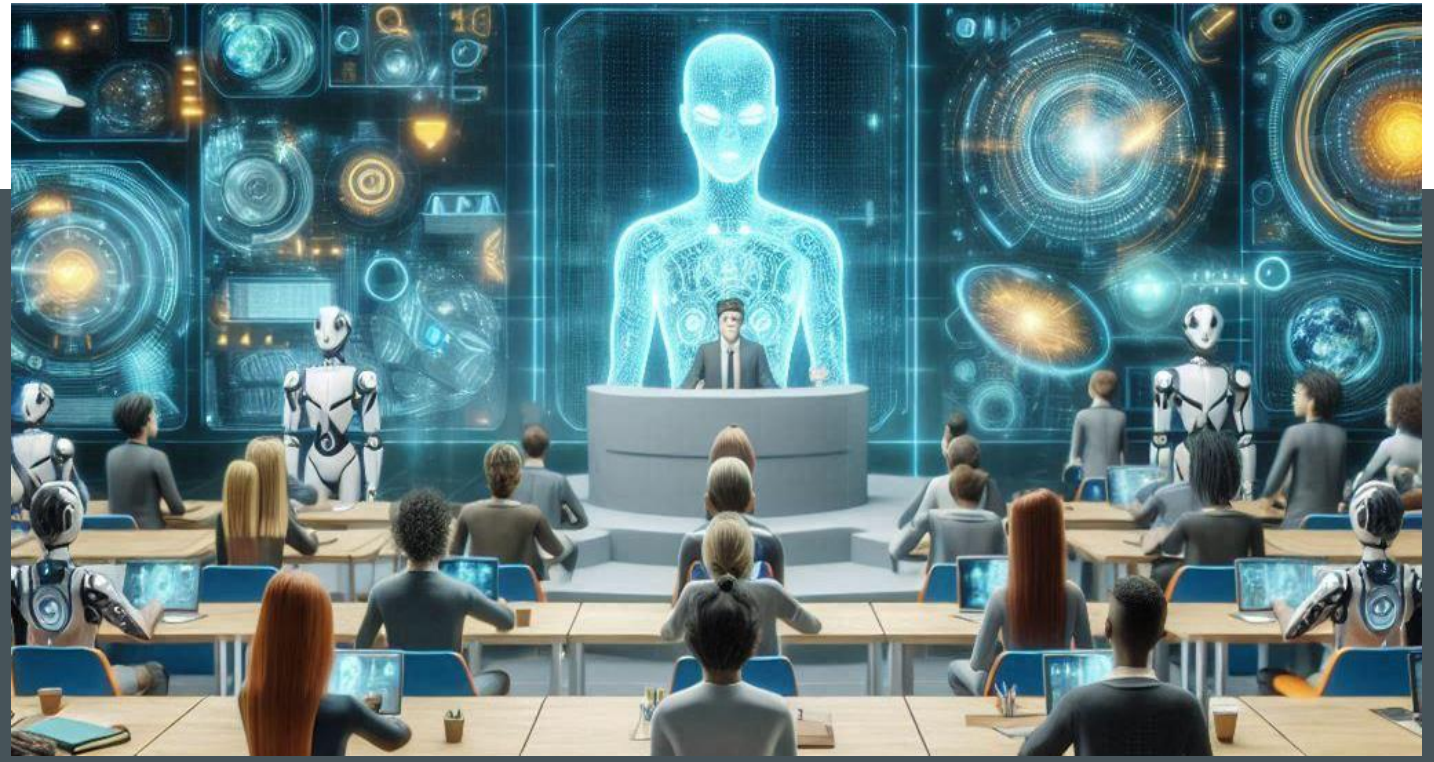
CS 180 INTRODUCTION TO DATA SCIENCE

WRAP-UP AND FINAL REVIEW

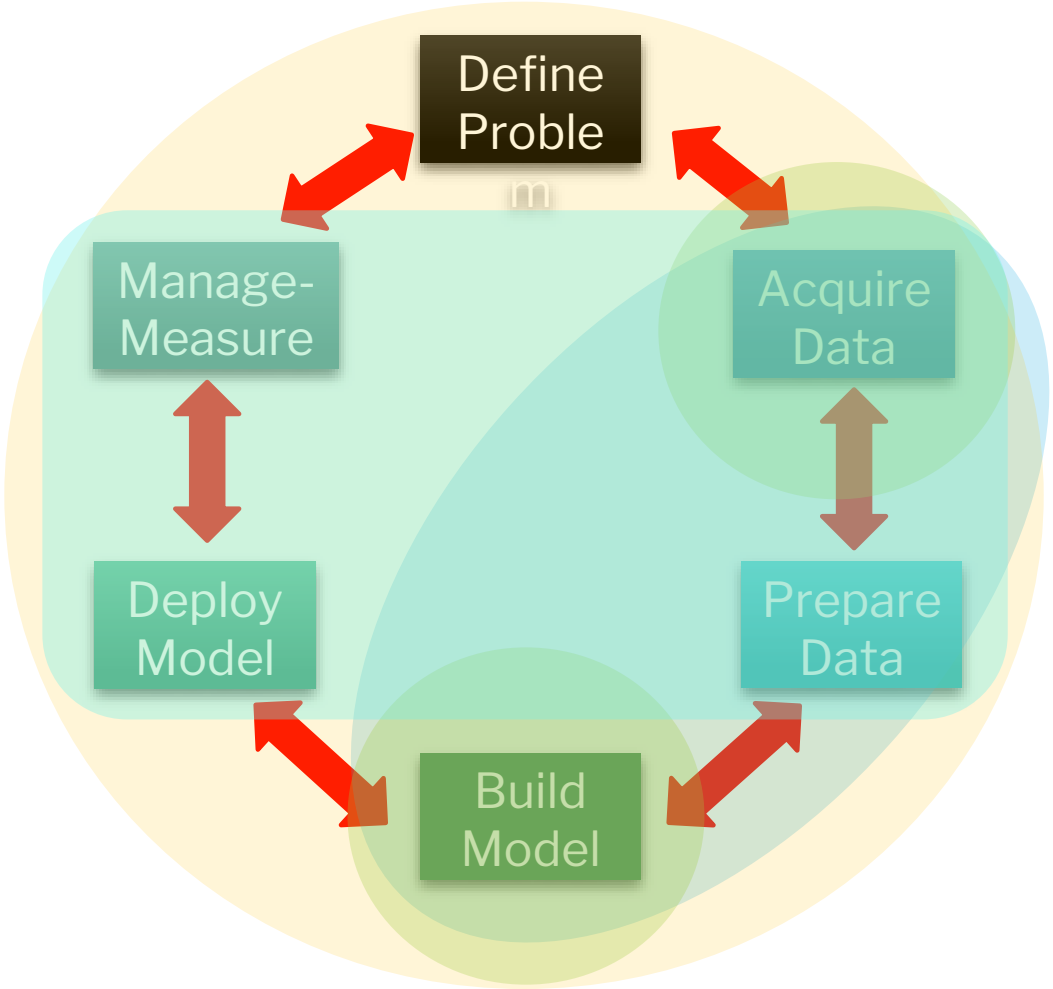


Created by DALL-E Prompt: “Artificial Intelligence Classroom”

General Data Science

BYU

YOUR DATA SCIENCE JOURNEY



BYU

SUMMARY: DATA ACQUISITION PREPARATION EXPLORATION

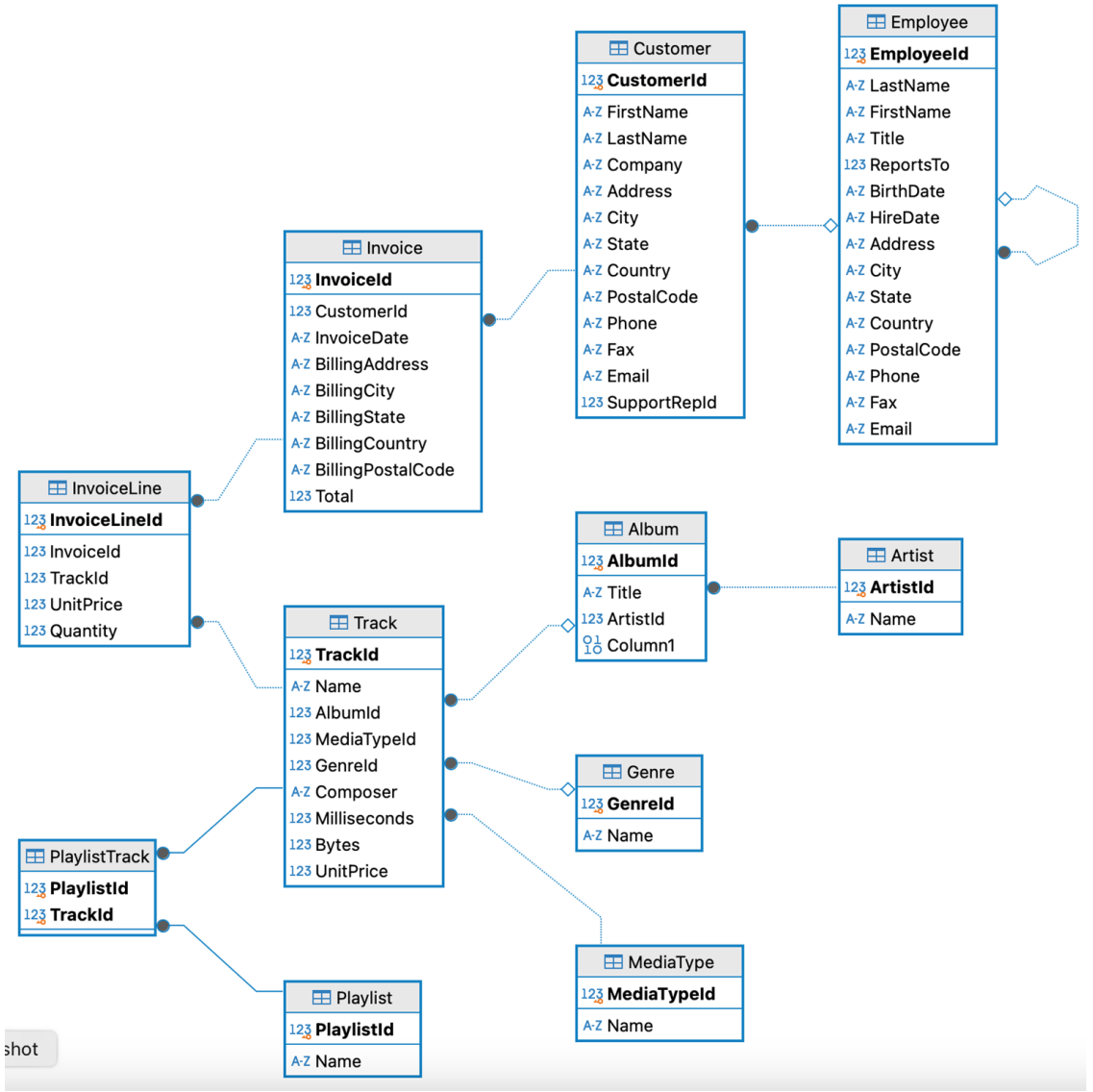
Data Analysis Processes	Tools
Acquire Data	pd.read_csv(), pd.read_excel(), df.to_csv(index = False), df.to_excel(index = False),
Profile Data	df.head(), df.tail(), df.info(), df.describe(include='all'), df.dtypes, df.shape
Manipulate Data	df.groupby(), df.pivot_table(), df.insert()
Clean Data Missing Data Imputing Data Duplicates Outliers Data Types	df.isnull().sum(), df.isna(), df.fillna(value=), df.drop(), df.drop_duplicates(), df.dropna(), df.replace(), df.astype(),
Exploratory Data Analysis Univariate Multivariate Visualization	df.plot(), plt.show(), df.plot.scatter(), df.plot.box(), plt.hist() plt.subplots(figsize=), sns.histplot(), sns.kdeplot(), sns.boxplot(), sns.violinplot() df.mean(), sns.scatterplot(), sns.swarmplot(), sns.stripplot(), plotly.express
Transform Data	preprocessing.scale(), preprocessing.MinMaxScaler().fit_transform() [Part of Modeling because we need to wait until after train/validation/test split]

SQL

BYU

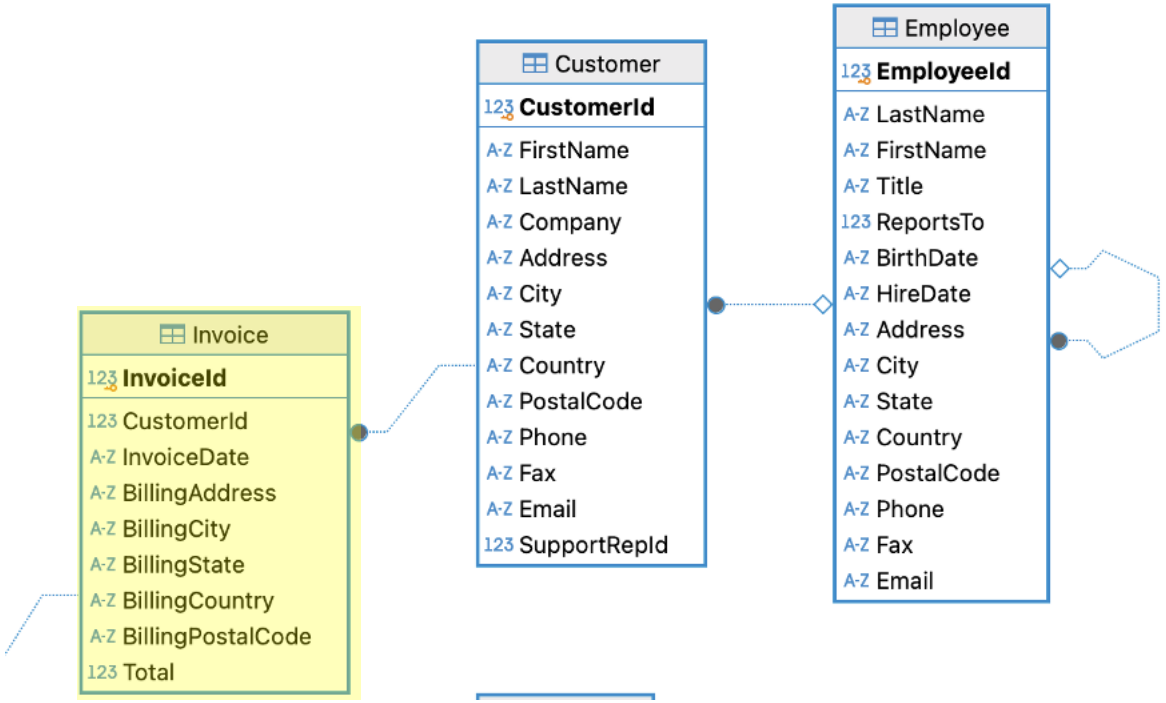
DATABASE SCHEMA

- The following is **Entity-Relationship Diagram (ERD)** and is used to summarize database schemas that create **Data Models**.
- This example represents a music store with key entities: Customer, Invoice, InvoiceLine, Track, Album, Artist, Genre, MediaType, Playlist, Employee as separate tables.



KEYS

- Database rows (see **Invoice** Table to the right)
  - Contain information about instance **Invoice** (CustomerID, InvoiceDate, BillingAddress, etc.)
  - Each **Invoice** is represented by exactly 1 row
  - Rows have identifiers to ensure that:
    - Data for each Invoice exists on only 1 row of the relation
    - Each row contains only data for 1 Invoice
    - **InvoiceID fills this role, thus each value of InvoiceID must be unique**
  - **Primary Key**: the key chosen as the attribute to uniquely define a row



BYU

SQL – STRUCTURED QUERY LANGUAGE

- Non-procedural language
  - Find the name of the customer with customer-id 192-83-7465

```
select  customer.customer-name
from    customer
where   customer.customer-id = '192-83-7465'
```

  - Find the balances of all accounts held by the customer with customer-id 192-83-7465

```
select  account.balance
from    depositor, account
where   depositor.customer-id = '192-83-7465' and
        depositor.account-number = account.account-number
```

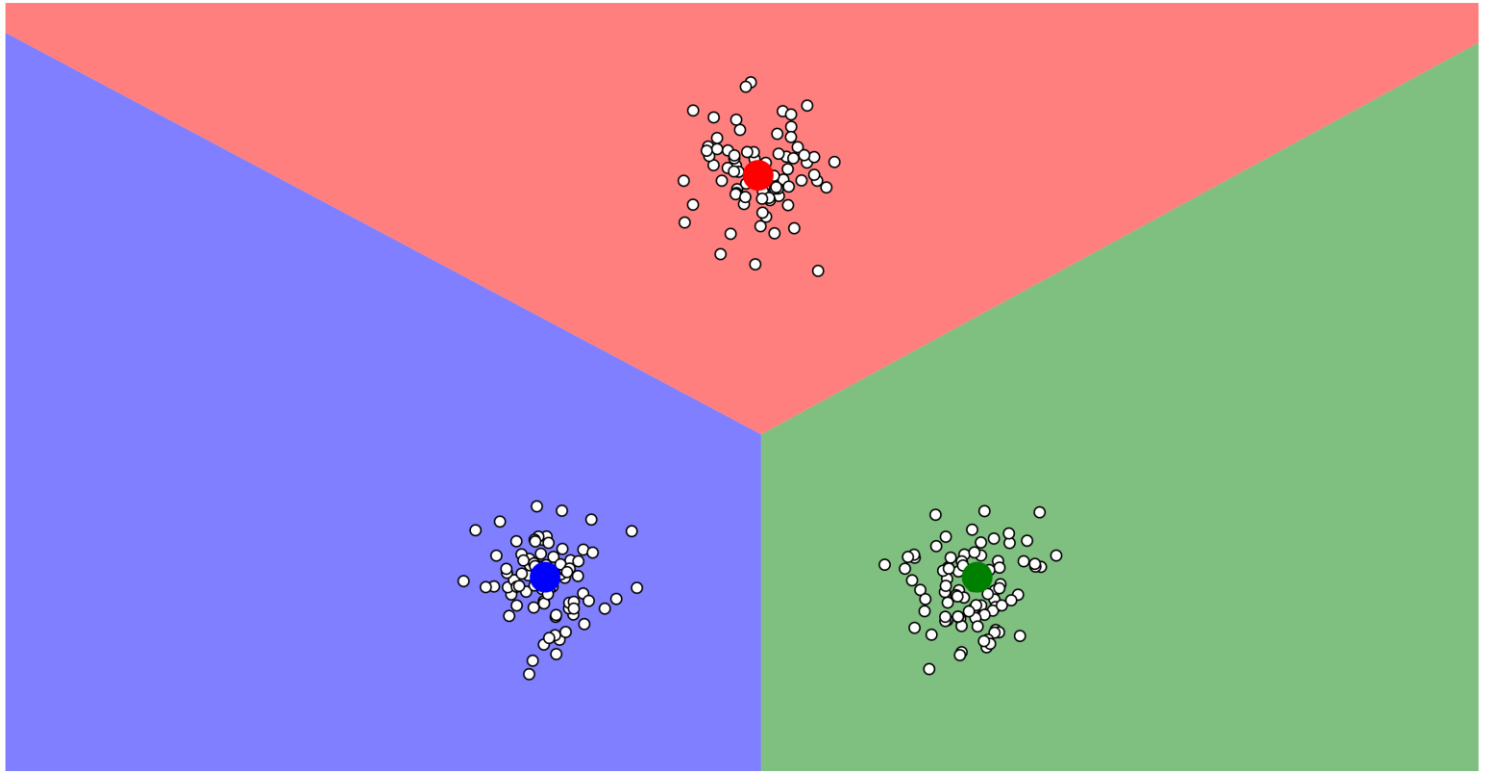
- Application programs generally access databases through ether:
  - Language extensions to allow embedded SQL (we will show how to do this with python)
  - Application program interface (e.g. ODBC/JDBC) which allow SQL queries to be sent to a database

BYU

Unsupervised Learning:

BYU

K-MEANS



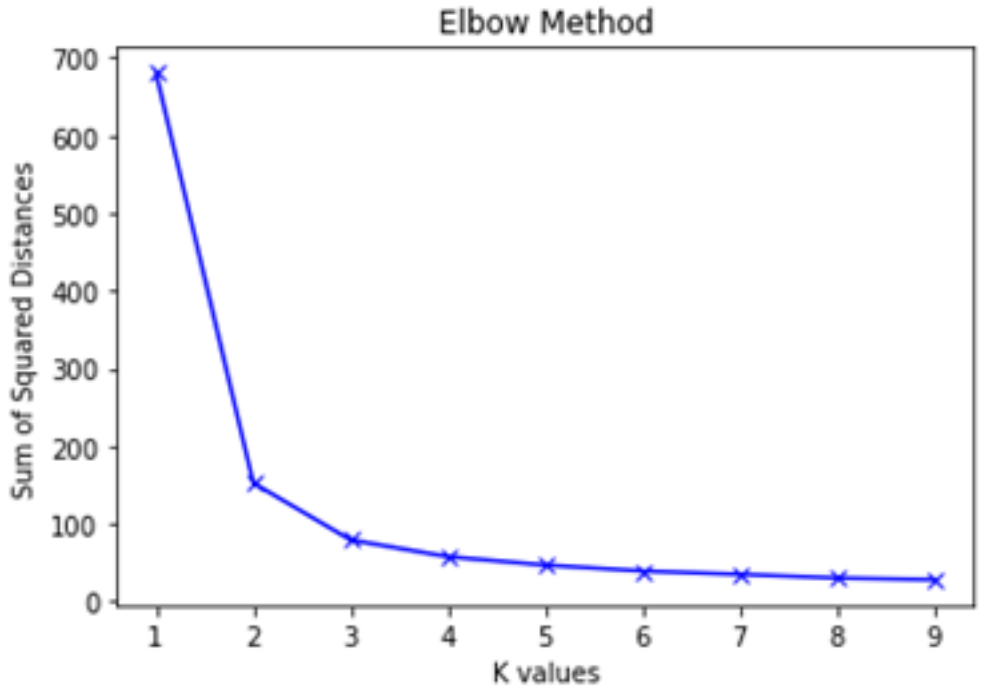
- What are some of the advantages/disadvantages of the K-Means algorithm for clustering? **When does it work well? When does it fail?**

BYU

FINDING THE OPTIMAL K VALUE USING ELBOW METHOD

Steps:

- 1. For different values of K, execute the following steps:
- 2. For each cluster, calculate the sum of the squared distance of every point to its centroid.
- 3. Add the sum of squared distances of each cluster to get the total sum of squared distances for that value of K.
- 4. Keep adding the total sum of squared distances for each K to a list.
- 5. Plot the sum of squared distances (using the list created in the previous step) and their K values.
- 6. Select the K at which a sharp change occurs (looks like an elbow of the curve).



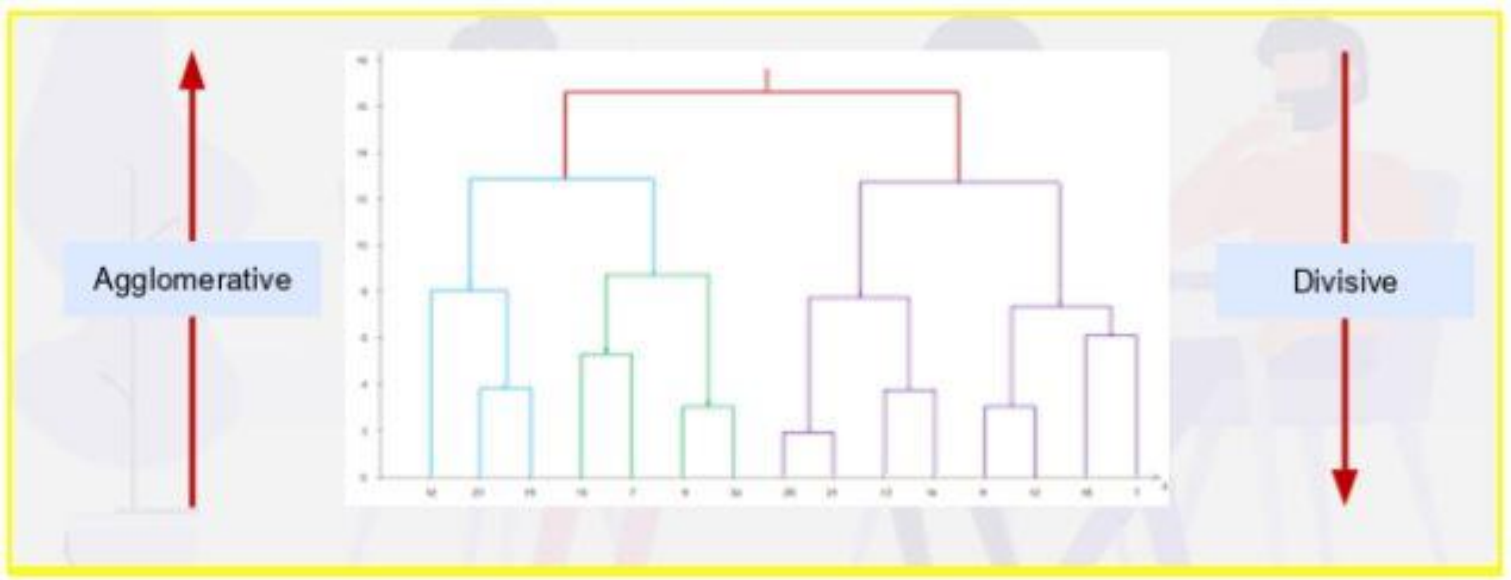
BYU

HIERARCHICAL CLUSTERING

There are two types of Hierarchical clustering techniques:

- **Agglomerative**
- **Divisive**

- Agglomerative clustering is a **bottom-up** approach. Start with every point as a cluster and combine points till only a single cluster. (most common)
- Divisive clustering is a **top-down** approach. Start with one large cluster and divide it into two, three, four, or more clusters.

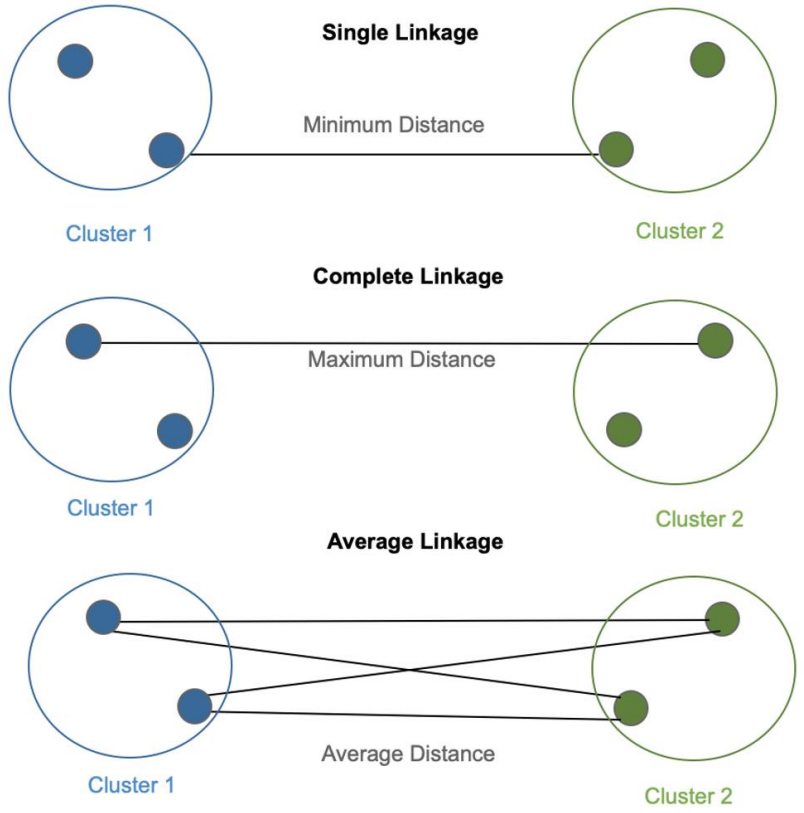


BYU

LINKAGE METHODS

Linkage methods are crucial as they determine how clusters are formed. Here are some standard linkage methods used in hierarchical clustering:

1. **Single Linkage:** (aka Nearest Neighbor) This method joins clusters based on the minimum distance between their data points. It tends to create long, "stringy" clusters and is sensitive to outliers.
2. **Complete Linkage:** Connects clusters by their maximum pairwise distance, resulting in compact, spherical clusters. However, it can be sensitive to outliers as well.
3. **Average Linkage:** This method calculates the average distance between data points in two clusters. It produces balanced clusters and is less sensitive to outliers than single or complete linkage.

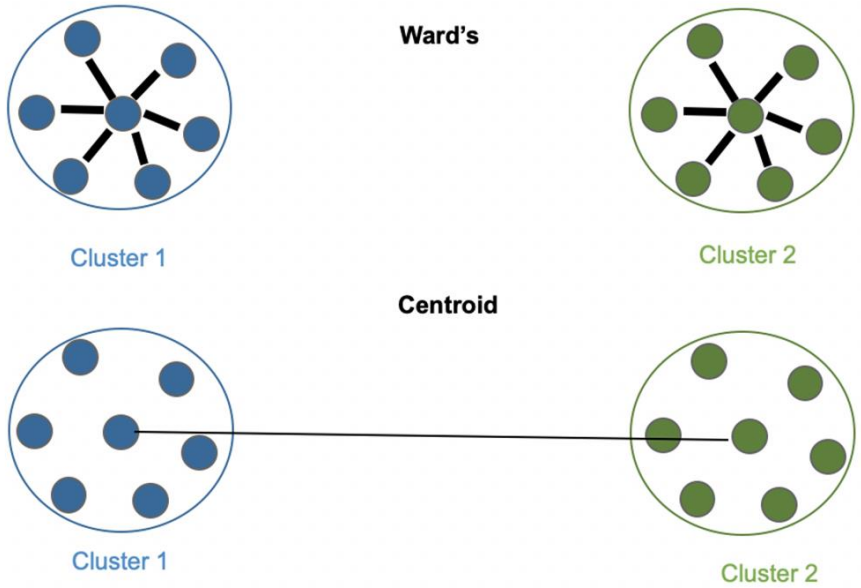


BYU

LINKAGE METHODS

1. **Ward's Linkage:** Ward's method minimizes the increase in variance within the clusters when merging them. It often produces equally sized clusters and is suitable for minimizing the variance in the data.
2. **Centroid Linkage**: Combines clusters with minimum distance between centroids (Mean, Median, Medoid, Mode)

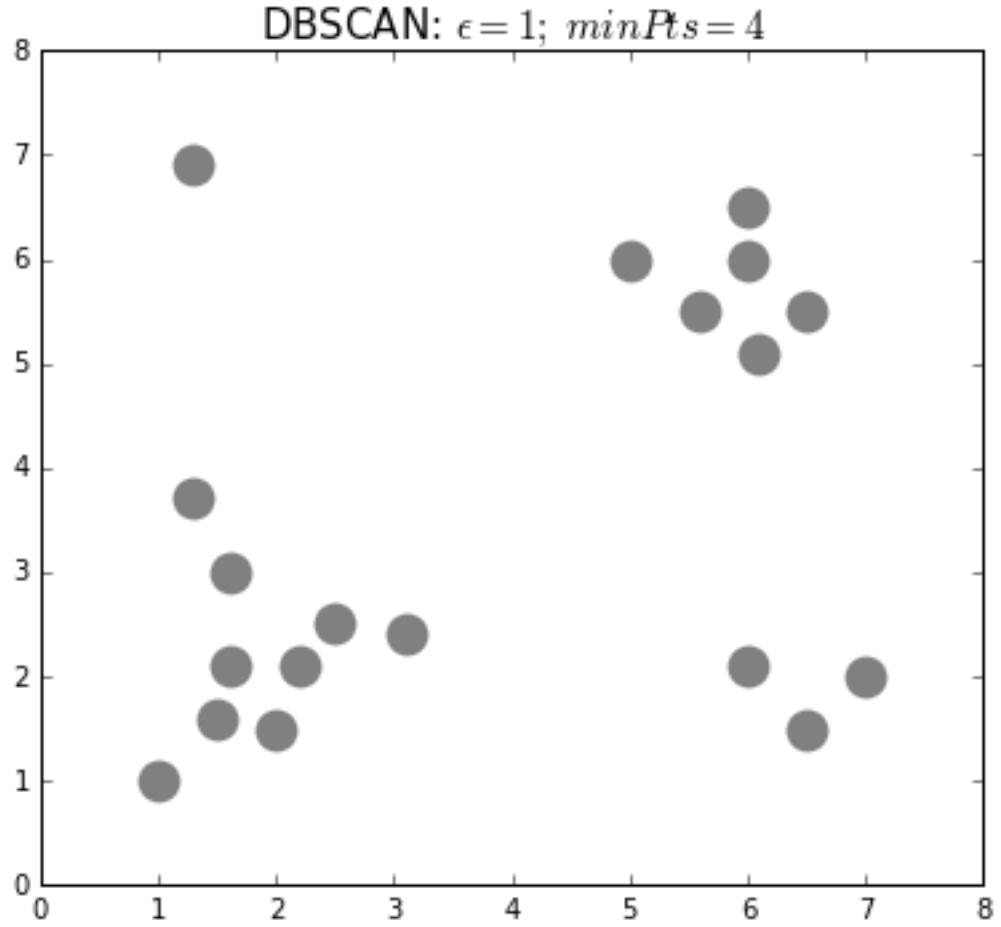
These linkage methods play a crucial role in shaping the hierarchical clustering output, and the choice depends on the specific characteristics of your data and your analysis goals.



BYU

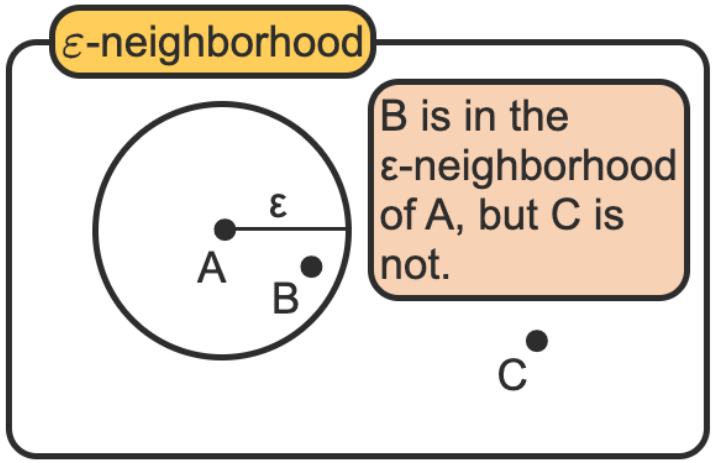
DBSCAN ALGORITHM

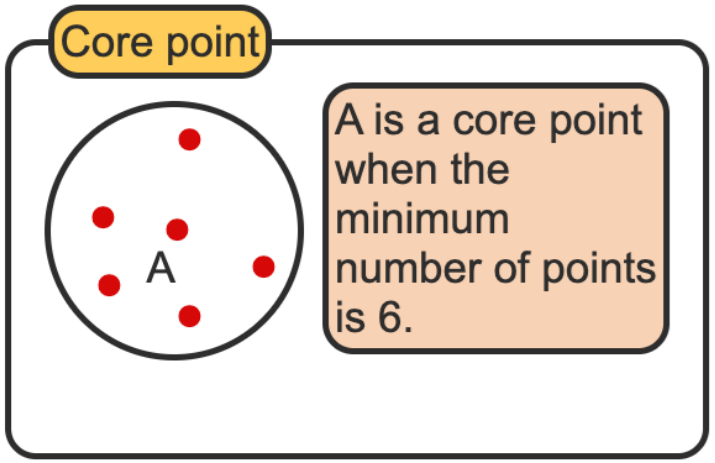
1. DBSCAN begins with an arbitrary starting data point that has not been visited. The neighborhood of this point is extracted (All points within the ε distance are neighborhood points).
2. If there are enough points (minPts) in this neighborhood, the current data point becomes the first point in the new cluster. Otherwise, the point will be labeled as noise).
3. For this first point in the new cluster, the points within its **ε** neighborhood also become part of the same cluster. This procedure of making all points in the ε neighborhood belong to the same cluster is repeated for all new points added to the cluster group.
4. Steps 2 and 3 are repeated until all points in the cluster are determined, i.e., all points within the ε neighborhood of the cluster have been visited and labeled.
5. Once we're done with the current cluster, a new unvisited point is selected, leading to a further cluster or noise discovery, this process repeats until all points are marked as visited.

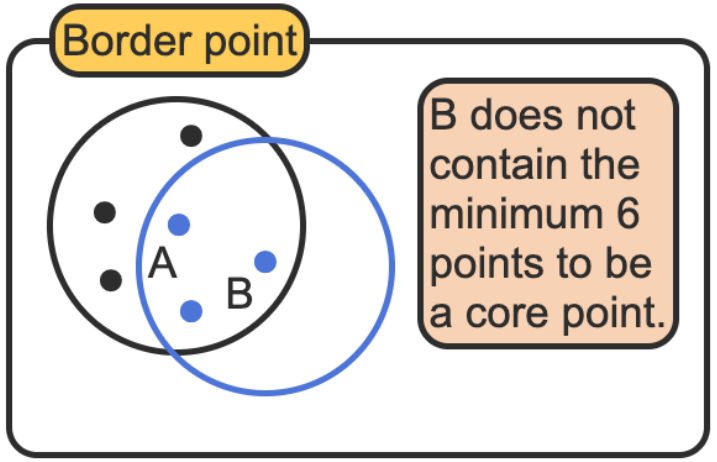


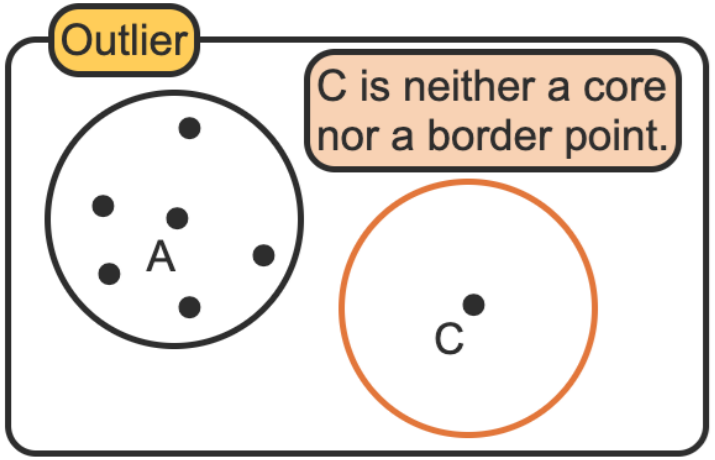
BYU

DBSCAN DEFINITIONS









BYU

Using sci-kit learn

BYU

THE SCIKIT-LEARN MODEL DEVELOPMENT PROCESS

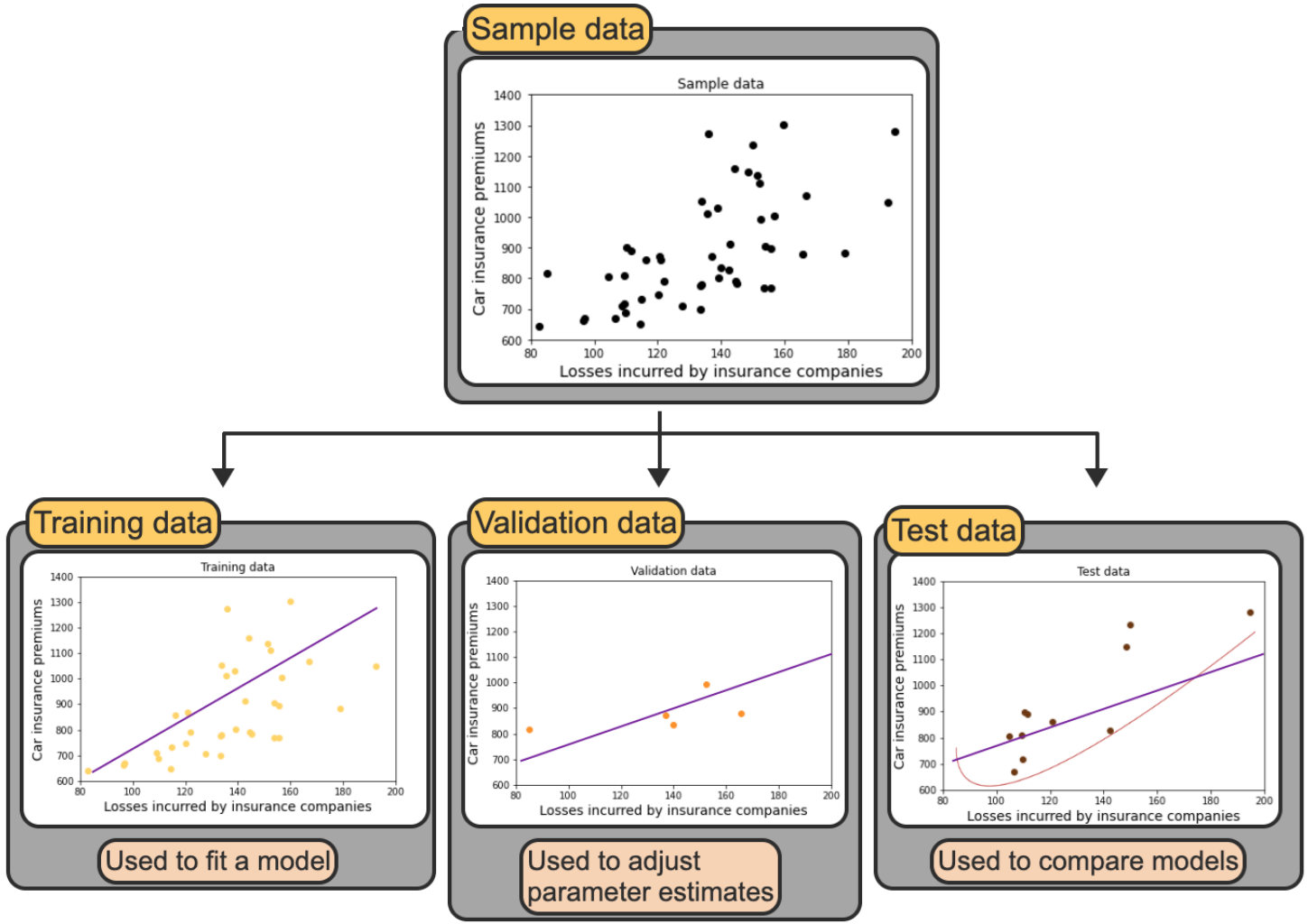
1. Load the Data set
2. Break data set into features and target
3. Create train/test split
4. Initiate the model
5. Fit the model to the training data
6. Use model to make predictions on the test data
7. Calculate model performance

```python
import numpy as np
import pandas as pd
from sklearn.datasets import load_iris
from sklearn.model_selection import train_test_split # to split the data
from sklearn.neighbors import KNeighborsClassifier   # KNN classifier
from sklearn.metrics import accuracy_score           # to evaluate the model

# 1. Load the Iris dataset
data = load_iris()

# 2. Break the data into features and labels
X = data.data    # Features
y = data.target  # Labels (classes)

# 3. Split the data into training and test sets
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.3, random_state=42)

# 4. Initialize the KNN classifier with k=3
k = 3
knn = KNeighborsClassifier(n_neighbors=k)

# 5. Train the classifier by fitting it to the training data
knn.fit(X_train, y_train)

# 6. Make predictions on the test set
y_pred = knn.predict(X_test)

# 7. Evaluate the model
accuracy = accuracy_score(y_test, y_pred)
print(f"Accuracy of KNN with k={k}: {accuracy:.2f}")
```

BYU

TRAINING, VALIDATION, AND TEST DATA SETS

- ***Training data*** is used to fit a model.
- ***Validation data*** is used to evaluate model performance while adjusting parameter estimates and conducting feature selection.
- ***Test data*** is used to evaluate final model performance and compare different models.

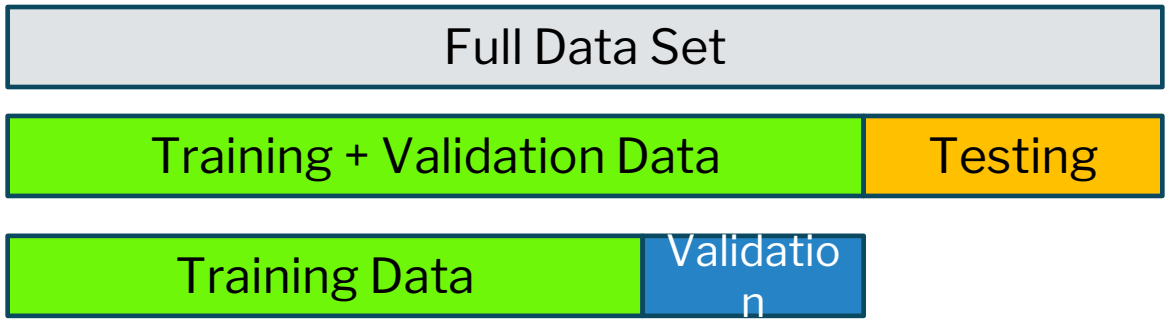


CREATING TRAINING, VALIDATION, AND TEST DATA SETS IN PYTHON

- **train_test_split** function splits the original dataset (badDrivers) into two parts:
  - **trainingAndValidationData**
  - **testData**
- **test_size** contains 20% of the data (as defined by **testProportion)**

- Next part further splits the trainingAndValidationData:
  - **trainingData**: contains data for training the model.
  - **validationData**: contains data for validating the model's performance during training.

```
# Set the proportions of the training-validation-test split
trainingProportion = 0.70
validationProportion = 0.10
testProportion = 0.20

# Split off the test data
trainingAndValidationData, testData = train_test_split(
    badDrivers, test_size=testProportion)

# Split the remaining into training and validation data
trainingData, validationData = train_test_split(
    trainingAndValidationData,
    test_size = validationProportion / trainingProportion
    # train_size = trainingProportion / (trainingProportion + validationProportion)
)
```



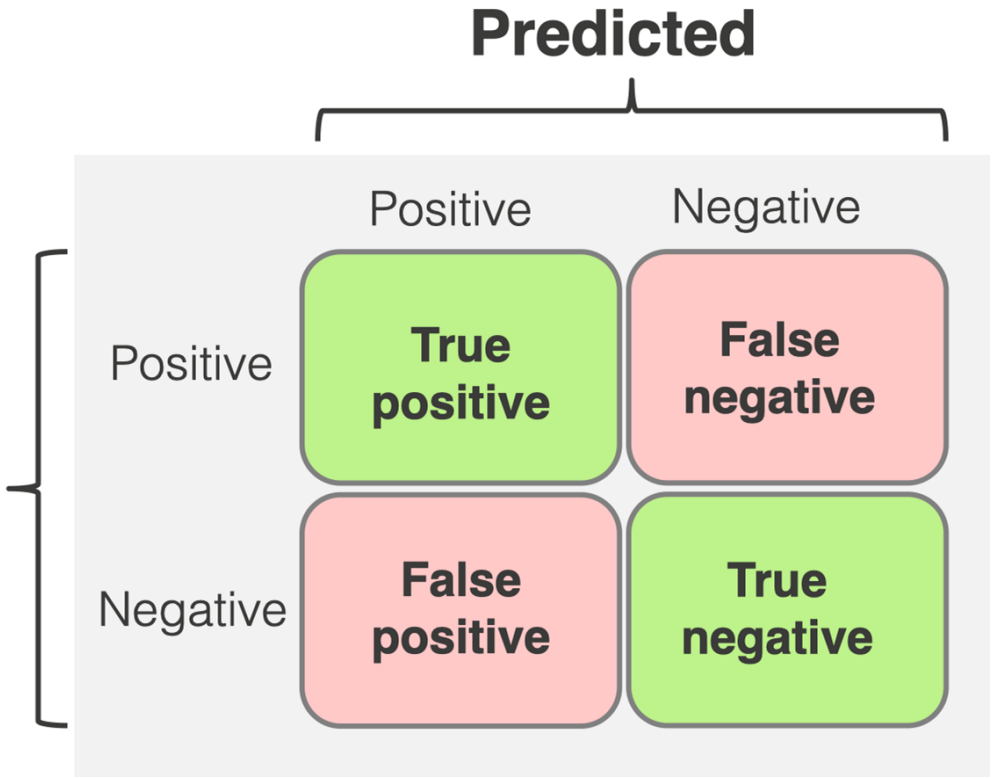
BYU

Model Performance

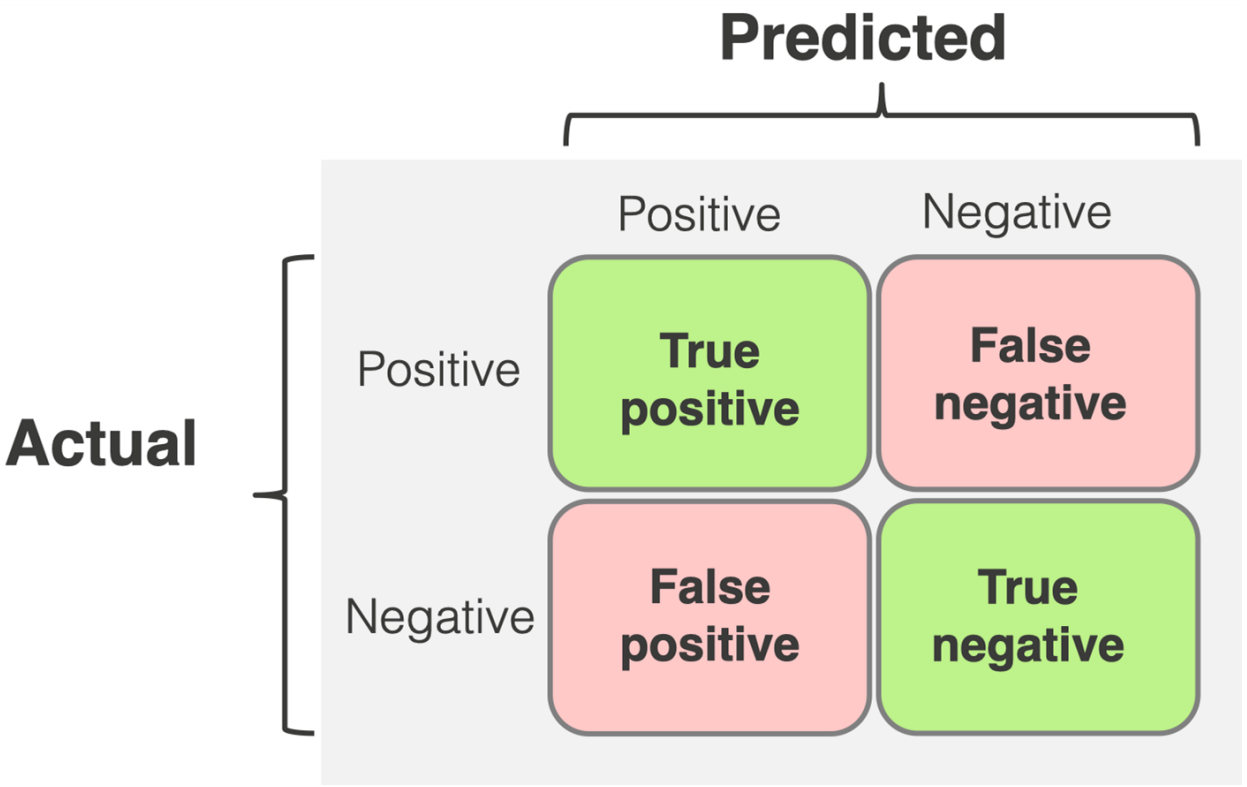
BYU

MEASURING MODEL PERFORMANCE WITH THE CONFUSION MATRIX

- A **confusion matrix** is a N x N matrix where N is total number of target categories (2x2 = Binary).
- It compares actual target values to those predicted by the **ML model**.
- Prediction results can be put into four categories:
  - **TP (True Positive):** Correctly predicted positive instances.
  - **TN (True Negative):** Correctly predicted negative instances.
  - **FP (False Positive):** Negative instances incorrectly predicted as positive.
  - **FN (False Negative):** Positive instances incorrectly predicted as negative.



ACCURACY



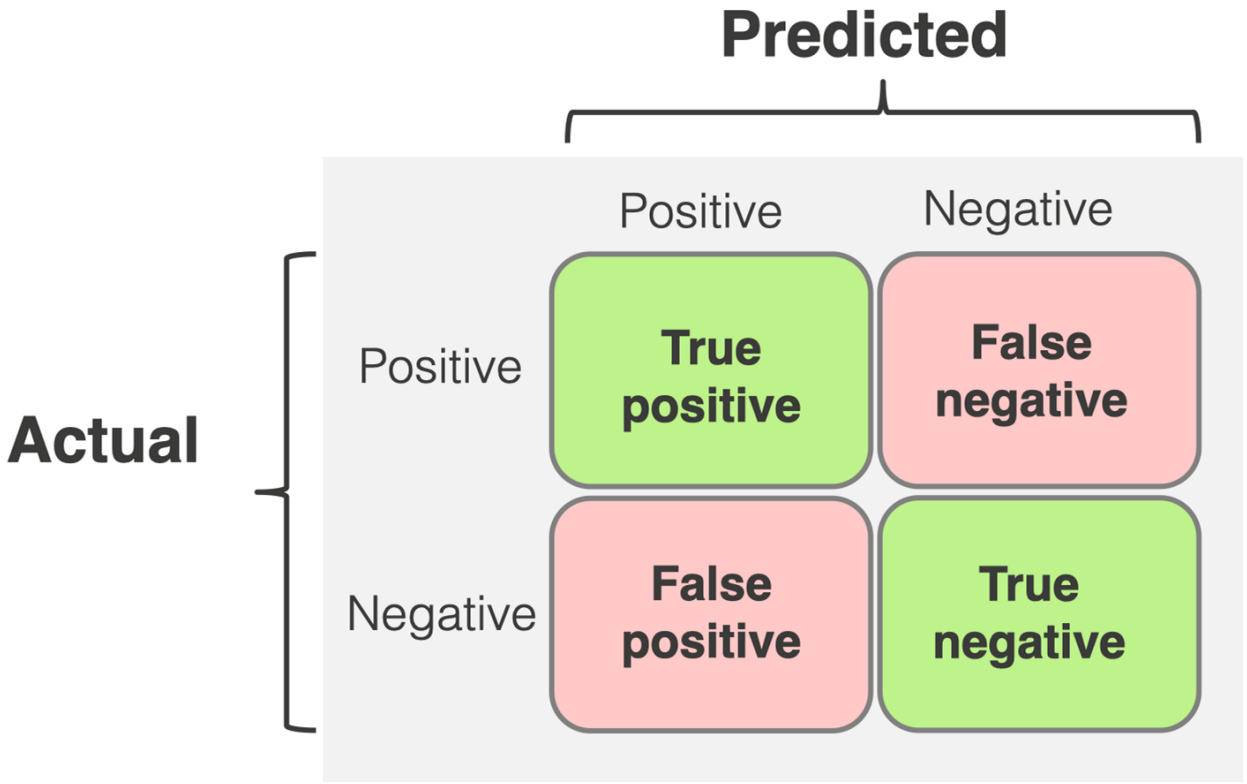
When is model accuracy NOT a good measure of performance?

$$\text{Accuracy} = \frac{\text{Correct predictions}}{\text{All predictions}}$$

$$\text{Accuracy} = \frac{TP + TN}{TP + TN + FP + FN}$$

BYU

PRECISION AND RECALL



Precision is a metric that measures how often a machine learning model correctly predicts the positive class.

$$\text{Precision} = \frac{\text{True Positives}}{\text{True Positives + False Positives}}$$

$$\text{Recall} = \frac{\text{True Positives}}{\text{True Positives + False Negatives}}$$

Recall is a metric that measures how often a machine learning model correctly identifies positive instances (true positives) from all the actual positive samples in the dataset.

F1-SCORE

- Combines two fundamental metrics: **Precision** and **Recall**, providing a single measure that balances both metrics.
- The F1-score is the **harmonic mean** of Precision and Recall, offering a balance between the two.
- It is particularly useful when the cost of false positives and false negatives is high, and when the class distribution is imbalanced.
-

$$F1 - Score = 2 * \frac{Precision * Recall}{Precision + Recall}$$

- This formula ensures that the F1-score is high only when both Precision and Recall are high.

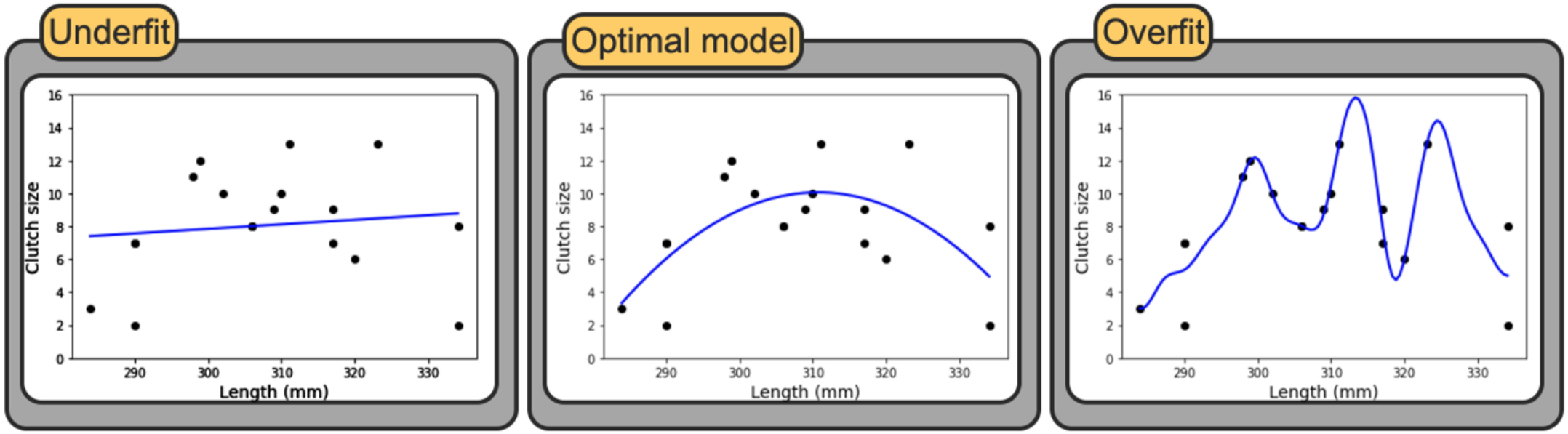
```python
from sklearn.metrics import f1_score

# True labels
y_true = [0, 1, 1, 0, 1, 1, 0, 0, 1, 0]

# Predicted labels
y_pred = [0, 0, 1, 0, 1, 1, 1, 0, 1, 0]

# Calculate F1-score
f1 = f1_score(y_true, y_pred)

print(f'F1-Score: {f1:.2f}')
```

Output:

```
makefile

F1-Score: 0.80
```

BYU

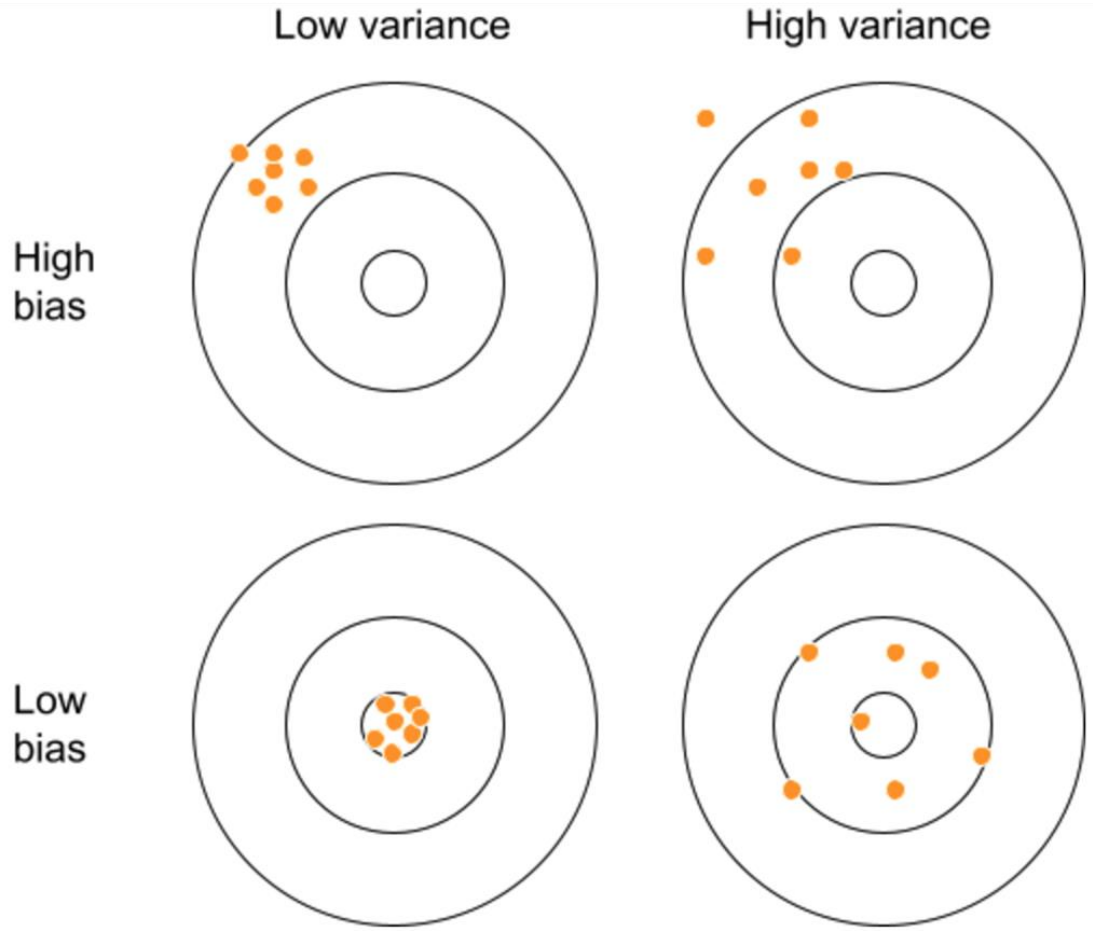
UNDER- VS OVER- FITTING OF DATA TO MODELS

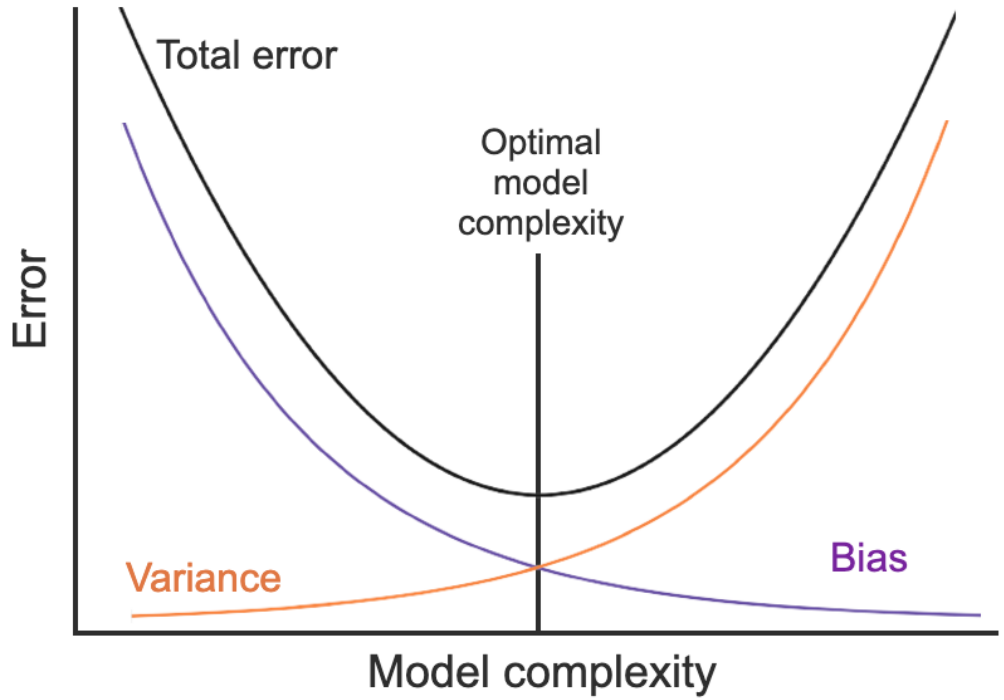


A quadratic regression fits the overall curved pattern of the data without passing through every point. As such, a model of moderate complexity is often optimal.

BYU

BIAS–VARIANCE TRADEOFF CONCEPT

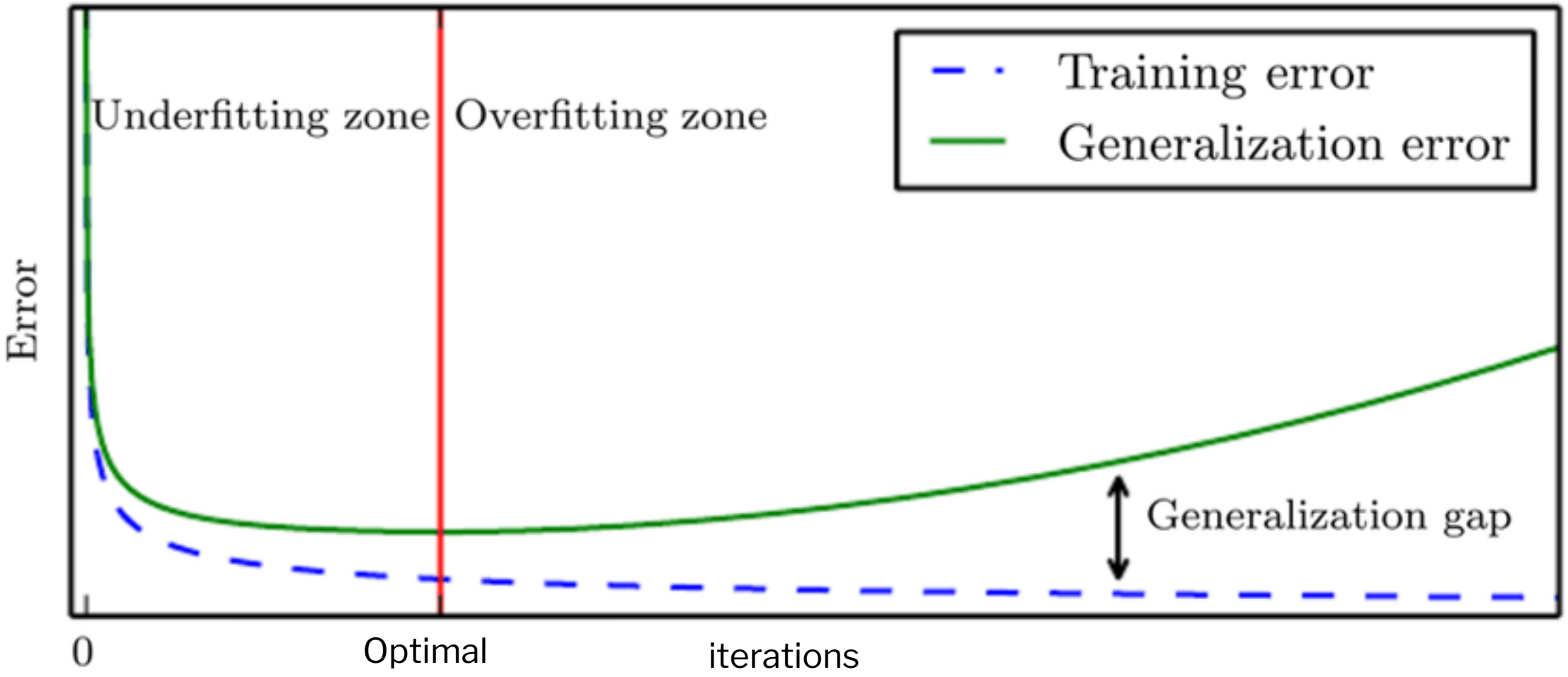




An optimal model minimizes total error, balancing bias and variance.

BYU

GENERALIZATION GAP



Supervised Learning:

BYU

K-NEAREST NEIGHBORS (KNN)

Main Idea: When given a new data point, KNN looks at the *k* closest data points in the training set (neighbors) and makes predictions based on their labels.

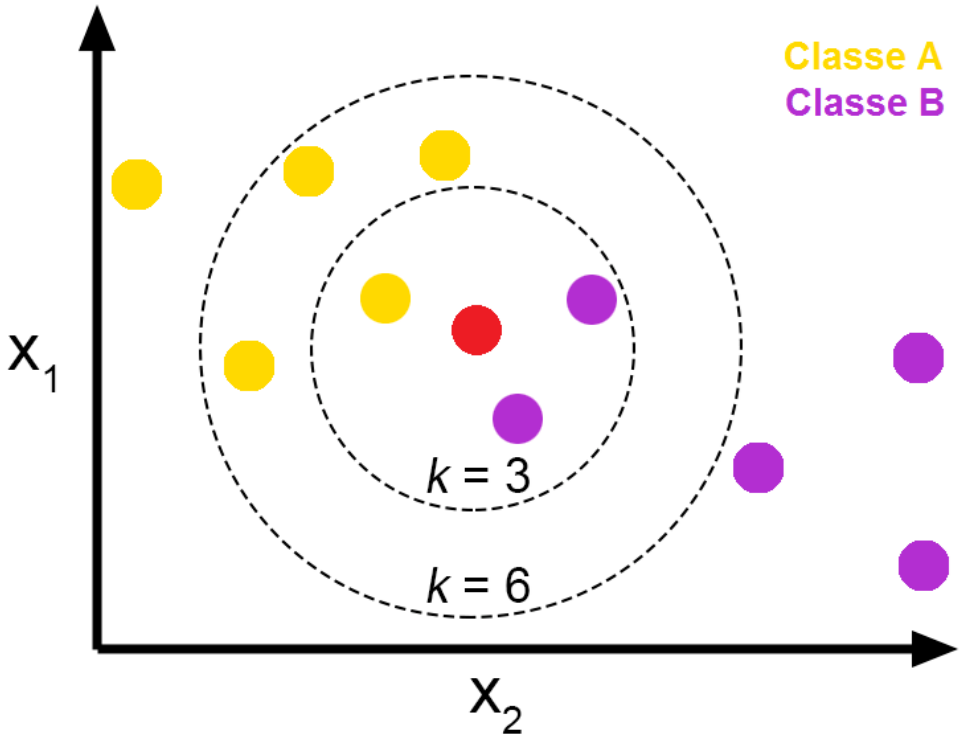
- For **classification**, the majority label among the neighbors is chosen.
- For **regression**, the average of the neighbors' values is used.

Steps:

1. Calculate the distance between the new point and all training points.
2. Sort the distances and find the k nearest neighbors.
3. Predict the label based on the neighbors.

The **K-nearest neighbor (KNN)** algorithm is considered a **supervised learning** algorithm because it uses labeled training data to make predictions about new, unseen data points.

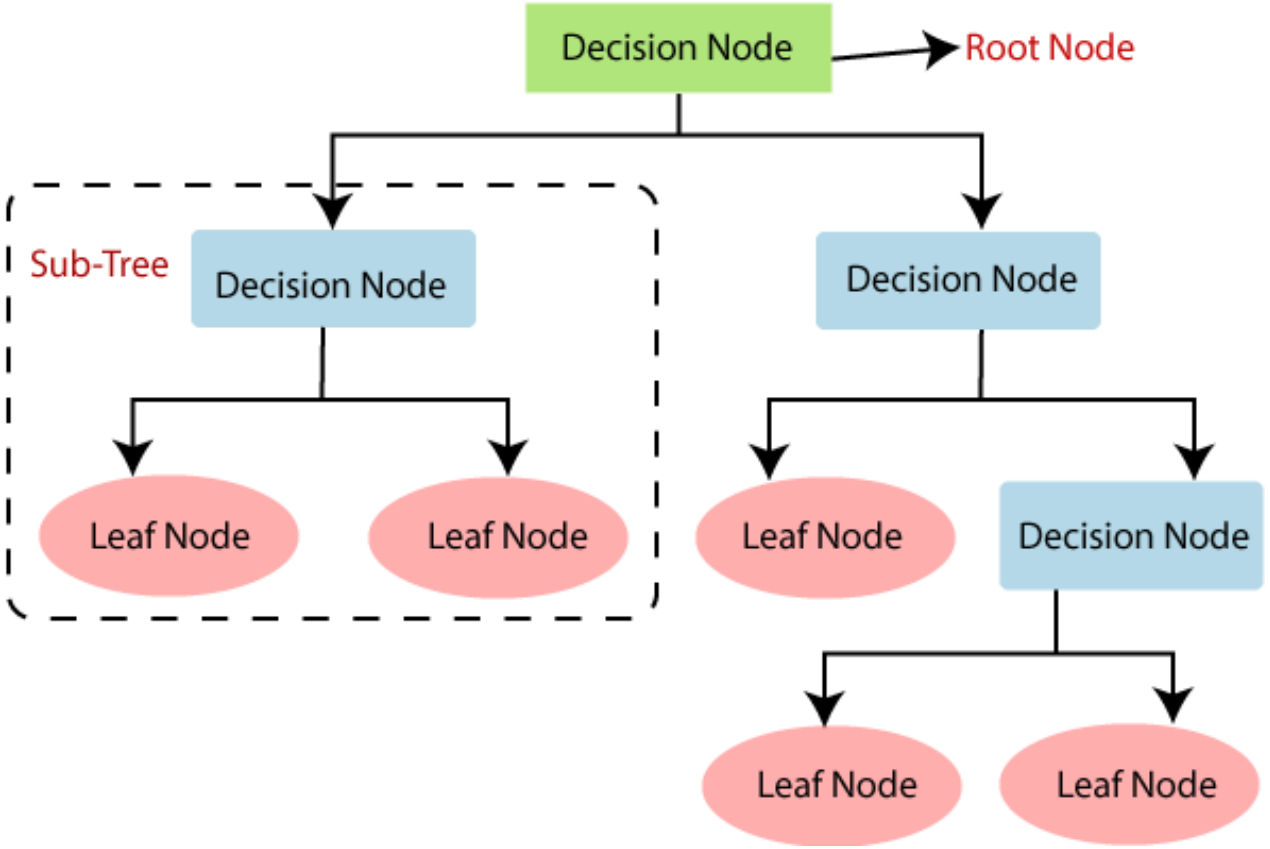
Instead of building a mathematical model or decision boundary during training, KNN stores the labeled training data.



BYU

DECISION TREES DEFINITIONS

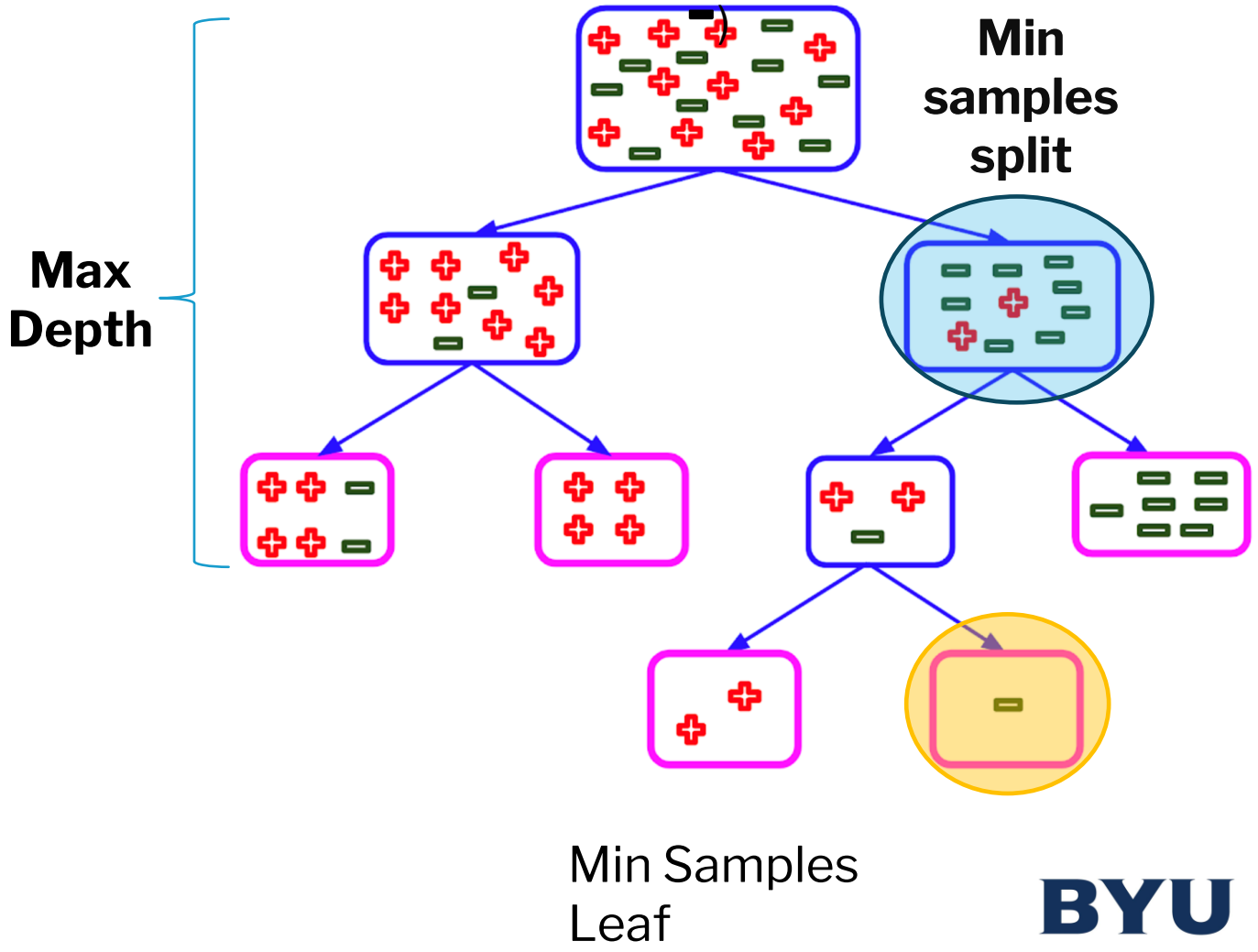
- **Root Node:** This attribute is used for dividing the data into two or more sets. Split is based on feature values.
- **Branch or Sub-Tree:** A part of the entire decision tree is called a branch or sub-tree.
- **Splitting**: Dividing a node into two or more sub-nodes based on if-else conditions.
- **Decision Node:** After splitting the sub-nodes into further sub-nodes, then it is called the decision node.
- **Leaf or Terminal Node:** This is the end of the decision tree where it cannot be split into further sub-nodes.
- **Pruning:** Removing a sub-node from the tree.



BYU

DECISION TREE HYPERPARAMETERS

- **max_depth**: The maximum depth of the tree. Limiting this helps prevent overfitting.
- **min_samples_split**: The minimum number of samples required to split an internal node.
- **min_samples_leaf**: The minimum number of samples required to be at a leaf node.
- **max_features**: The number of features to consider when looking for the best split.

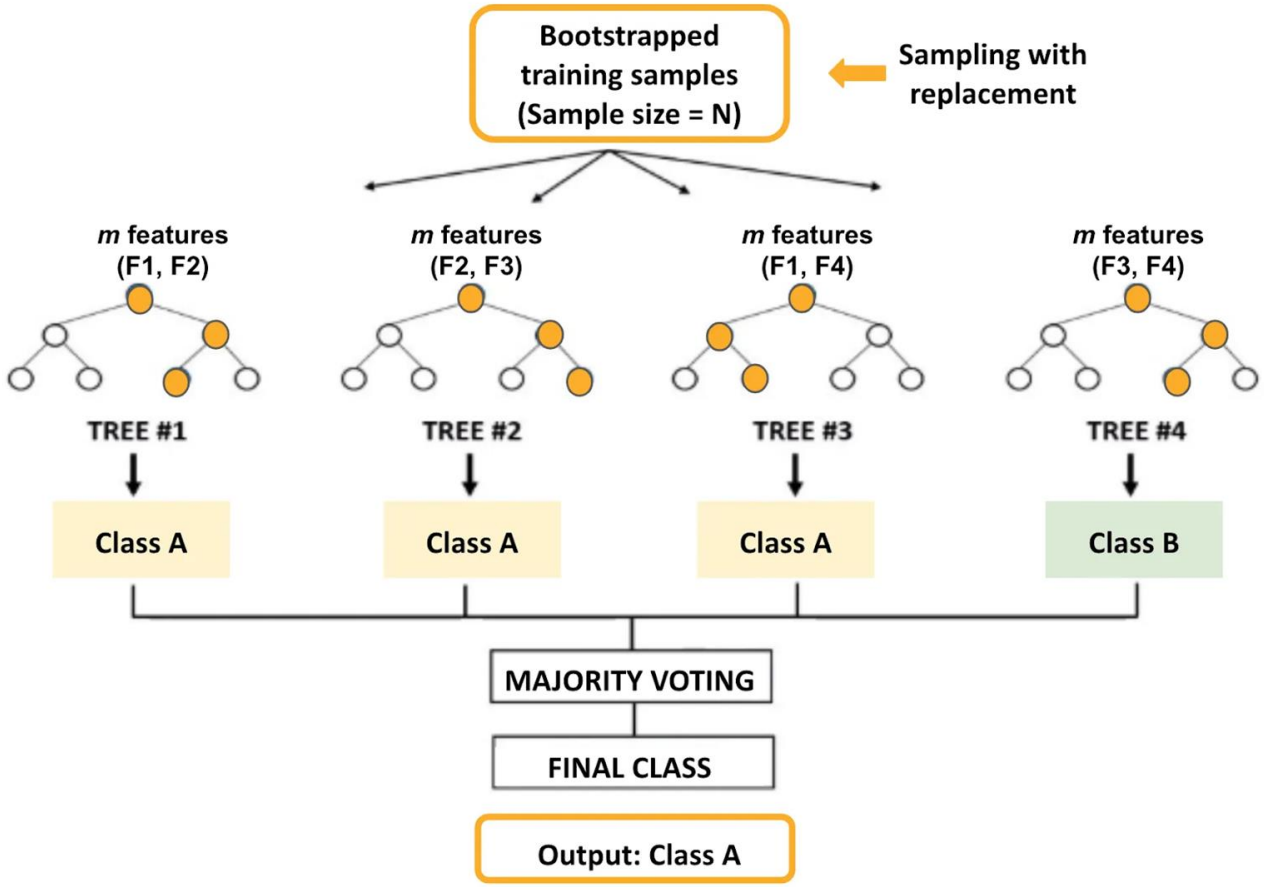


RANDOM FOREST

The **Random Forest** algorithm is an ensemble learning technique used for both **classification** and **regression**. It builds multiple decision trees during training and combines their outputs to improve performance and reduce overfitting.

Key Concepts

1. **Ensemble Learning**: Combines predictions from multiple models (decision trees) to produce a more robust and accurate prediction.
2. **Bagging (aka Bootstrap Aggregation)**:
   - Creates multiple subsets of the training data by sampling with replacement.
   - Each subset is used to train a decision tree.
3. **Random Feature Selection**:
   - At each split in a decision tree, a random subset of features is considered, making trees less correlated.
4. **Voting/Averaging**:
   - For classification, majority vote across trees is taken.
   - For regression, the average prediction of all trees is computed.



LOGISTIC REGRESSION

Problem Types:

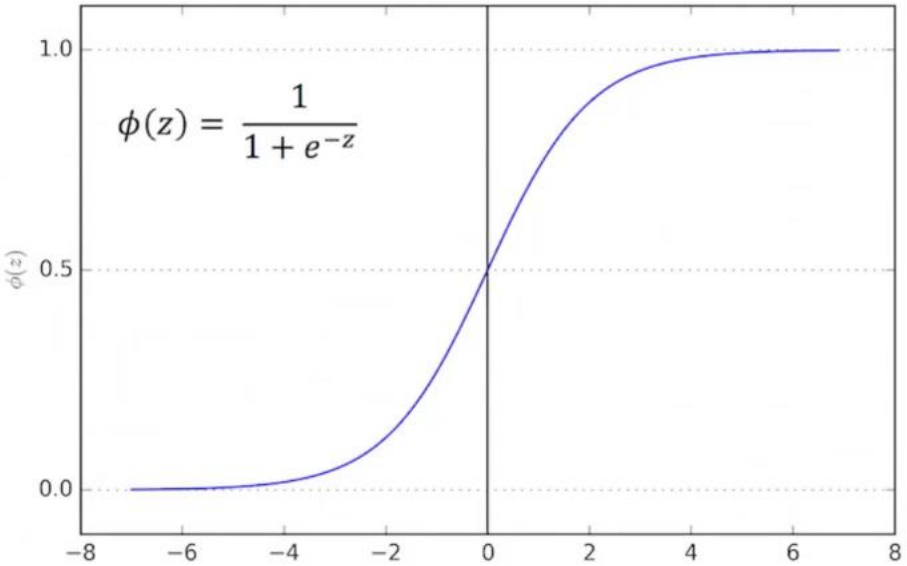
- Binary Classification: Predicts one of two possible outcomes (e.g., spam vs. non-spam emails).
- Multi-Class Classification: Extended by using techniques like One-vs-Rest (OvR) or Softmax for handling multiple classes.

Advantages:

- **Simplicity**: Easy to understand and implement.
- **Interpretability**: Weights provide insights into feature importance.
- **Probabilistic Outputs**: Predicts probabilities for confidence estimation.

Limitations:

- **Linearity**: Assumes a linear relationship between input features and log-odds of the outcome.
- **Outliers**: Sensitive to outliers; regularization helps mitigate this.
- **Multicollinearity**: Correlated features can reduce interpretability.



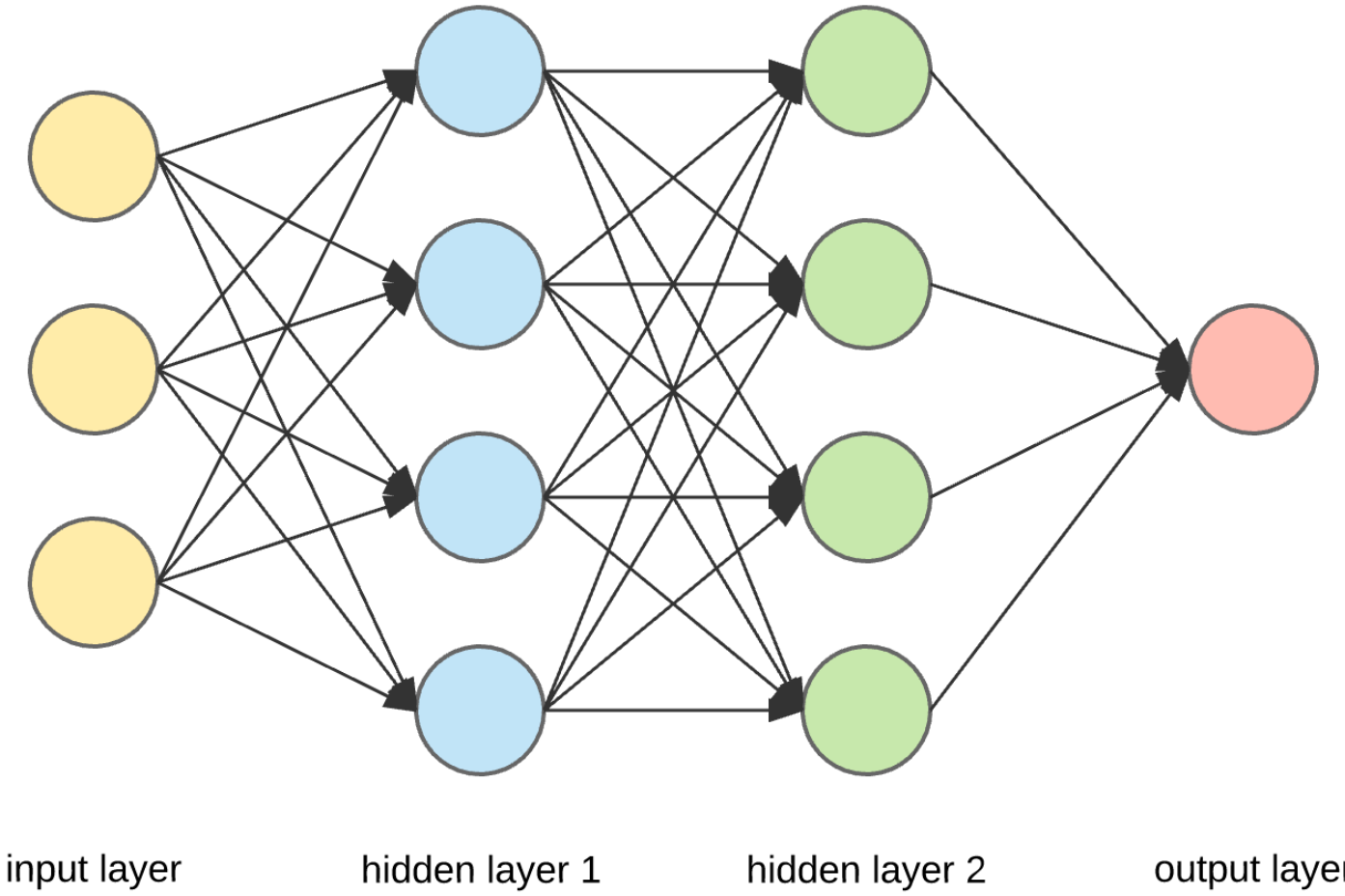
Not suitable for predicting continuous values…ie “regression” problems.

For a detailed tutorial: https://www.analyticsvidhya.com/blog/2021/08/conceptual-understanding-of-logistic-regression-for-data-science-beginners/

BYU

Deep Neural Network Architecture

- DNNs are structured into layers:
  - **Input Layer**: Takes the features of the data.
  - **Hidden Layers**: Learn complex patterns. (Two or more)
  - **Output Layer**: Produces the final predictions.
- Deep Neural Networks are technically defined as any network with **more than one hidden layer.**
- But in practice, they normally have many hidden layers.

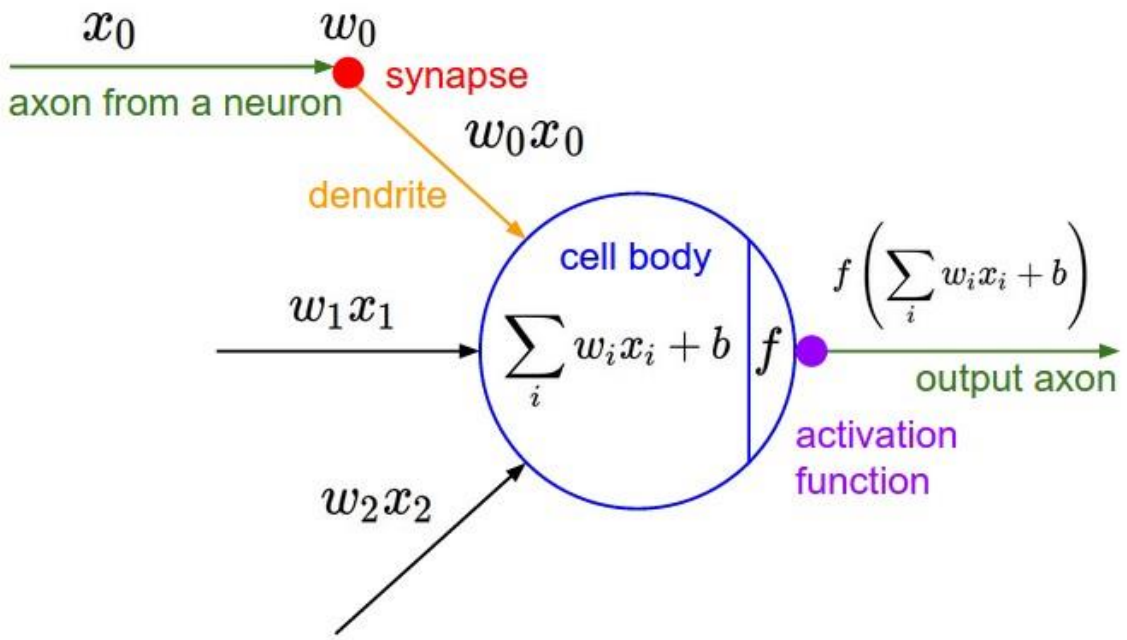


BYU

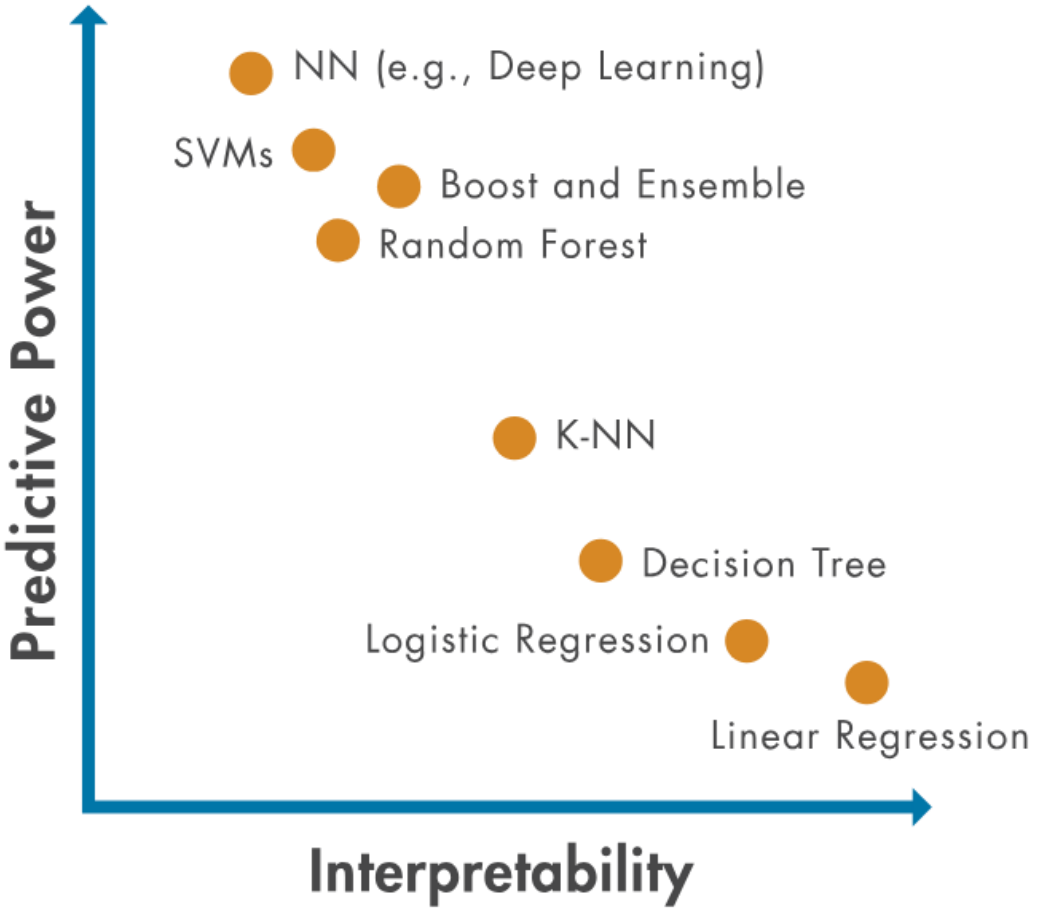
ARTIFICIAL NEURON STRUCTURE

- **Inputs**: The features or data fed into the neuron.
- **Weights**: Scalars that signify the importance of each input.
- **Bias**: A scalar added to the weighted sum to shift the activation function. Provides greater modeling flexibility.
- **Activation Function**. Determines the neuron's output based on the weighted sum.
- **Output**: The result of applying the activation function to the weighted sum.
- The formula for the output is:

$$y = f\left(\sum_{i}^{n} w_i x_i + b\right)$$



PREDICTIVE POWER VS INTERPRETABILITY



BYU