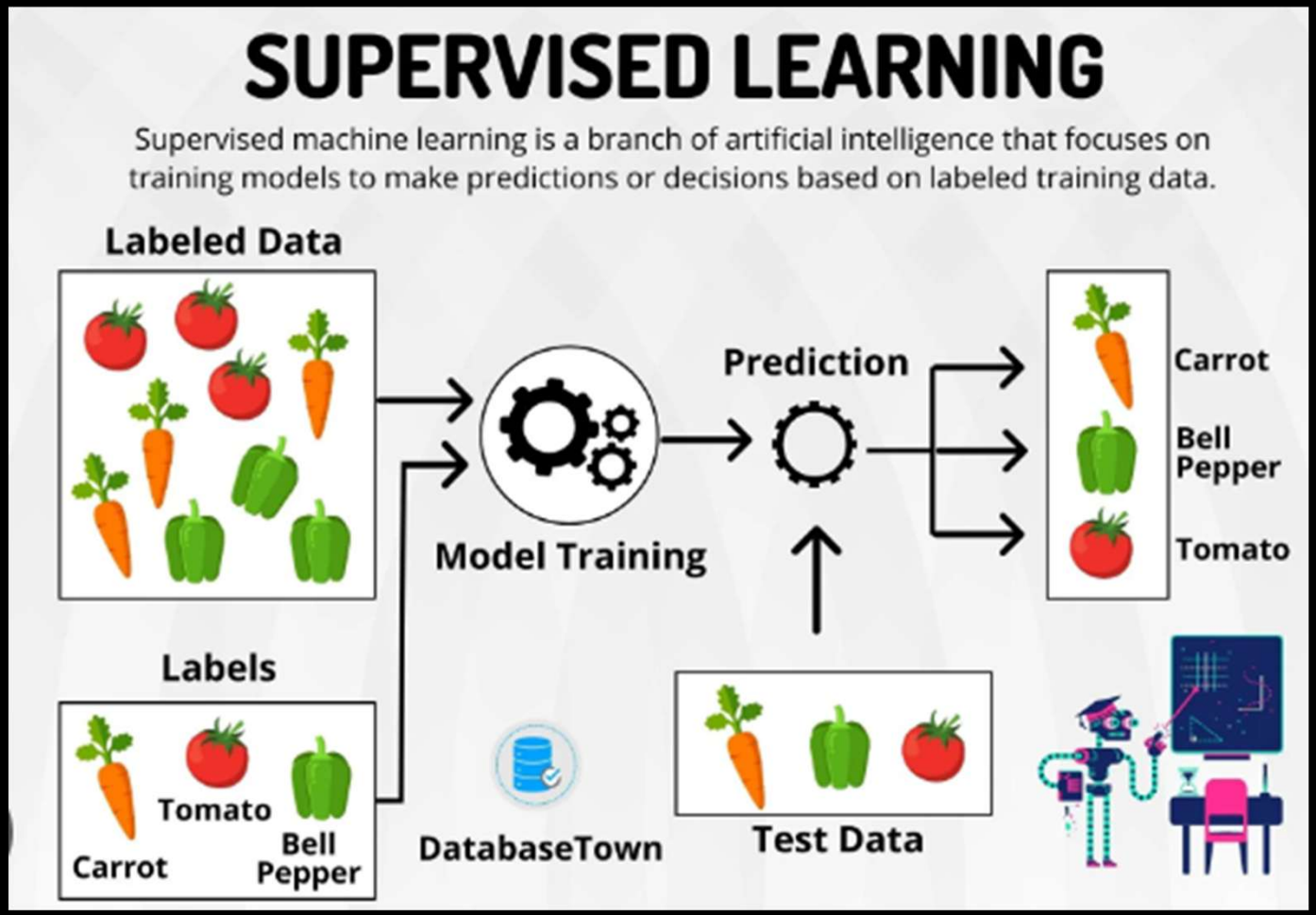
- Is it a cat?
- Is this a fraud transaction?
- Will the customer default on the loan?

- Binary Classification
- Multiclass Classification

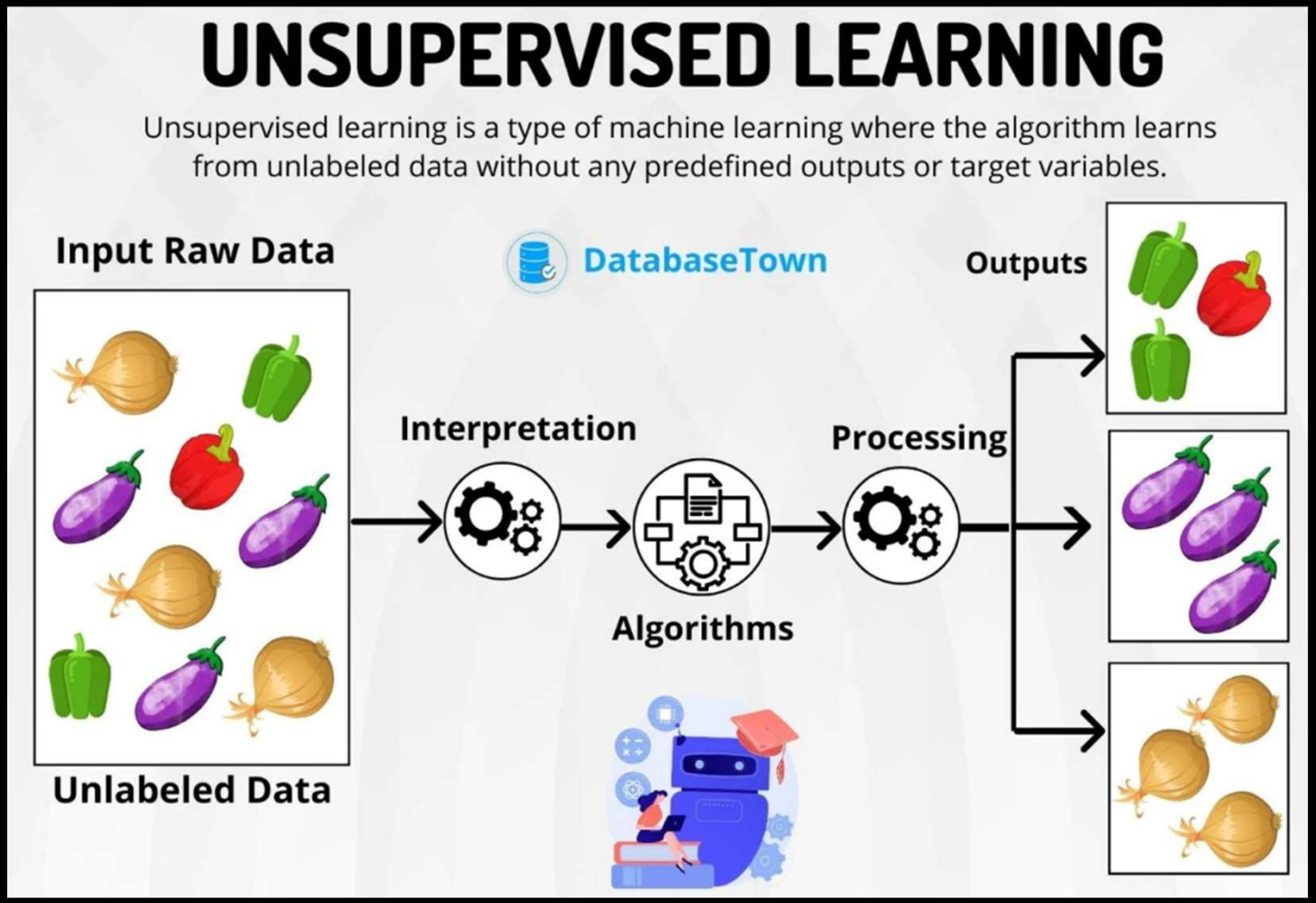
- home price prediction
- stock price prediction
- weight prediction based on height

Regression is a method for understanding the relationship between independent variables or features and a dependent variable or outcome. Outcomes can then be predicted once the relationship between independent and dependent variables has been estimated.

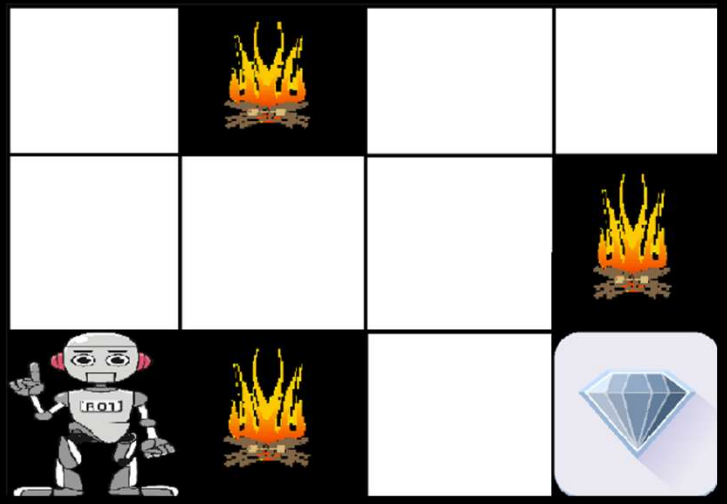
How do we train the model? – supervised learning

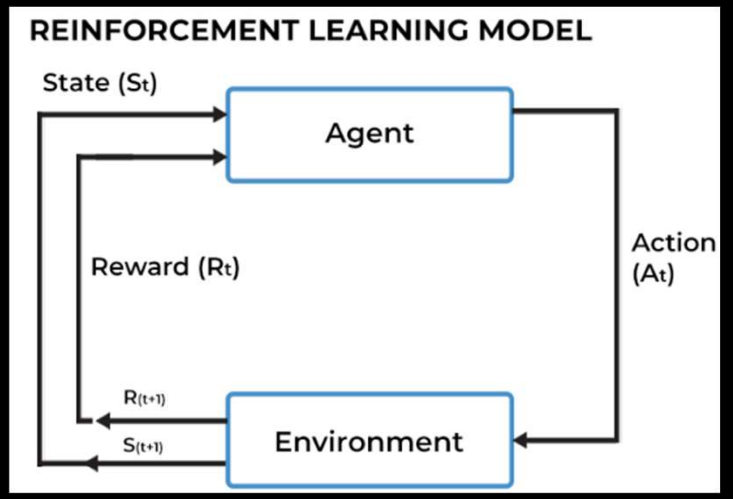


How do we train the model? – unsupervised learning



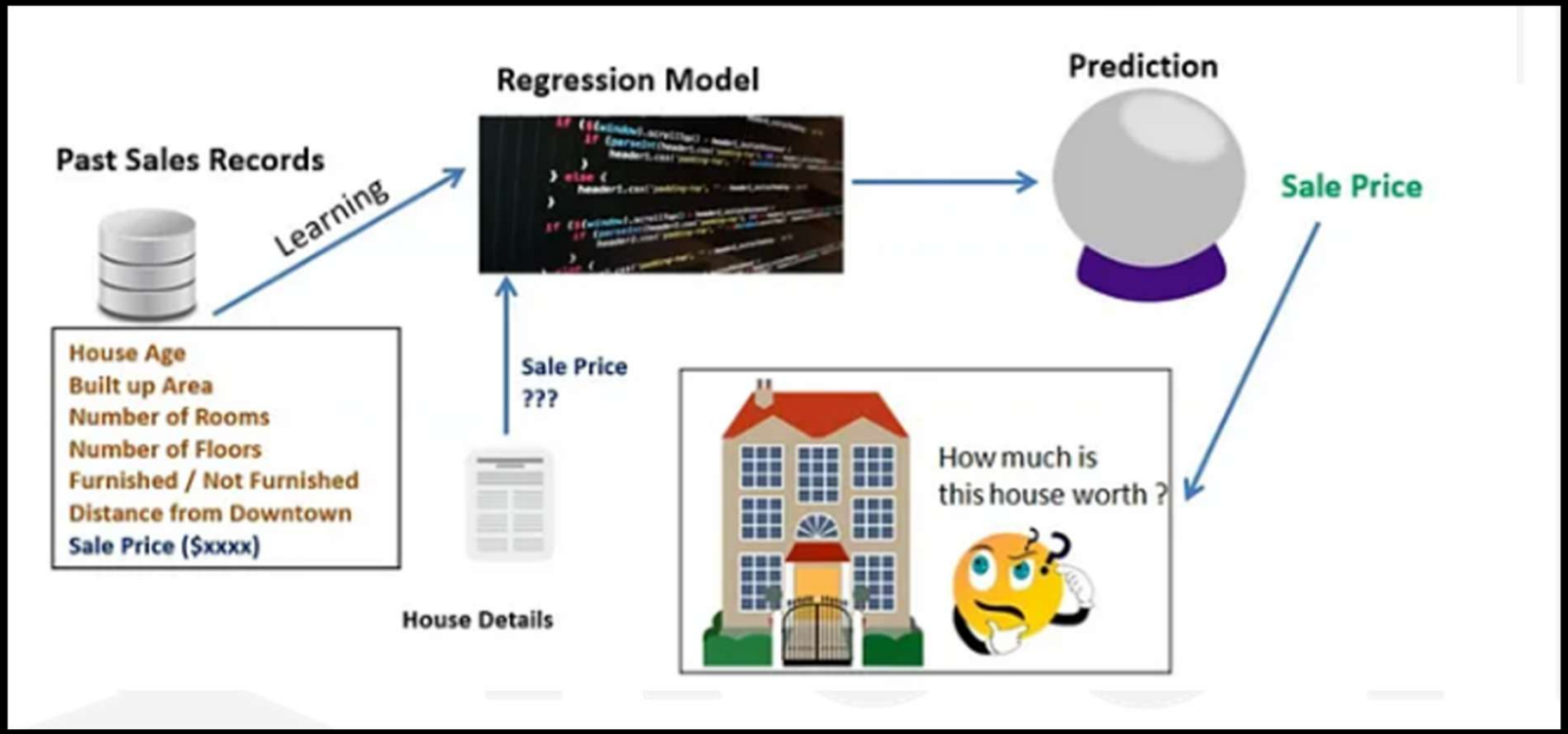
Reinforcement learning



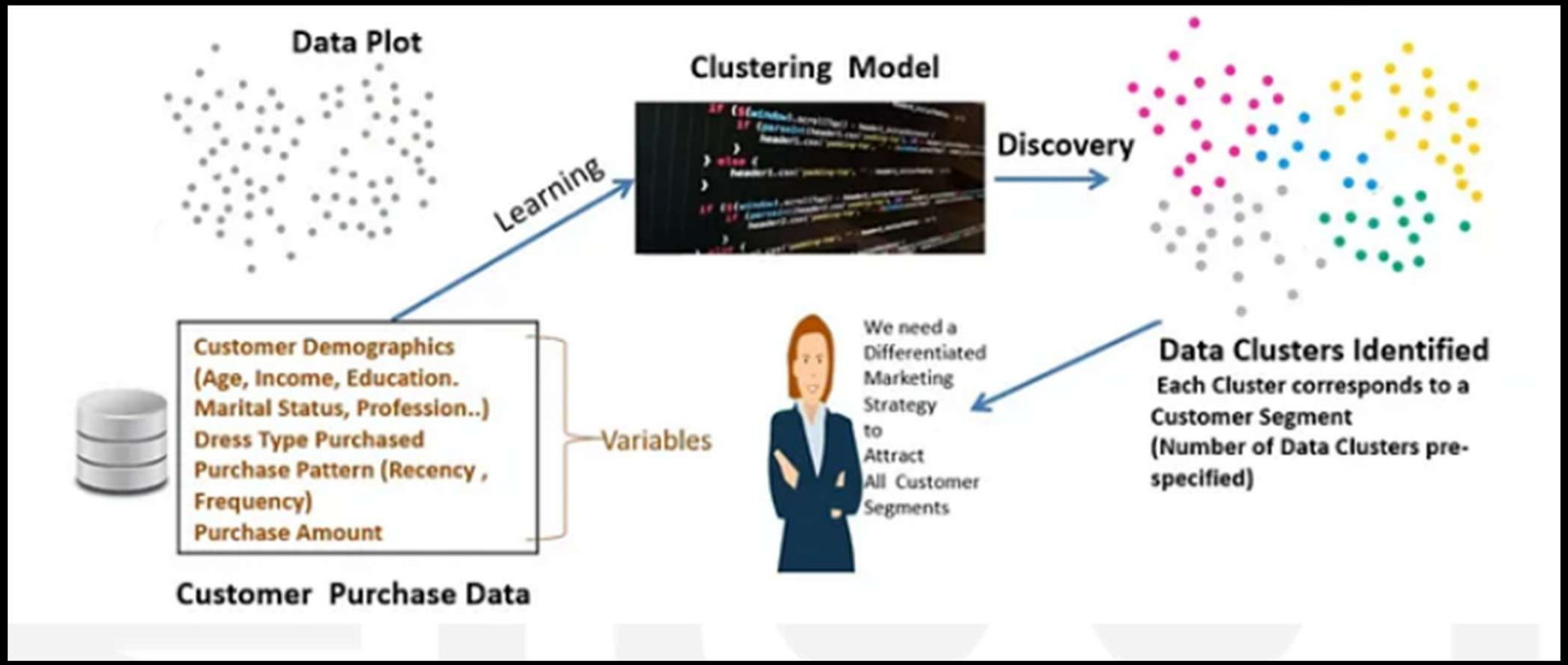


- The goal of the robot is to reach the diamond without hitting the fire squares.
- For each correct move, it is rewarded
- There is no input data
- State, Action, Reward are the key steps
- Robot is the Agent in our example
- Game Board is the Environment

Regression example: property price prediction



Clustering example: customer segmentation



Data science problem types summary

Problem	Description	Examples
Classification	Predicting whether a data point belongs to one of the predefined classes	Customer Churn Prediction, Spam Detection
Regression	Predicting the numerical value of the target variable	Agricultural Yield, Inflation Rate Prediction
Clustering	Identifying natural groups within the dataset based on similar inherent property of the data points	Social Network Analysis, Crime Incidence Analysis, Search Result Grouping
Anomaly Detection	Predicting whether a data point is an outlier in comparison with other points in the data set	Detecting Network Intrusions, Predicting Machine Failures, Detecting Gene Mutations
Association	Discovering rules that govern frequent simultaneous occurrence of certain items or phenomena	Medical Diagnosis, Protein Sequencing, Building Intelligent Transportation Systems
Recommendation	Suggesting items for users based on the past preferences of theirs and similar users	Recommendation of Movies, Books, Restaurants, Holiday Destinations

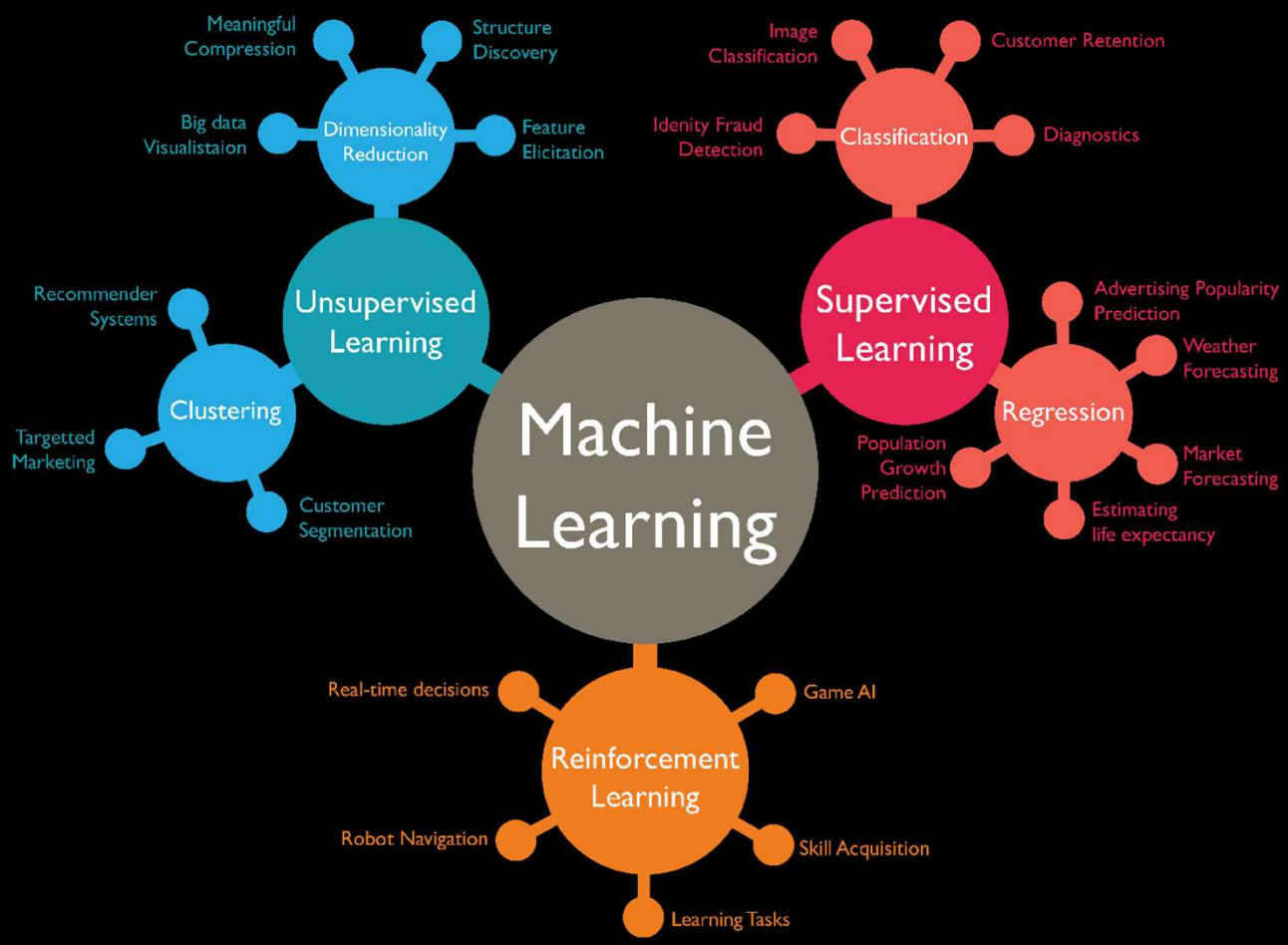
ML Problem types and training methods

Problem Type	Training Method	Common Algorithms
Classification	Supervised	Logistic Regression, Decision Trees, Random Forest, Support Vector Machines (SVM), K-Nearest Neighbors (KNN), Neural Networks
Regression	Supervised	Linear Regression, Ridge/Lasso Regression, Polynomial Regression, Random Forest, Gradient Boosting Machines (GBMs), Neural Networks
Clustering	Unsupervised	K-Means, Hierarchical Clustering, DBSCAN, Gaussian Mixture Models (GMMs), Spectral Clustering
Anomaly Detection	Semi-supervised, Unsupervised, or Supervised	Isolation Forest, One-Class SVM, Autoencoders, DBSCAN, Gaussian Mixture Models, LOF (Local Outlier Factor)
Association	Unsupervised	Apriori, Eclat, FP-Growth (Frequent Pattern Growth)
Recommendation Engine	Supervised, Unsupervised, or Reinforcement Learning	Collaborative Filtering (Matrix Factorization, Singular Value Decomposition), Content-Based Filtering, Neural Collaborative Filtering, Reinforcement Learning (Bandit algorithms)

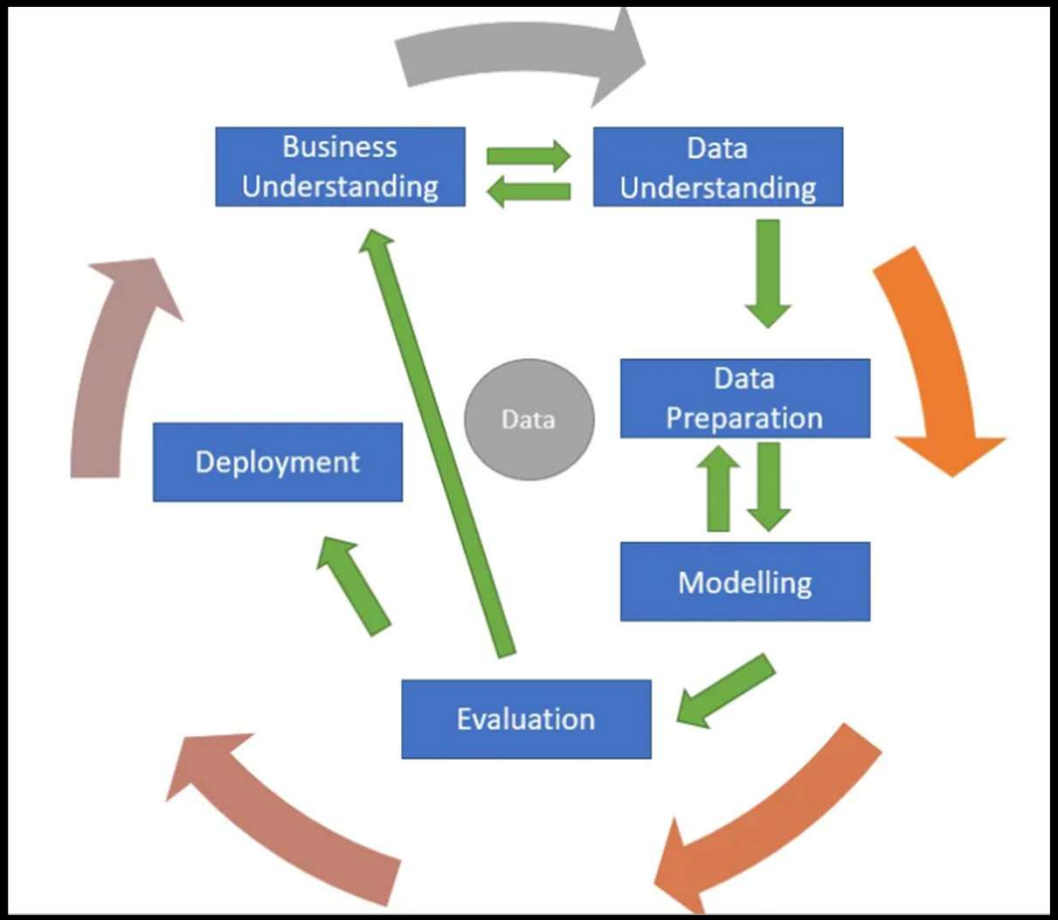
ML Problem types and training methods

Problem Type	Training the model	Model examples
Classification	Supervised	Logistic regression, K-nn, Decision trees
	Unsupervised (Not commonly used)	K-means clustering
Regression	Supervised	Linear regression
	Unsupervised (Not commonly used)	Neural Networks
Clustering	Unsupervised	Neural Networks
	Semi-supervised	Uses a small amount of labeled data to guide the clustering process

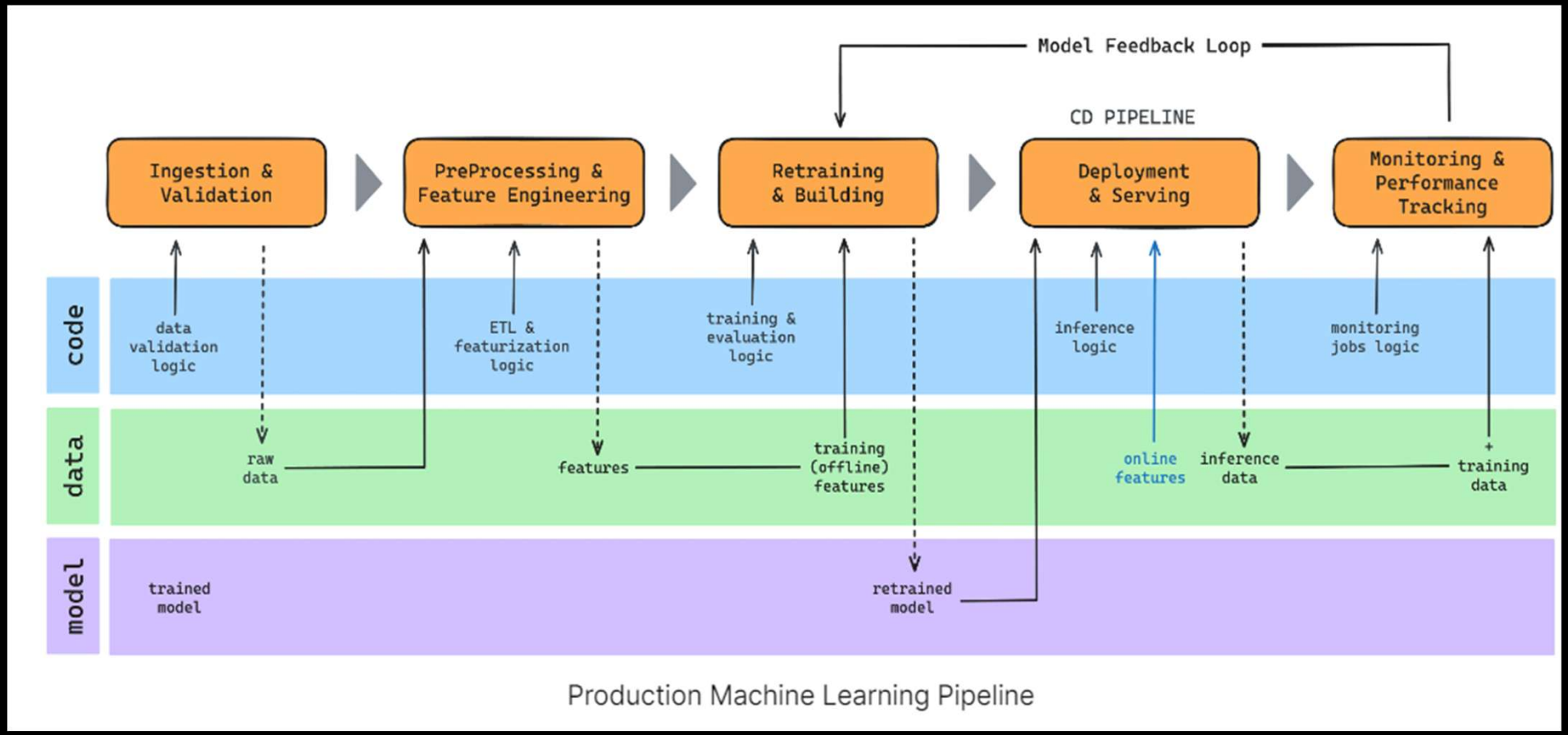
Machine learning map



Machine learning problem solving steps

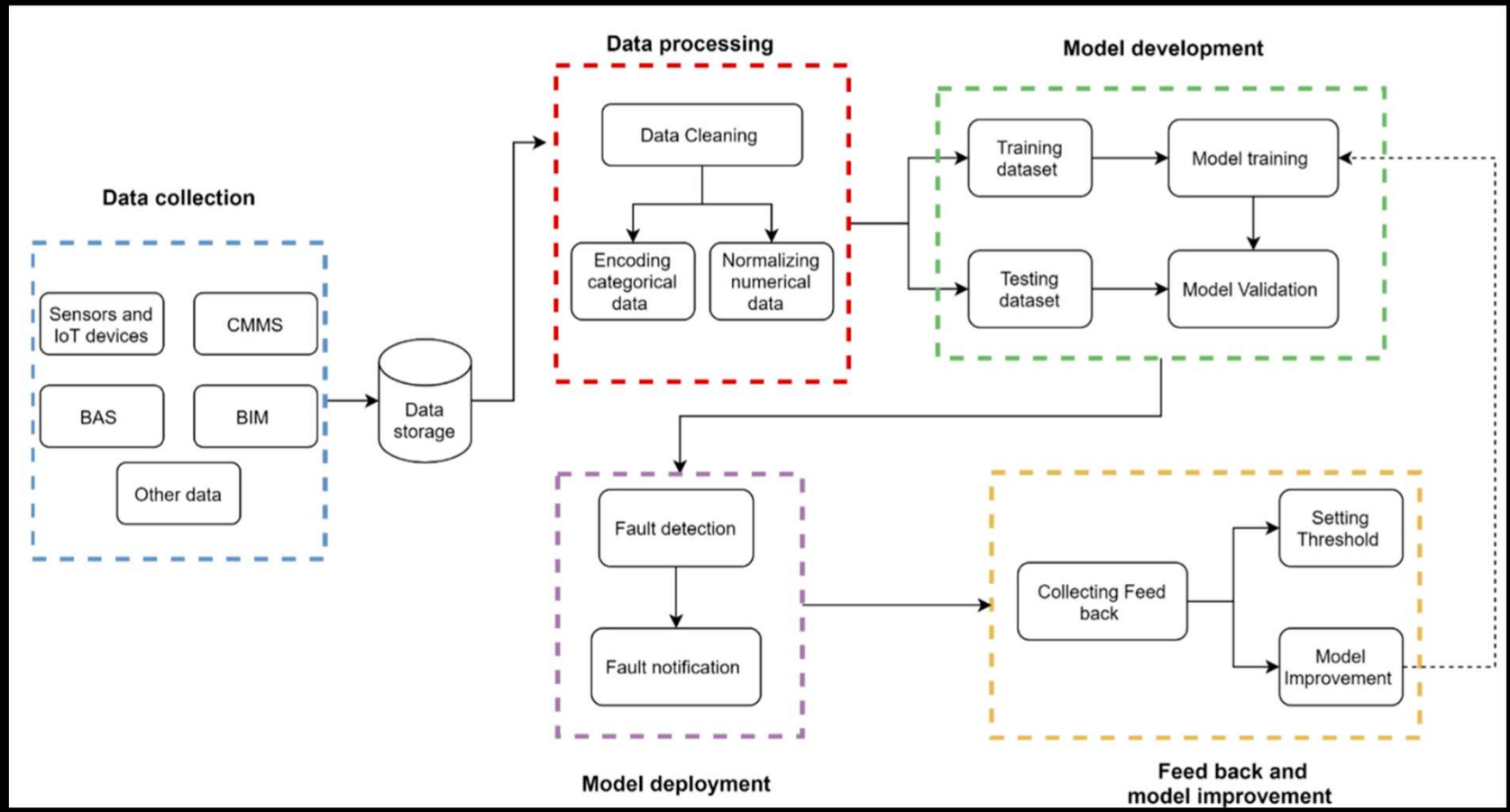


Machine learning pipeline



Production Machine Learning Pipeline

Detailed flow and steps to execute a data science project



Modeling – Linear Regression

How it works:
Linear regression models the relationship between a dependent variable y and one or more independent variables X by fitting a linear equation:

$y = \beta_0 + \beta_1 X_1 + \beta_2 X_2 + \ldots + \beta_n X_n + \varepsilon$
where β_0 is the intercept, $\beta_1 \ldots \beta_n$ are coefficients, and ε is the error term.

The model estimates these coefficients to minimize the sum of squared residuals (differences between predicted and actual values).

Evaluation Metrics:
- Mean Squared Error (MSE)
- Root Mean Squared Error (RMSE)
- R-squared (coefficient of determination)
- Adjusted R-squared

Modeling – Linear Regression

Advantages

- Simplicity and Interpretability
- Computational Efficiency
- Scales well to large datasets with efficient implementations
- Works Well for Linearly Separable Data
- Performs optimally when the relationship between variables is indeed linear
- Basis for Many Other Methods
- Well-studied Statistical Properties
- Well-established methods

Limitations

- Assumes Linearity
- Assumes Features are Independent (Multicollinearity Issues, Performance degrades when independent variables are highly correlated)
- Cannot capture intricate patterns or interactions without manual feature engineering
- Assumes Independence of Errors (Assumes that the residuals are uncorrelated, which is often violated in time series data)
- Assumes Homoscedasticity (constant variance of errors, which may not hold in many real-world scenarios)
- Not Suitable for Classification Tasks

Evaluating Linear Regression

OLS Regression Results

Dep. Variable:	y	R-squared:	0.963
Model:	OLS	Adj. R-squared:	0.961
Method:	Least Squares	F-statistic:	618.2
Date:	Tue, 11 Feb 2025	Prob (F-statistic):	4.55e-67
Time:	07:36:54	Log-Likelihood:	-200.55
No. Observations:	100	AIC:	411.1
Df Residuals:	95	BIC:	424.1
Df Model:	4		
Covariance Type:	nonrobust		

Evaluating Linear Regression

Metric	Description
Dep. Variable: `y`	The dependent (response) variable being predicted.
Model: `WLS`	Indicates that this is a Weighted Least Squares regression model.
Method: `Least Squares`	The estimation method used for regression (minimizing the sum of squared errors).
No. Observations: `200`	Number of observations (data points) used in the regression.
Df Residuals: `198`	Degrees of freedom for residuals, calculated as: **Total Observations - Number of Parameters** = `200 - 2 = 198`.
Df Model: `1`	The number of predictor variables (excluding the intercept), here it is `1` because we have only `x1`.

Goodness of fit statistic

Metric	Description
R-squared: `0.899`	The coefficient of determination, which measures how much of the variability in `y` is explained by `x1`. A value of `0.899` means **89.9% of the variance in `y` is explained** by the model.
Adj. R-squared: `0.899`	Adjusted R^2 accounts for the number of predictors and is useful when comparing models with different numbers of predictors. Since we have only one predictor, R^2 and Adj. R^2 are almost the same.
F-statistic: `1771`	Tests whether the independent variable(s) explain a significant amount of variance in `y`. A large value suggests a strong relationship.
Prob (F-statistic): `1.05e-100`	The **p-value** for the F-statistic. A very small value (`< 0.05`) suggests that at least one predictor significantly contributes to explaining `y`. Here, it's effectively zero, meaning the model is highly significant.

Residual diagnostic

Metric	Description
Omnibus: `7.476`	Tests whether the residuals are normally distributed. Higher values suggest deviation from normality.
Prob(Omnibus): `0.024`	The p-value for the Omnibus test. If < 0.05, it suggests **non-normality** in residuals. Here, `0.024` suggests some deviation from normality.
Jarque-Bera (JB): `9.907`	Another test for normality. Similar to Omnibus but based on skewness and kurtosis. Higher values indicate non-normality.
Prob(JB): `0.00706`	The p-value for the Jarque-Bera test. A small value (< 0.05) suggests non-normality in residuals.
Skew: `0.246`	Measures asymmetry in residuals. A value close to `0` suggests symmetry.
Kurtosis: `3.973`	Measures whether the residuals have heavy or light tails compared to a normal distribution. A value near `3` is normal; here it is slightly higher.
Durbin-Watson: `2.124`	Tests for **autocorrelation** in residuals (important for time series data). A value close to `2` suggests no significant autocorrelation.
Cond. No.: `2.06`	Measures **multicollinearity**. A low value (< 30) is good; **very high values** (> **1000**) suggest strong multicollinearity. Here, `2.06` is very low, indicating no collinearity issues.

Understanding R-squared Vs F-statistic

Understanding R-Squared vs. F-Statistic

1. R^2 (Goodness of Fit)
   - Measures **how well the model fits the data**.
   - High R^2 means **the model explains most of the variance**.
   - But **does not tell if the predictors are statistically significant**.
2. **F-Statistic (Significance of Predictors)**
   - Tests whether **the independent variables as a group** explain a **significant** amount of variation in y.
   - A **high F-statistic and low p-value** (< 0.05) suggest the model **as a whole** is statistically significant.

Heteroscedasticity

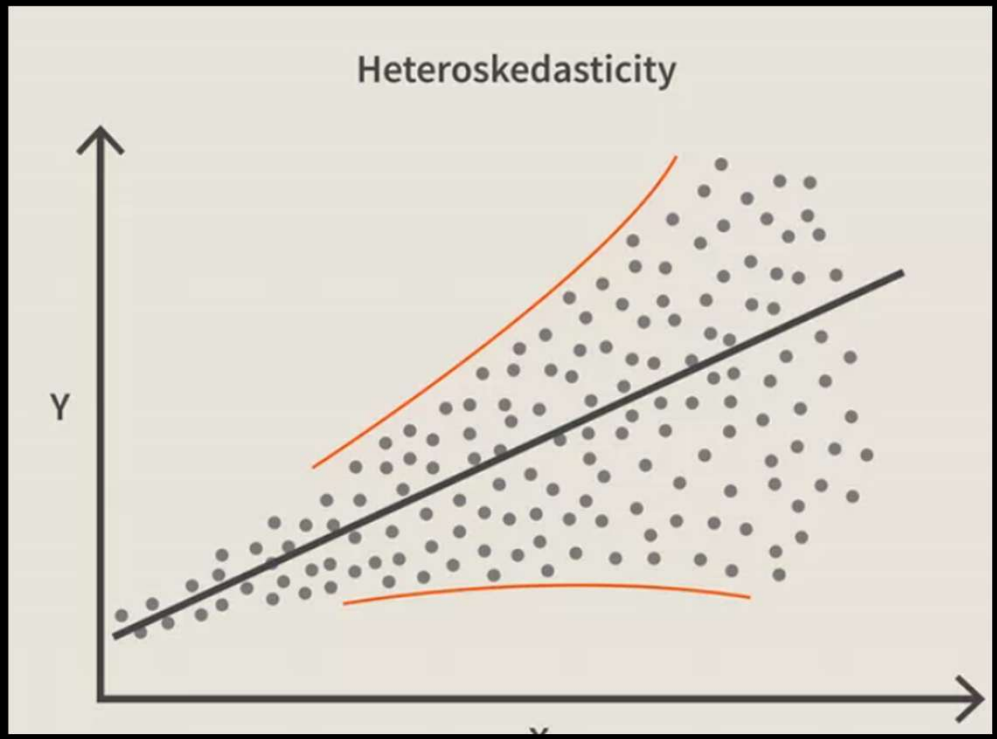
Heteroscedasticity means the variance of the error term is not constant across observations.

Classic OLS assumes errors are identically and independently distributed with constant variance.

Ways to address Heteroscedasticity:

- Weighted Least Squares (WLS):
  Assign weights inversely proportional to the error variance.

- Variance-Stabilizing Transformations
  Transform either the response or predictors to stabilize variance, e.g., using a log transform if variance grows with the mean.



More ways to address heteroscedasticity

- **Transform the target variable**
  Such as Log transform If the error grows multiplicative (not additive)

- **Two stage modeling**
  predicting the residuals separately
  Final Prediction = Base Model + Residual Model

- **Heteroscedasticity aware models:**
  Generalized Least Squares (GLS), which can model variance explicitly.
  Neural Networks or Gradient Boosting Models, which can learn variance patterns

- **Polynomial regression**
  Instead of forcing a straight line, fit a quadratic or cubic function.

Modeling – Ridge Regression

Addresses muti-collinearity

Ridge Regression adds a ℓ1 penalty term ($\lambda||\beta||^2$) to the objective function.
This term discourages large coefficients by adding their squared values to what we're minimizing.
The full objective becomes:

$$\min ||y - X\beta||^2 + \lambda||\beta||^2$$

The λ (lambda) parameter controls how much we penalize large coefficients.
When λ = 0, we get OLS; as λ increases, the coefficients shrink toward zero (but never quite reach it).

Modeling – Lasso Regression

Addresses muti-collinearity

Lasso stands for Least Absolute Shrinkage and Selection Operator.
It is a type of linear regression that includes an $\ell 1$ penalty on the magnitude of the regression coefficients.

Where doe Lasso help?

- In scenarios where the number of predictors p is large
- Because Lasso can shrink some coefficients to exactly zero, the resulting model is more parsimonious.
- Users can focus on a smaller set of meaningful predictors without manually searching across all possible subsets.
- Multi-collinearity – Lasso tends to select a few and make other feature coefficients ZERO!

Lasso

Lasso shines when:

- Many Predictors, Few Observations (e.g., $p \gg n$)
  - Genomics (many gene expression features, relatively few samples).
  - Text analysis (thousands of word features, moderate sample size).
- You Suspect Only a Few Predictors Truly Matter
- A Need for Interpretability
- When You Want to Automate Variable Selection

Ridge Vs Lasso

Use **Ridge** If:

1. You suspect *all* features might have some influence, even if small.
2. You have high collinearity among predictors and want to **shrink** them without discarding any.
3. Your primary goal is to reduce **variance** (and potential overfitting) rather than to identify a strict subset of predictors.

Use **Lasso** If:

1. You expect **only a few predictors** truly matter and want to perform **feature selection** automatically.
2. You want a **sparse** model for interpretability or computational reasons.
3. You are in high-dimensional settings ($p \gg n$) and need to identify a small subset of important variables.

Modeling – Logistic Regression

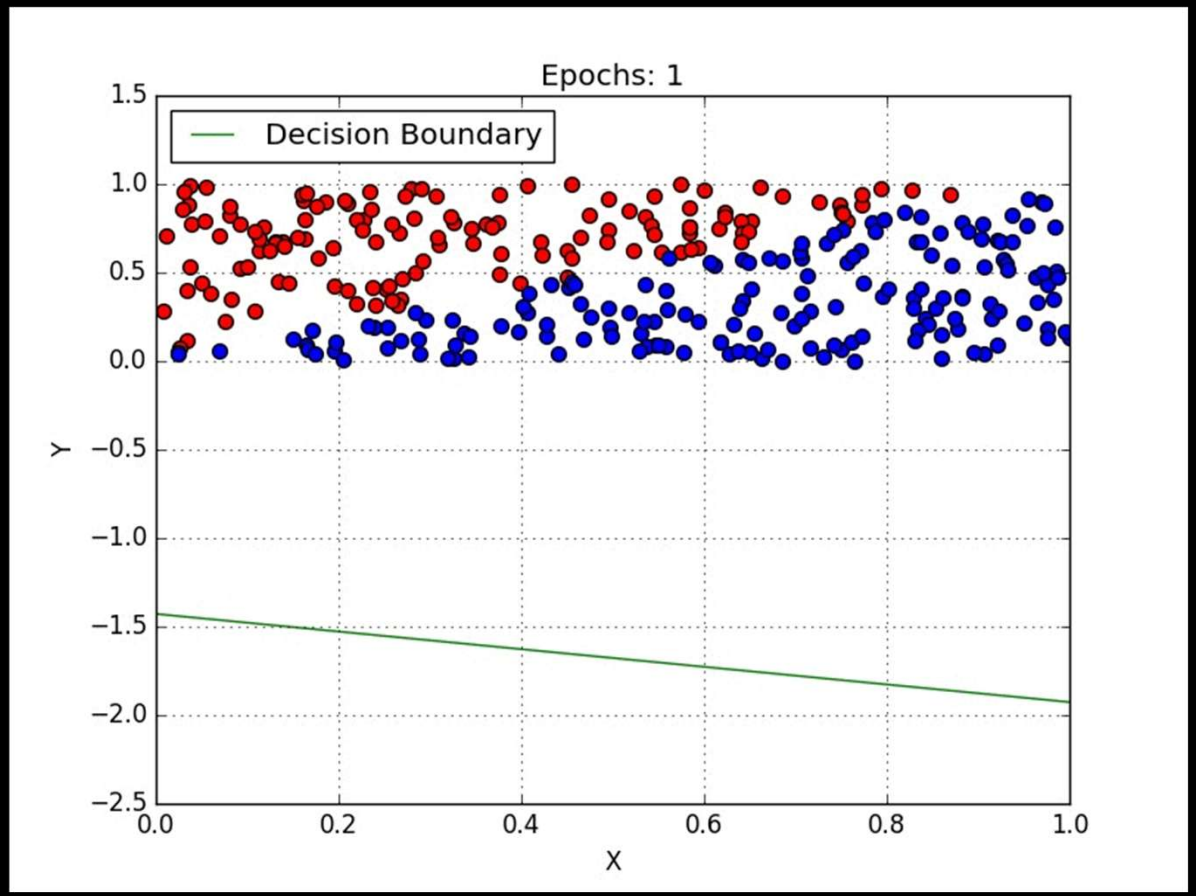
Logistic Regression is a supervised learning algorithm used for classification problems (binary or multi-class).

It models the **probability** that an input belongs to a particular class using the sigmoid (logistic) function.

- Applies decision boundary (e.g., probability > 0.5 → Class 1, otherwise Class 0)
- Uses Cross-Entropy Loss for optimization
- Trained using Gradient Descent
- Extended to Multi-Class using Softmax (Multinomial Logistic Regression)

Modeling – Logistic Regression

Illustration with loan approval classification

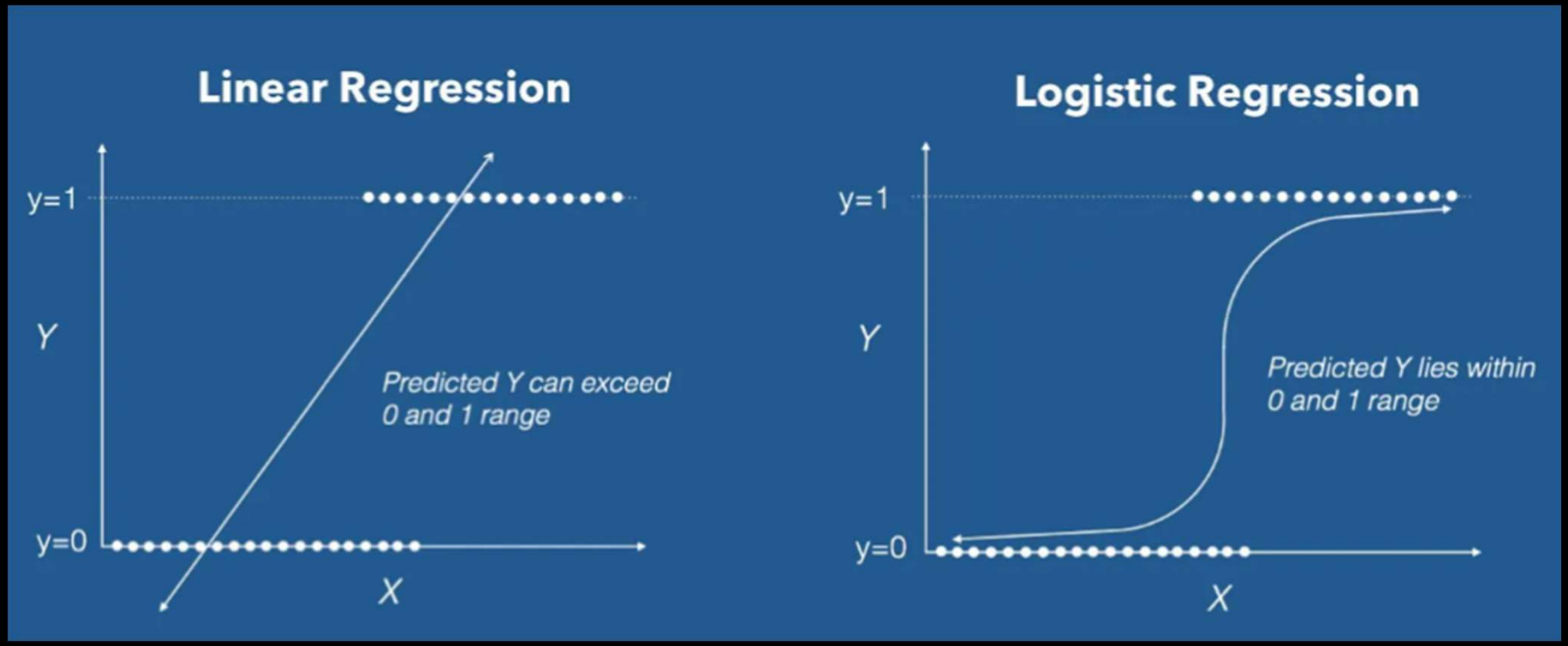


X-axis: income

Y-axis: existing loan amount

Red: loan rejected
Blue:loan approved

Modeling – Logistic Regression



Logistic Regression uses a more complex cost function, this cost function can be defined as the '**Sigmoid function**' or also known as the 'logistic function' instead of a linear function.
https://towardsdatascience.com/introduction-to-logistic-regression-66248243c148

Modeling – Logistic Regression

How it works:
Logistic regression uses the logistic function to model the **probability** of a binary outcome:
$P(Y=1|X) = 1 / (1 + e^{(-z)})$
where $z = \beta_0 + \beta_1 X_1 + \beta_2 X_2 + ... + \beta_n X_n$
The model estimates the coefficients using maximum likelihood estimation.

Evaluation Metrics:
- Accuracy
- Precision, Recall, F1-score
- ROC AUC
- Log loss

Hyperparameters:
- Regularization strength (C or lambda)
- Solver algorithm (e.g., 'liblinear', 'newton-cg', 'lbfgs')
- Max iterations

Model evaluation: is accuracy sufficient?

Let us take example of a classification model that classifies if a transaction is fraudulent or not
The training data is labeled as:
“yes” means the transaction is fraudulent (True positive)
“no” means transaction is not fraudulent (True negative)

Actual Values

Output	Count
Yes (Fraudulent)	30
No (Genuine)	970

Predicted Values

Output	Count
Yes (Fraudulent)	1
No (Genuine)	999

Accuracy

$$\frac{True\ Positives\ +\ True\ Negatives}{Total\ Predictions}$$

$$\frac{971(?)}{1000} = 97.1\ \%$$

The model has a best-case **accuracy** of **97.1%**, even when it didn’t detect 29 out of 30 fraudulent transactions

Hence other evaluation techniques such as confusion matrix are used
False negatives and False positives should be close to 0

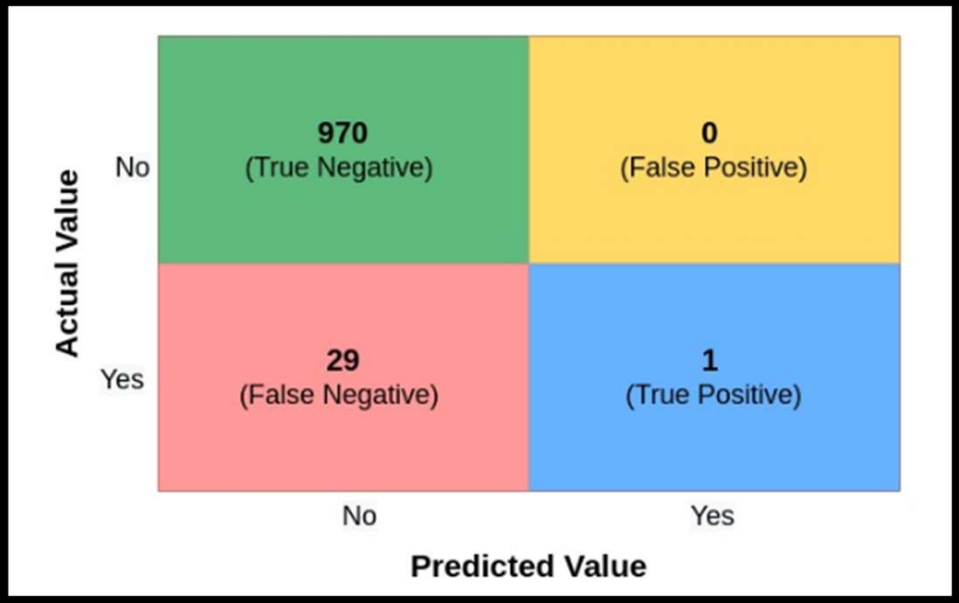
Confusion Matrix

True positive: transaction is fraudulent and predicted as fraudulent

True negative: transaction is not fraudulent and predicted as not fraudulent

False positive: transaction is not fraudulent and predicted as fraudulent

False negative: transaction is fraudulent and predicted as not fraudulent



Confusion Matrix

Precision

Measure of how many **predicted positives** it got right
(predicted as fraudulent and they were actually fraudulent)

$$\text{Precision} = \frac{\text{True Positives}}{\text{True Positives} + \text{False Positives}} = \frac{1}{1 + 0} = 1\ (100\%)$$

Recall

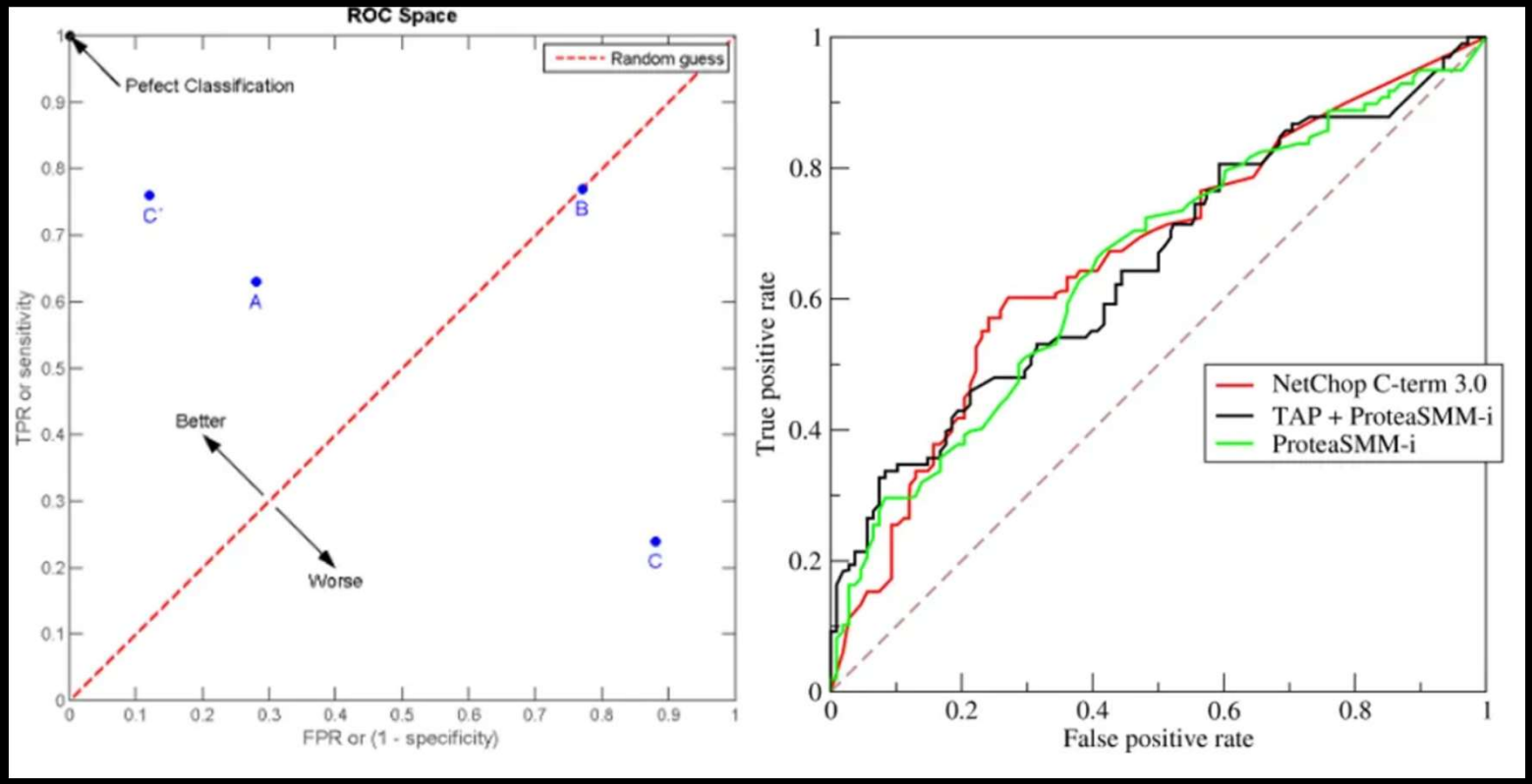
Measure of **all** positives correctly predicted or not?
Were **all** fraudulent transactions predicted?

$$\text{Precision} = \frac{\text{True Positives}}{\text{True Positives} + \text{False Positives}} = \frac{1}{30 + 0} = 0.03\ (3\%)$$

F1 score

Harmonic mean of precision and recall, balancing both for a single score.

ROC and AUC



ROC: Receiver Operator Characteristic

AUC: Area under the curve

Logistic Regression Demo and Assignment

Logistic Regression – how does it work?

Step 1: linear predictor (Logit)

- Logistic Regression starts by computing a **linear combination** of input features X.

$$z = \beta_0 + \beta_1 x_1 + \beta_2 x_2 + \ldots + \beta_n x_n$$

where:

- z is the **logit** (linear predictor),
- β_0 is the **intercept (bias term)**,
- $\beta_1, \beta_2, \ldots, \beta_n$ are the **weights (coefficients)**,
- $x_1, x_2, \ldots, x_n$ are the **feature values**.

Logit: Linear predictor

Step 2: Sigmoid function – class probability

- Since we need **probabilities** (between 0 and 1), we pass z through the **sigmoid function**:

$$p = \frac{1}{1 + e^{-z}}$$

- The sigmoid function transforms any real number into a probability:
  - If $z \to +\infty$, then $p \approx 1$.
  - If $z \to -\infty$, then $p \approx 0$.
- The predicted probability p represents the likelihood that the given instance belongs to class **1**.

Step 3: Log Loss, Cross-Entropy; measure the error

- Logistic Regression **does not use Mean Squared Error (MSE)** because it leads to non-convex optimization.
- Instead, it uses the **Log-Loss (Binary Cross-Entropy Loss)** function:

$$L = -\frac{1}{m} \sum_{i=1}^{m} [y_i \log(p_i) + (1 - y_i) \log(1 - p_i)]$$

where:

- m is the total number of training samples,
- y_i is the actual class (0 or 1),
- p_i is the predicted probability.
- This function **penalizes wrong predictions** more heavily when they are overconfident.

Step 4a: Minimize the loss: Compute gradients (backprpagation)

- We calculate the **gradient (derivative) of the loss function** w.r.t. each weight.
- This tells us **how much each weight contributes to the error**.

For Softmax + Cross-Entropy, the gradient simplifies to:

$$\frac{\partial L}{\partial w_j} = (p_j - y_j)x$$

where:

- p_j is the predicted probability for class j,
- y_j is **1** if it's the correct class, else **0**,
- x is the input feature.

Step 4b: Minimize the loss: Update weights: Gradient Descent

Gradient Descent updates weights using:

$$w_j = w_j - \alpha \cdot \frac{\partial L}{\partial w_j}$$

where:

- α (learning rate) controls step size.
- If the gradient is large → big update.
- If the gradient is small → small update.

This **reduces the loss step by step**, making the model better at predicting.

We now have a trained model with updated weights (coefficients)

Step 5: Make predictions

- After training, we get the final weights β.
- For a new input x, we compute:

$$z = \beta_0 + \beta_1 x_1 + ... + \beta_n x_n$$

- Convert it into a probability:

$$p = \frac{1}{1 + e^{-z}}$$

- Apply a **decision rule**:
  - If $p \geq 0.5$, classify as **1**.
  - If $p < 0.5$, classify as **0**.

For multi class, the model extends to softmax regression.
Softmax is a mathematical function used to convert raw scores (logits) into probabilities for multi-class classification

Modeling – Logistic Regression

Advantages

- Simple, interpretable, and computationally efficient.
- Provides probabilistic outputs (confidence scores).
- Performs well for linearly separable data.
- Supports regularization to handle high-dimensional data.

Limitations

- Assumes linear relationships between features and the target variable.
- Sensitive to outliers and multicollinearity.
- Limited performance on non-linear and imbalanced datasets.
- May overfit in high-dimensional settings and require careful feature engineering.

Log loss

Log Loss, also known as cross-entropy loss, measures the performance of a classification model where the prediction output is a probability value between 0 and 1.

Log Loss heavily penalizes confident and wrong predictions.

It provides a more nuanced view of the model's performance compared to accuracy, especially for imbalanced datasets.

Limitations:

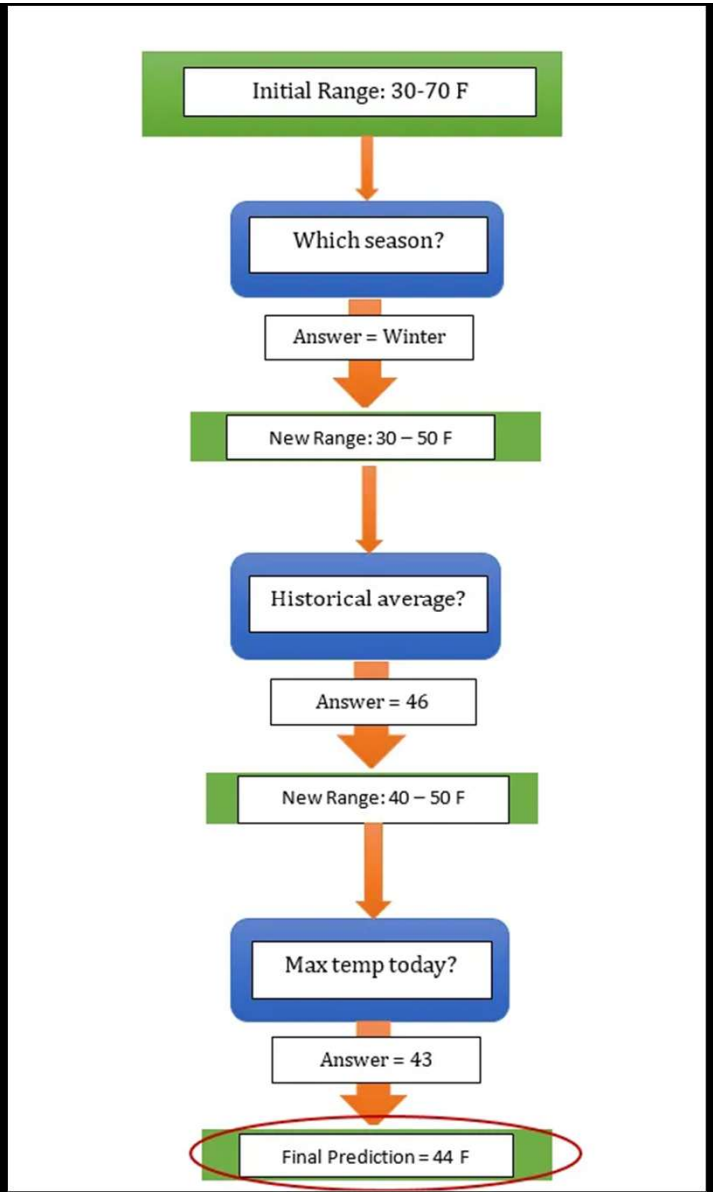
- Can be sensitive to outliers.
- Not as intuitive to interpret as other metrics like accuracy or F1-score.

Model : Decision Tree

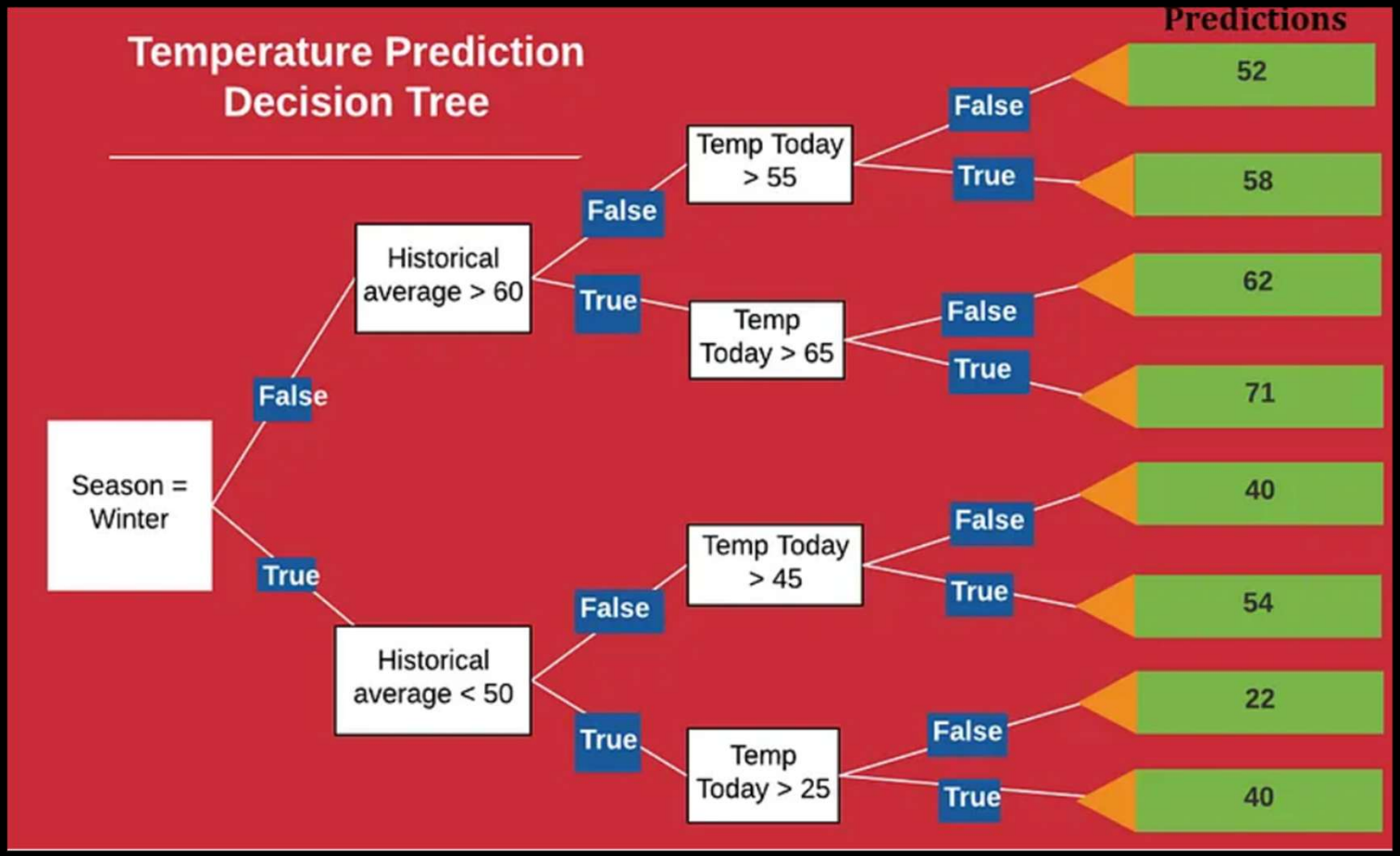
Problem statement

predicting the tomorrow’s maximum temperature for our city.

https://medium.com/towards-data-science/decision-tree-classifier-explained-a-visual-guide-with-code-examples-for-beginners-7c863f06a71e

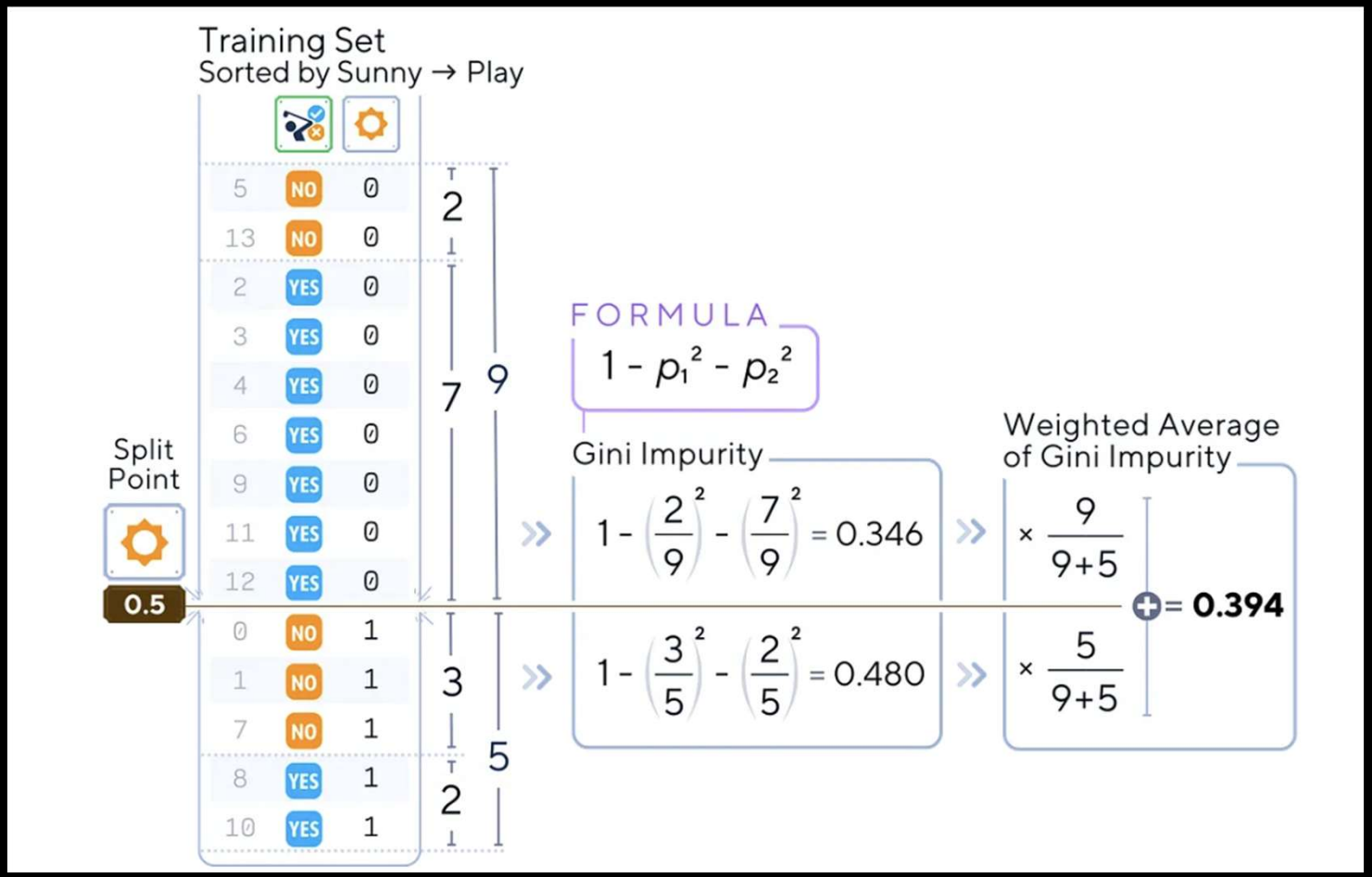


Model : Decision Tree



Model : Decision Tree

Train the model to learn to predict whether person will goto play golf or not? – Gini Impurity



Decision Tree: Based on CART

CART stands for Classification and Regression Trees, a popular decision tree methodology introduced by Leo Breiman, Jerome Friedman, Richard Olshen, and Charles Stone in their 1984 book Classification and Regression Trees.

It is a systematic framework for:

- Classification (when the target variable is categorical).
- Regression (when the target variable is continuous).
- CART benefits: handles both classification and regression trees, non parametric (not dependent on distribution), estimates feature importance
- It splits data using impurity or variance reduction, grows until a stopping rule, and typically uses pruning to avoid overfitting.
- CART's design has influenced many other tree-based models (e.g., Random Forest, Gradient Boosted Trees).

Model : Decision Tree

How it works:
Decision trees recursively split the data based on feature values to create a tree-like structure. At each node:
1.Select the best feature to split on (using metrics like Gini impurity or information gain)
2.Split the node into child nodes
3.Repeat until a stopping criterion is met (e.g., max depth, min samples per leaf)

Evaluation Metrics:
•For classification: Accuracy, Precision, Recall, F1-score
•For regression: MSE, RMSE, MAE

Hyperparameters:
•Max depth
•Min samples split
•Min samples leaf
•Max features
•Criterion (Gini impurity or information gain)

Decision tree: How does it predict numerical outcome?

It predicts numeric values by dividing the data into regions where the outputs are similar

It then uses a simple aggregate (like the mean) of the target values in each region.

This approach works effectively because the predictions are based on the finite set of training data values, sidestepping the issue of the theoretically infinite range of numbers.

It also makes it deterministic within the range of values in the training data

Gini Coefficient

The Gini impurity ranges between 0 and 1.

- A Gini of 0 means the node is perfectly pure (i.e., all samples in that node belong to a single class).
- A Gini of 1 (theoretical max, approached if classes are perfectly evenly split) indicates maximum impurity or perfect heterogeneity among classes in a node.

Probability interpretation: You can think of Gini impurity as the probability of misclassifying a sample if one were to randomly pick a label from the distribution of labels in the node

Gini impurity is simpler to compute than other common impurity measures like entropy .

Entropy

Entropy is a fundamental concept from information theory representing uncertainty or impurity.

In decision trees, it's used to decide which feature-split yields the greatest information gain (i.e., reduction in entropy).

Numerically, it ranges from 0 (all samples in one class) to log2(C) (classes are equally distributed).

Entropy vs. Gini Impurity
Gini Impurity:
Both measure "impurity" in a classification setting:

They often lead to similar splits.
Entropy has a stronger penalty for very impure nodes (because of the log term).
Gini can be slightly faster to compute (no logarithms), but in practice, the differences are usually small.

Model : Decision Tree

Advantages

- No need of feature scaling or transformation
- Handles both categorical and numeric data
- No assumptions about distribution
- Explainability, easy to understand

Limitations

- Overfitting
- Biased when data is imbalanced
- Unstable (High variance)
- Inefficient with high dimensional data

Exercise: Build a decision tree

https://medium.com/towards-data-science/decision-tree-classifier-explained-a-visual-guide-with-code-examples-for-beginners-7c863f06a71e

Models : Random Forest

A Random Forest is an ensemble learning method that builds multiple decision trees and merges their outputs to improve prediction accuracy and control overfitting.

Models : Random Forest

Ensemble Approach:
Constructs a collection of decision trees (the "forest") rather than relying on a single tree.

Bootstrap Aggregation (Bagging):
Each tree is trained on a random subset (with replacement) of the training data, which increases robustness by reducing variance.

Random Feature Selection:
At each split in a tree, a random subset of features is considered, ensuring that trees are decorrelated and capturing diverse patterns in the data.

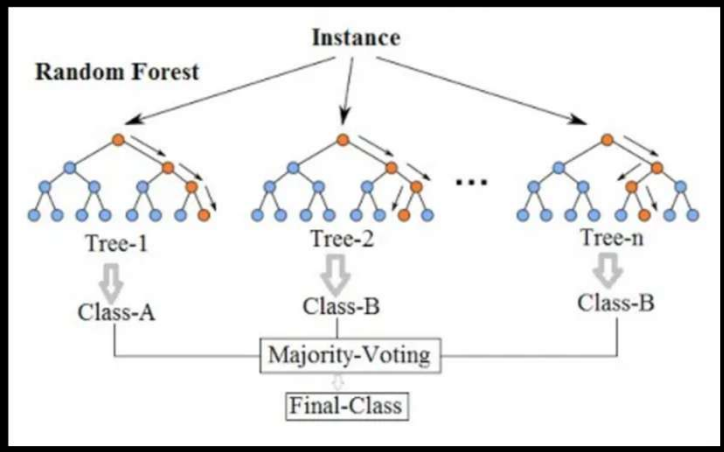
Aggregation of Predictions

Classification: Uses majority voting (mode) across trees.
Regression: Uses the average of predictions from all trees.

Models : Random Forest

Decision trees have high variance

Models such as Random forest Gradient Boosting helps minimize the variance



Ensemble models

Random Forest Model

Uses **Bagging** technique to reduce variance.
Bagging is an ensemble learning technique that builds multiple independent models (in the case of Random Forest, these are decision trees) using different random subsets of the training data.

Gradient Boosting Model

Boosting builds trees sequentially, where each new tree tries to correct the errors made by the previous ones. The trees are not independent and boosting focuses on **reducing bias** rather than variance.

Models : XG Boost

XGBoost (**Extreme Gradient Boosting**) is a highly efficient and scalable **machine learning algorithm** based on **gradient boosting**. It is widely used for structured/tabular data problems and has gained popularity due to its **speed, accuracy, and scalability**.

It is an **ensemble learning method** that builds decision trees **sequentially**, where each new tree corrects the errors of the previous trees by focusing on the most misclassified samples.

XGBoost is optimized with:

- **Parallel and Distributed Computing** (fast training)
- **Regularization (L1 & L2)** (reduces overfitting)
- **Handling Missing Data Automatically**
- **Tree Pruning (reduces unnecessary splits)**

It is particularly useful for **classification, regression, and ranking tasks**.

XGBoost Hyperparameters

Parameter	Effect
`n_estimators`	Number of trees (boosting rounds)
`learning_rate`	Shrinkage factor for tree outputs
`max_depth`	Maximum depth of each tree (controls complexity)
`subsample`	Fraction of data used per tree (avoids overfitting)
`colsample_bytree`	Fraction of features used per tree
`gamma`	Minimum Gain required for a split (pruning)
`lambda`	L2 Regularization (controls complexity)
`alpha`	L1 Regularization (feature selection)
`min_child_weight`	Minimum sum of Hessians for a leaf (avoids small splits)

XGBoost key characteristics

- ✅ XGBoost builds trees **sequentially**, where each tree corrects the errors of the previous ones.
- ✅ It **uses the same training data**, but the predictions get updated iteratively.
- ✅ It decides **how to split** based on **Gain calculation (Gradient & Hessian)**.
- ✅ It **reduces errors** using **gradient-based optimization** and **regularization**.
- ✅ It's **fast, efficient, and scalable** due to parallelism and hardware optimizations.

Random Forest Vs XGBoost

Feature	XGBoost	Random Forest
Type	Boosting	Bagging
Tree Building	Sequential (each tree corrects the previous one)	Parallel (trees are built independently)
Bias-Variance Tradeoff	Lower bias, slightly higher variance	Higher bias, lower variance
Overfitting	More prone, but mitigated with regularization (L1, L2)	Less prone to overfitting due to averaging
Handling of Errors	Learns from previous mistakes, reducing error gradually	Averaging of predictions reduces variance but doesn't correct past mistakes
Computation Speed	Optimized (uses parallelism, GPU, and distributed computing)	Slower than XGBoost for large datasets
Performance on Small Data	Performs well but may overfit	Works well, more stable
Performance on Large Data	Faster and more efficient	Slower, needs more trees

Models : K-Nearest Neighbors (KNN)

How it works:

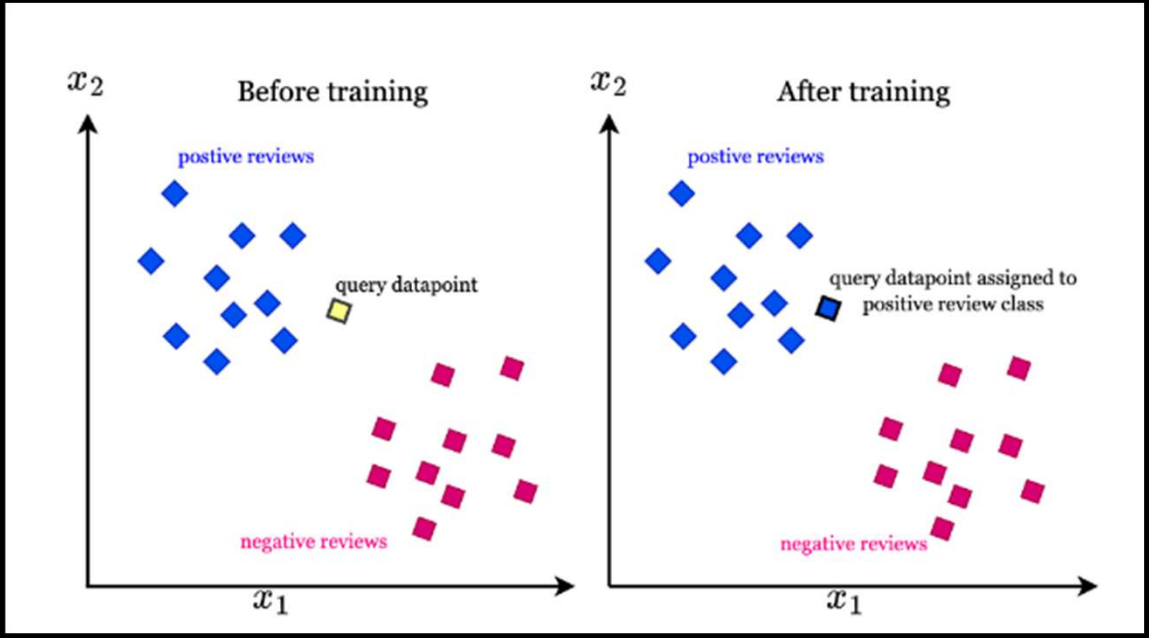
K-Nearest Neighbors is a supervised, non-parametric, instance-based learning algorithm used for both classification and regression tasks. The core idea is that similar data points tend to have similar outcomes.

Distance Metrics

- Euclidean distance (most common)
- Manhattan distance
- Cosine similarity

Hyperparameters

- K (number of neighbors)
- Weight function
- Distance metric
- Algorithm



x1, x2 represent features used to classify the reviews (such as number of negative words)

Models : K-Nearest Neighbors (KNN)

Advantages

- Simple and intuitive algorithm
- No assumptions about data distribution (non-parametric)
- Can be used for both classification and regression

Limitations

- Computationally expensive, especially with large datasets
- Sensitive to the scale of features
- Struggles with high-dimensional data (curse of dimensionality)
- Sensitive to imbalanced datasets
- Requires feature scaling for optimal performance
- Does not work well with categorical features unless properly encoded

https://intuitivetutorial.com/2023/04/07/k-nearest-neighbors-algorithm/

Models : K-Means

K-Means is a popular unsupervised machine learning algorithm used for **clustering** data points into **K groups** based on feature similarity.

It aims to minimize intra-cluster variance by iteratively updating cluster centroids.

The number of clusters K needs to be specified beforehand

K-Means - Usecases

Customer Segmentation – Group customers based on purchasing behavior.

Anomaly Detection – Detect unusual patterns in network security, fraud detection.

Image Segmentation – Cluster pixels for object recognition or background removal.

Document Clustering – Organize text documents based on content similarity.

Genetic Data Analysis – Identify similar gene expression profiles in bioinformatics.

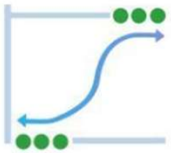
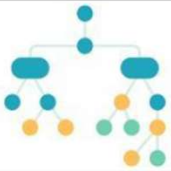
Market Basket Analysis – Find groups of products frequently purchased together.

K-Means Vs KNN

Feature	K-Means	K-Nearest Neighbors (KNN)
Algorithm Type	**Unsupervised Learning (Clustering)**	**Supervised Learning (Classification or Regression)**
Goal	Group data points into **K clusters** based on similarity	Classify a new data point based on majority vote of its **K nearest neighbors**
Output	Cluster labels (e.g., "Group 1", "Group 2")	Class labels (e.g., "Disease" or "No Disease")

Hyperparameters - summary

Hyperparameter Tuning in Machine Learning

Representation	Algorithm Name	Hyperparameter
	Linear Regression	Regularization parameter (alpha for Ridge/ Lasso Regression
	Logistic Regression	C (Inverse of regularization strength), penalty (L1, L2)
	Decision Tree	Max_depth, min_samples_splits, min_samples_leaf, criterion
	K- Nearest Neighbors	n_neighbors, weights, metric
	Support Vector Machines	C, Kernel, gamma, degree (for polynomial kernel)

Evaluating models – Variance and Bias

Variance and bias are two key sources of error in a machine learning model.

Bias

- Refers to the error introduced due to oversimplified assumptions in the learning algorithm.
- A high-bias model fails to capture the complexities of the data, leading to underfitting.
- Example: A linear regression model applied to a dataset with a non-linear pattern.

Variance

- Refers to the error due to excessive sensitivity to fluctuations in the training data.
- A high-variance model learns the noise along with the signal, leading to overfitting.
- Example: A deep decision tree that memorizes the training data instead of generalizing.

Example: reducing variance in a decision tree model

To Reduce Bias (Underfitting):
- Increase the number of trees (n_estimators).
- Reduce pruning of individual trees (increase max_depth or min_samples_split).
- Reduce regularization (decrease min_samples_leaf).
- Use a more complex base model if Random Forest is too simplistic.

To Reduce Variance (Overfitting):
- Limit the depth of trees (max_depth).
- Increase the min_samples_split.
- Increase the minimum number of samples per leaf (min_samples_leaf).
- Reduce correlation among trees by using feature bagging (adjust max_features).
- Use Bootstrapping (bootstrap=True) to increase robustness.
- Reduce the number of trees if they are too correlated.

Example: reducing variance in a decision tree model contd.

Hyperparameter Tuning for Bias-Variance Tradeoff

• Perform cross-validation to find the optimal settings.
• Use Grid Search or Random Search to fine-tune max_depth, min_samples_split, min_samples_leaf, and max_features.
• Implement Out-of-Bag (OOB) scoring (oob_score=True) to evaluate performance without needing a validation set.

Alternative Methods

• Try Gradient Boosting or XGBoost for potentially better performance if bias is high.
• Use Dimensionality Reduction (PCA, Feature Selection) to remove noisy features that can contribute to variance.

Cross Validation

Cross-validation is a resampling technique used to assess the performance and generalizability of a machine learning model.

Done during scoring/evaluation phase using APIs available in the libraries

Helps to avoid overfitting by training and testing the model on **different subsets of data.**

Why Use Cross-Validation?

- Prevents overfitting by ensuring the model generalizes well to unseen data.
- Provides a more reliable estimate of model performance.
- Useful when data is limited, as it maximizes the use of available data.

Types of cross validation

K-Fold Cross-Validation

- The dataset is split into K subsets (folds).
- The model is trained on K-1 folds and tested on the remaining one.
- This process repeats K times, with each fold used as a test set once.
- The final performance metric is the average of all iterations.

Stratified K-Fold Cross-Validation

- Similar to K-Fold but preserves the proportion of classes
- seful for imbalanced datasets

..there are more..

Demo of cross validation

Steps for a complete ML solution

- Data Gathering
- Data cleaning
- Feature engineering
- Model selection
- Split into train and test data
- Model fitting with training data
- Model testing with test data
- Evaluate the model
- Adjust hyperparameters (tune the model)
- Re-evaluate
- Deploy