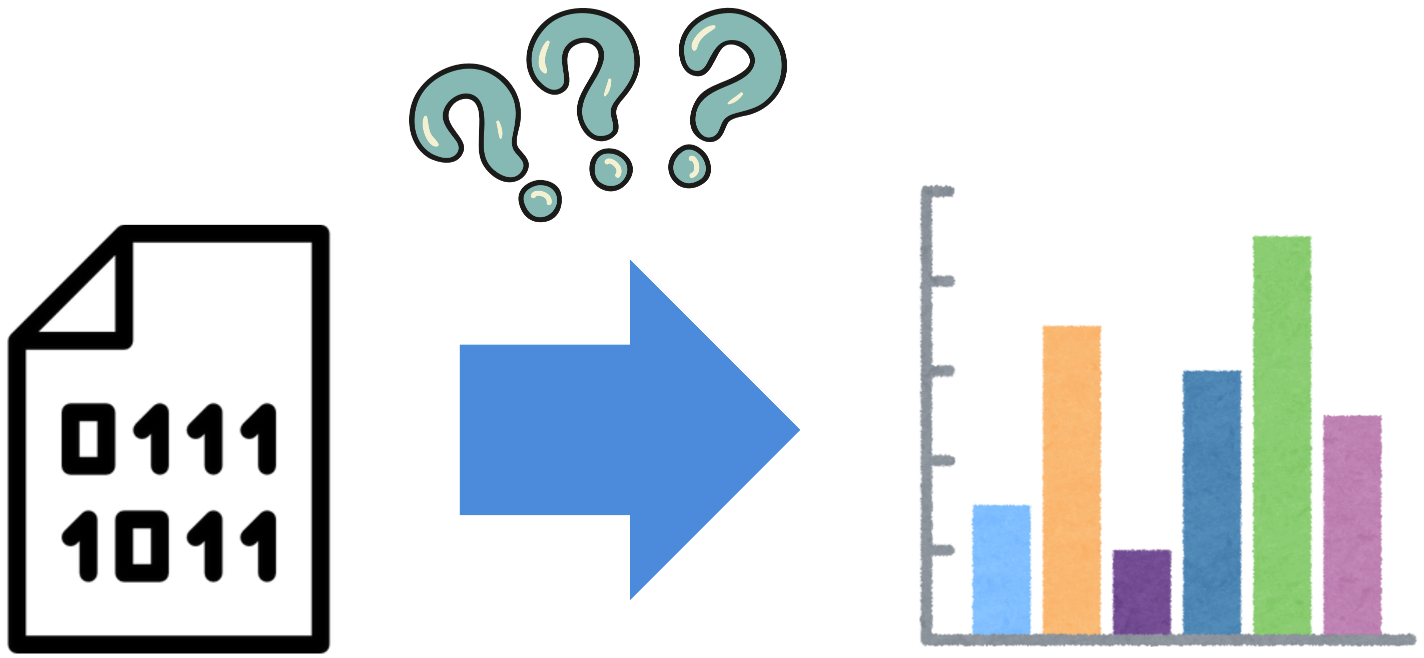
Model Building & Evaluation

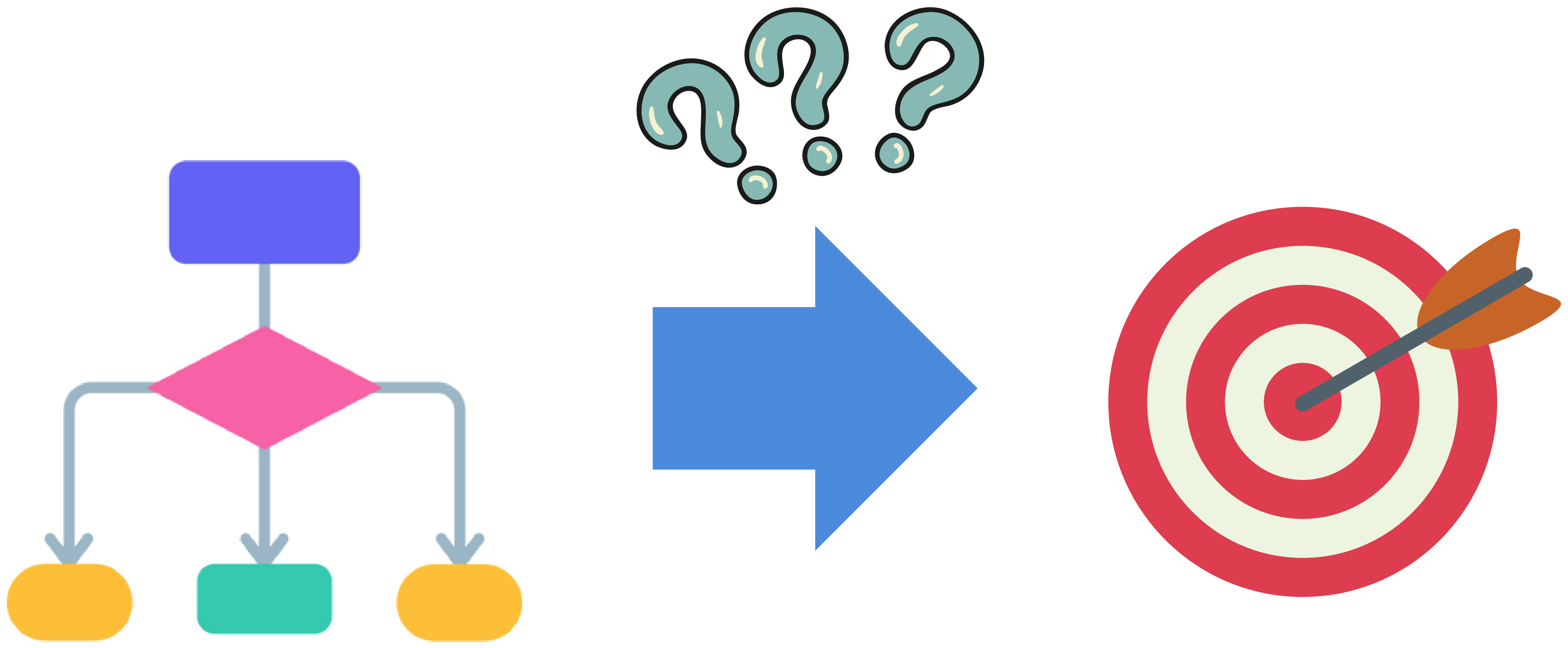
DSCUBED
UNIVERSITY OF MELBOURNE

Motivation: Why Build Models?

Why Build Models?



Learn patterns from data



Make reliable predictions

Models (Regression)

What is Regression?

Supervised learning task → predict a numeric outcome from a set of ‘independent’ variables

Examples

predict house price

predict income

predict stock price

Common Regression Models

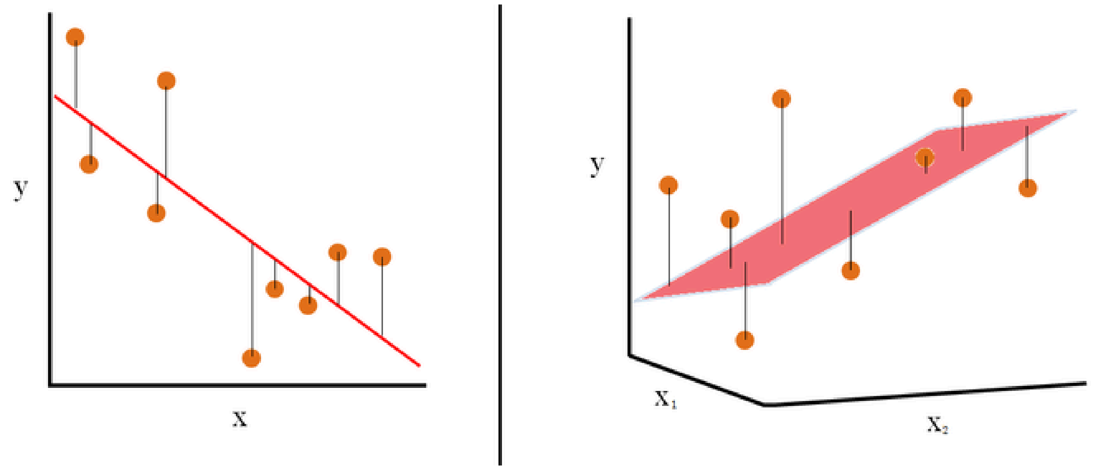
Linear Regression
Regularized Linear Regression
Decision Tree
Random Forest

Linear Regression

Weighted sum of features

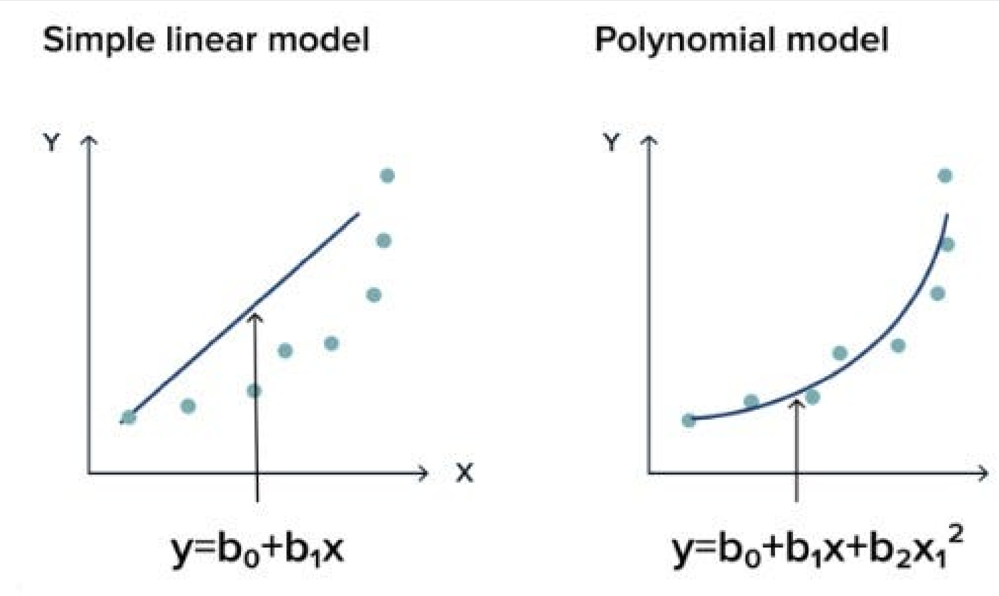
Goal: find optimal 'weights' to find the line (or plane for > 1 variables) that best fits the data

$$y \approx w_0 + w_1 x_1 + \cdots + w_k x_k$$




```python
from sklearn.linear_model import LinearRegression
from sklearn.metrics import mean_squared_error
from sklearn.metrics import mean_absolute_error

# fit on training data
model = LinearRegression()
model.fit(X_train, y_train)

# Predict on test data
y_pred = model.predict(X_test)

# Evaluate performance
mse = mean_squared_error(y_test, y_pred)
mae = mean_absolute_error(y_test, y_pred)
```

🤔 Want more? *Polynomial Regression*



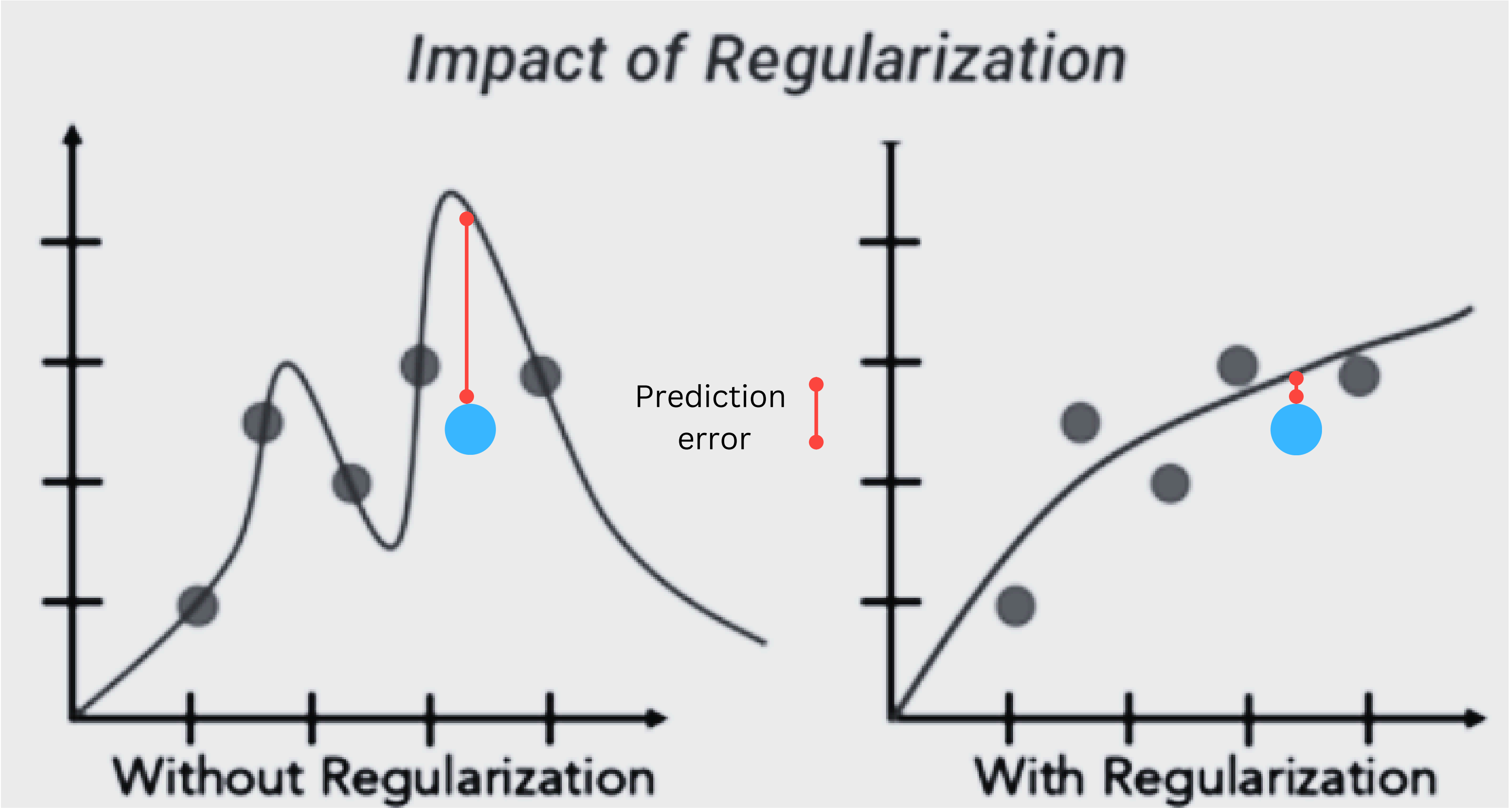

```python
from sklearn.preprocessing import PolynomialFeatures
from sklearn.pipeline import make_pipeline

# Create a pipeline: PolynomialFeatures → LinearRegression
degree = 2
model = make_pipeline(PolynomialFeatures(degree), LinearRegression())

# Fit the model
model.fit(X_train, y_train)

# then predict on test data and evaluate performance as usual.
```

Regularized Linear Regression

(Constrained) Weighted sum of features

Goal: find 'weights' to find the best fitting line/plane, **but we prefer smaller, more stable weights**

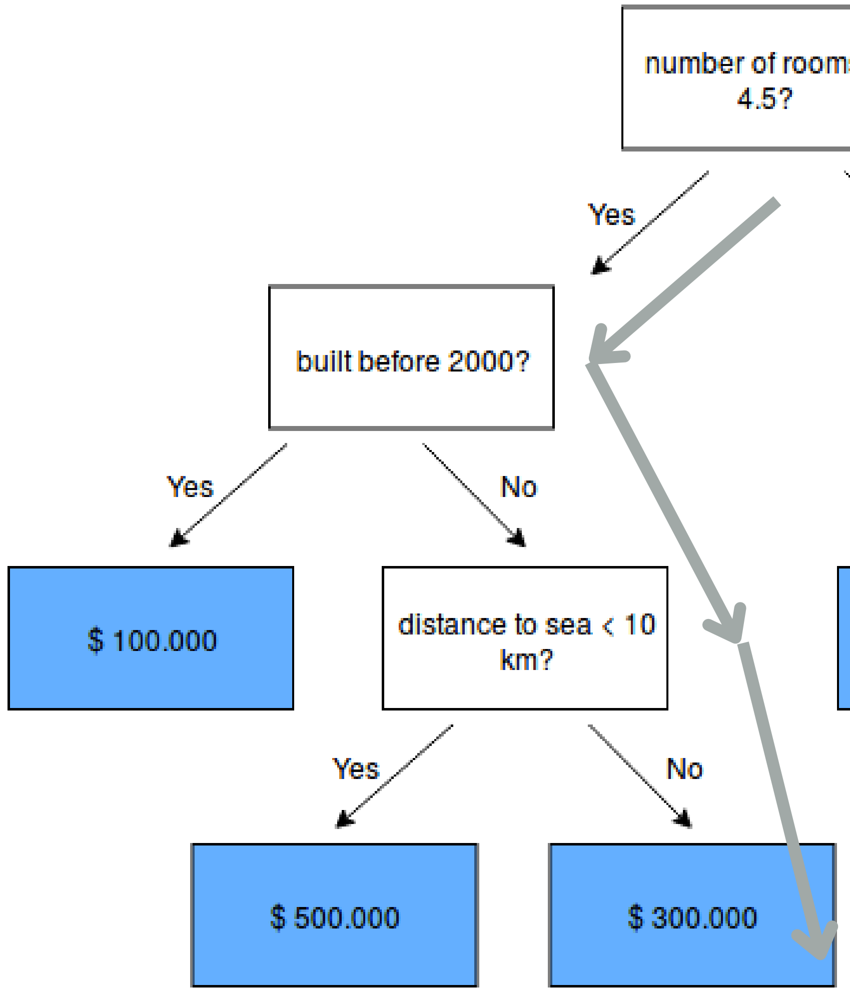
But why regularization?



What if you have a test point ?

Decision Tree Regressor

Prediction by split and conquer




```python
import numpy as np
from sklearn.tree import DecisionTreeRegressor
from sklearn.model_selection import train_test_split
from sklearn.metrics import mean_squared_error


X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.3, random_state=42)
regressor = DecisionTreeRegressor(max_depth=3, random_state=42)
y_pred = regressor.predict(X_test)

mse = mean_squared_error(y_test, y_pred)
print(f"Mean Squared Error: {mse:.4f}")
print(f"Root Mean Squared Error: {np.sqrt(mse):.4f}")
```

Maximum *depth* of tree is at most 3.

Very important *hyperparameter*.

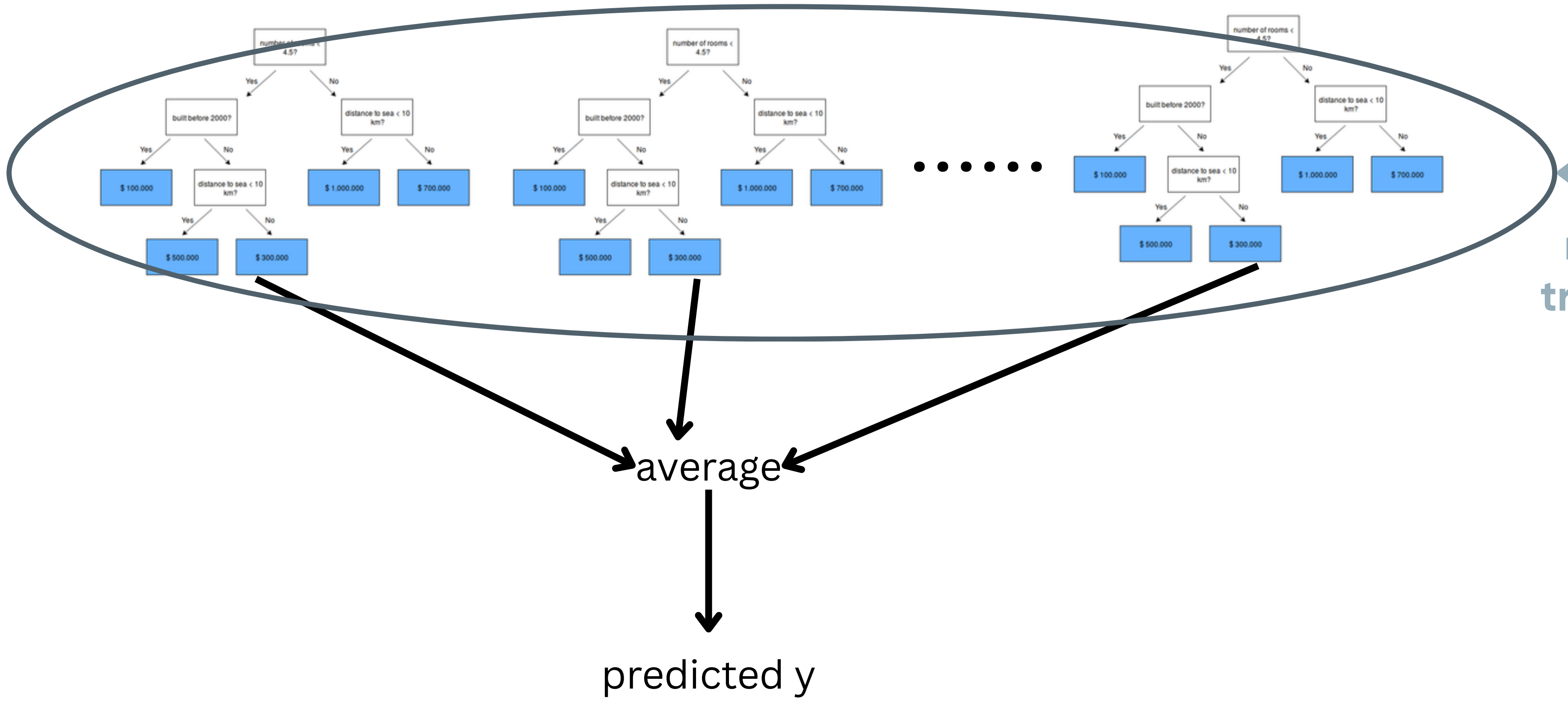
- Very deep tree → more flexible, but prone to *overfitting* (stay tuned: model evaluation)
- Very shallow tree → too 'dumb', miss important patterns

Mathematically,

1. divide the feature space into rectangles, and
2. predict the average of the y-values for any x falling inside each rectangle

Random Forest Regressor

Many trees, one forest, smoother predictions



n_estimators: How many trees do we grow? (default is 100)

```python
from sklearn.ensemble import RandomForestRegressor
from sklearn.metrics import mean_squared_error
import numpy as np

# 1. define the model (no hyperparameter tuning)
model = RandomForestRegressor(n_estimators=100,
                              max_features=None,
                              random_state=42)

# 2. fit on training data
model.fit(X_train, y_train)

# 3. predict
y_pred = model.predict(X_test)

# 4. get eval metrics
mse = mean_squared_error(y_test, y_pred)
rmse = np.sqrt(mse)

print(f"MSE: {mse:.4f}")
print(f"RMSE: {rmse:.4f}")
```

(ℹ️) How is the random forest made?

1. **Bootstrapping (sampling with replacement) the training set to obtain 'multiple training datasets'**
2. **Each tree only considers a random subset of features.**

- **Implication of n_estimators**
- **How about the depth of each tree?**

Models (Classification)

What is Classification?

Supervised learning task → predict a category label

Examples

spam vs not spam

disease A vs disease B

cat, dog, horse

Common Classification Models

Logistic Regression
Decision Tree
Random Forest
K-Nearest Neighbors

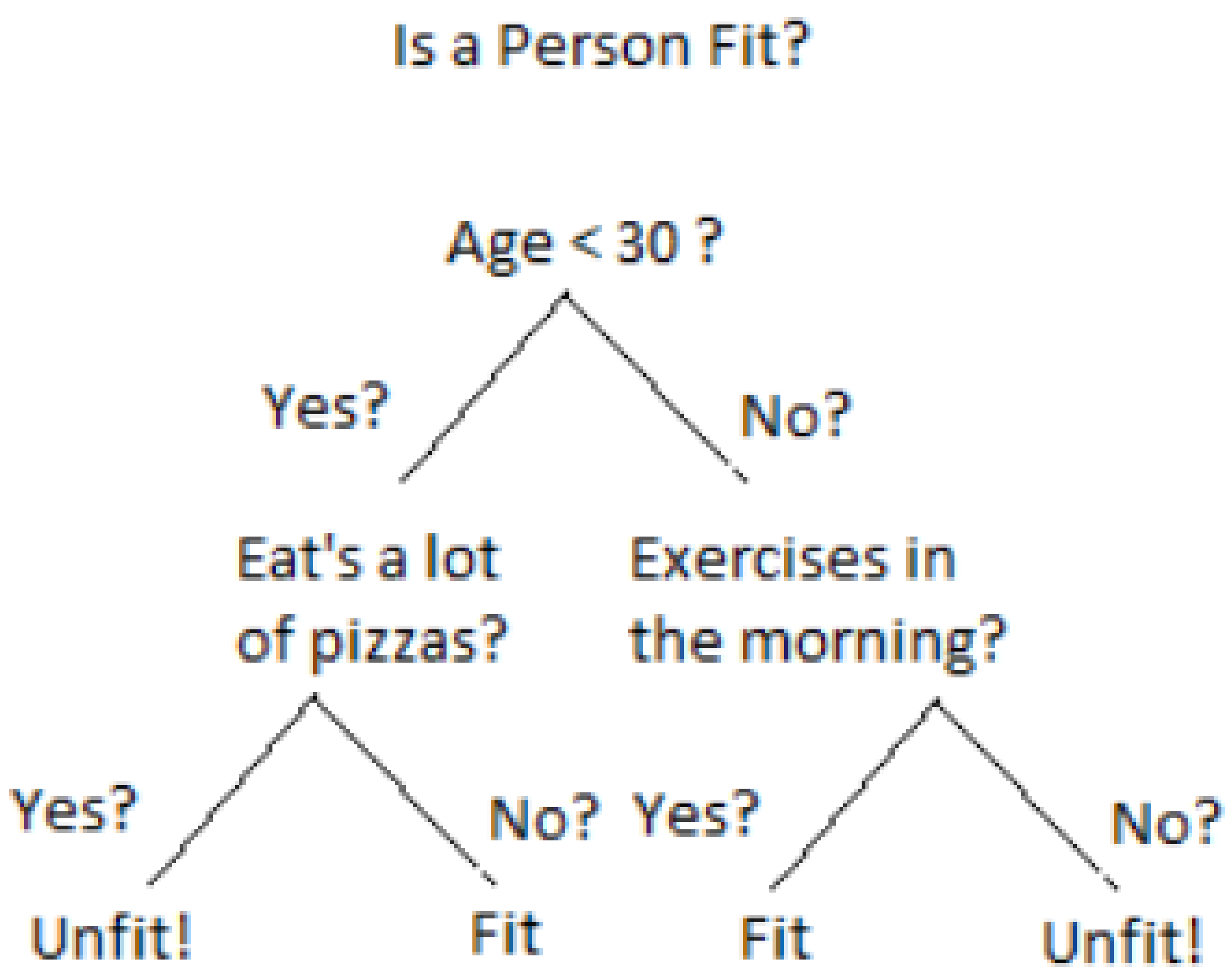
Logistic Regression

- Logistic Regression predicts the probability that something belongs to a class (e.g., 80% chance it's a cat).
- It uses a sigmoid curve to squeeze any number into 0-1 range.

Decision Tree

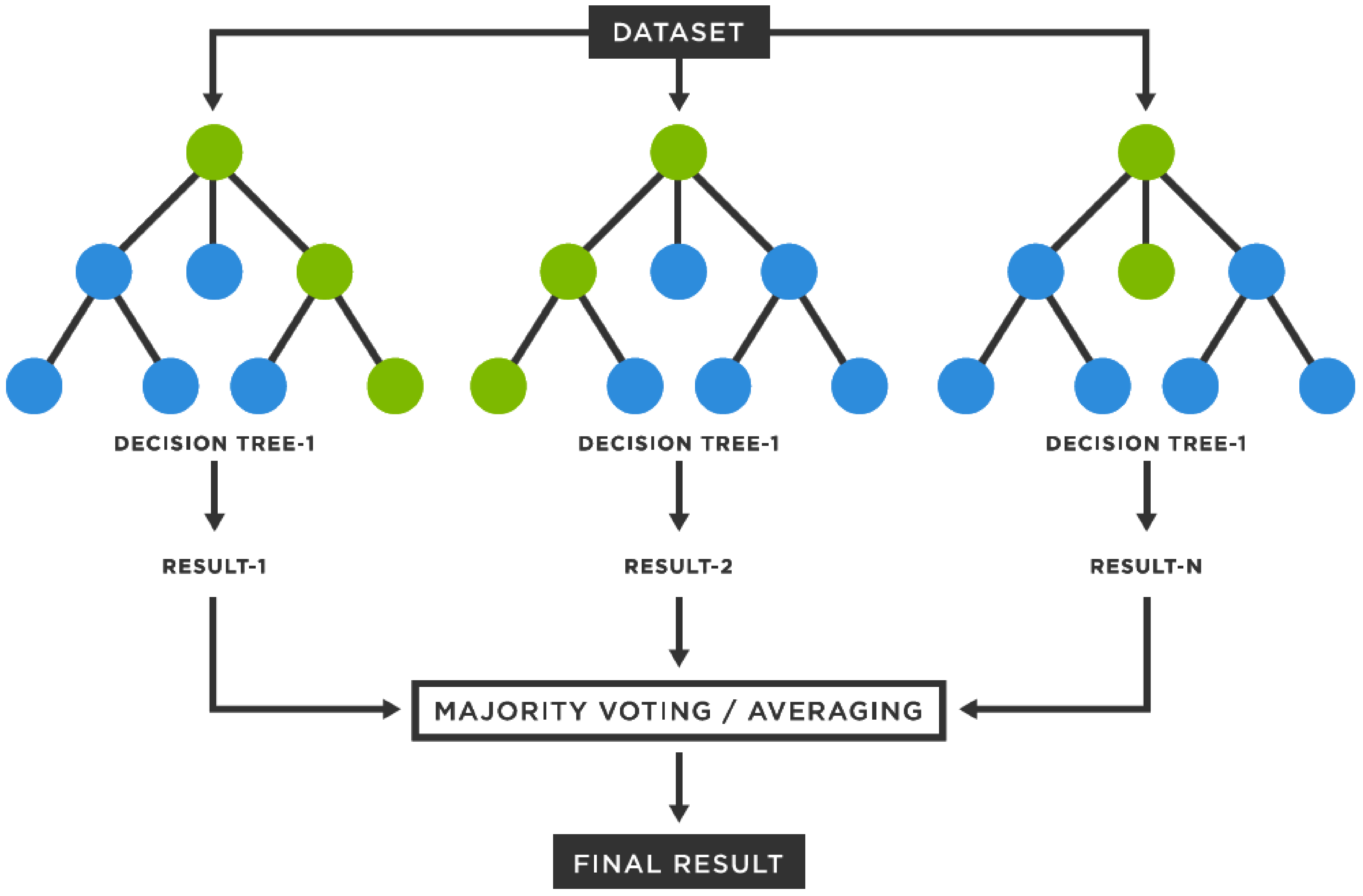
- Decision Trees split data step-by-step, asking 'yes' or 'no' questions about features, and predict the class based on majority vote at the end.

Analogy:
Game of 20 questions, Each node has a yes or no question, based on the answer, go left or right, until you reach a leaf → predict based on majority class.



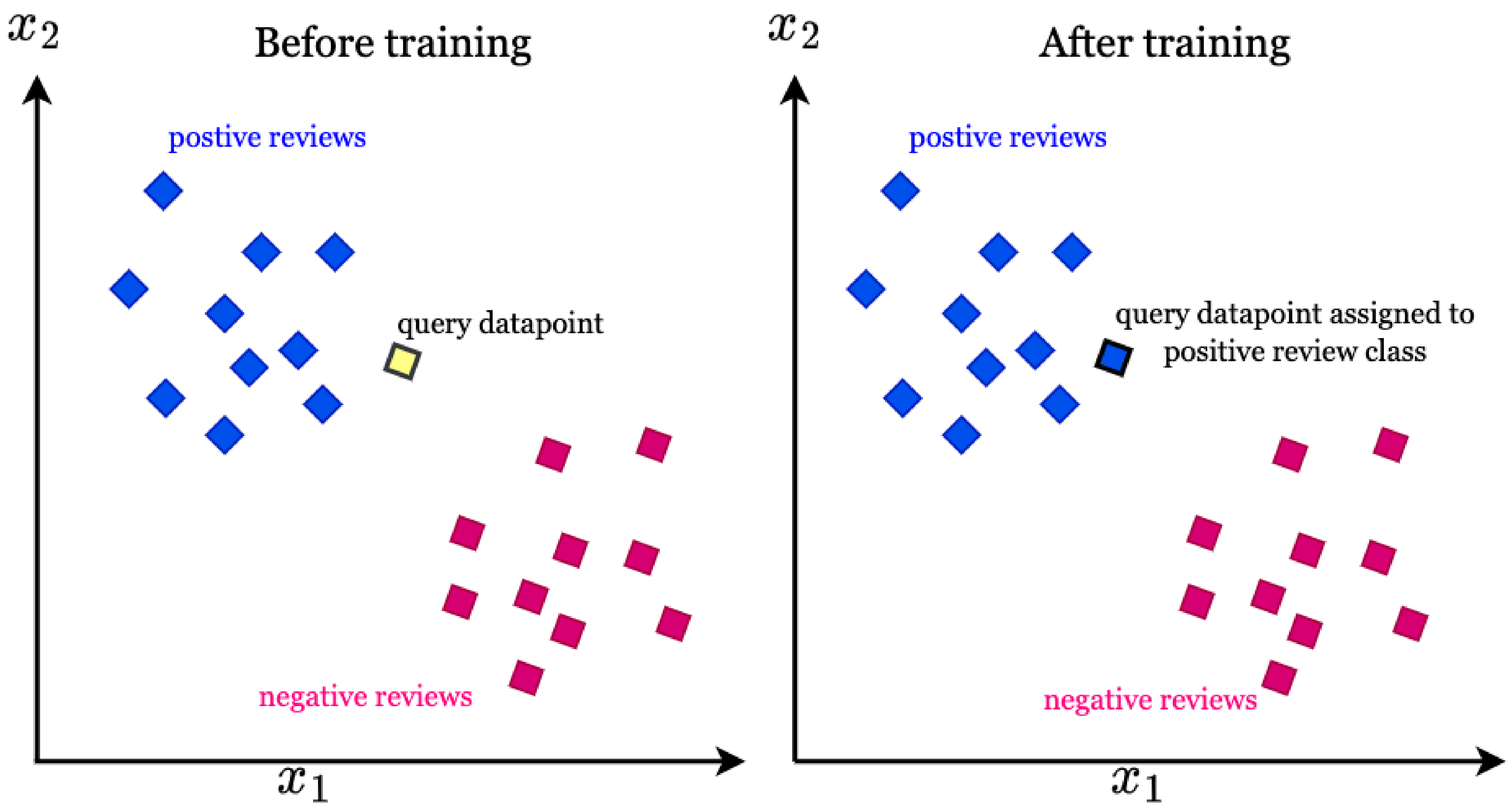
Random forest

- Random Forest = many decision trees.
- Each tree is trained differently (on different data/features).
- At the end, they vote for the final answer.



K-Nearest neighbors

- KNN doesn't learn a model.
- It stores the training data.
- When you want to predict, it looks at k closest points.



Metrics to evaluate

Accuracy - How often the model is correct overall.

Precision - Of the samples predicted as positive, how many were actually positive?

Recall - Of all the actual positive samples, how many did we correctly find

F1 Score - Harmonic mean of Precision and Recall.

Example

	Predicted Spam	**Predicted Not Spam**
Actually Spam	70	30
Actually Not Spam	20	80

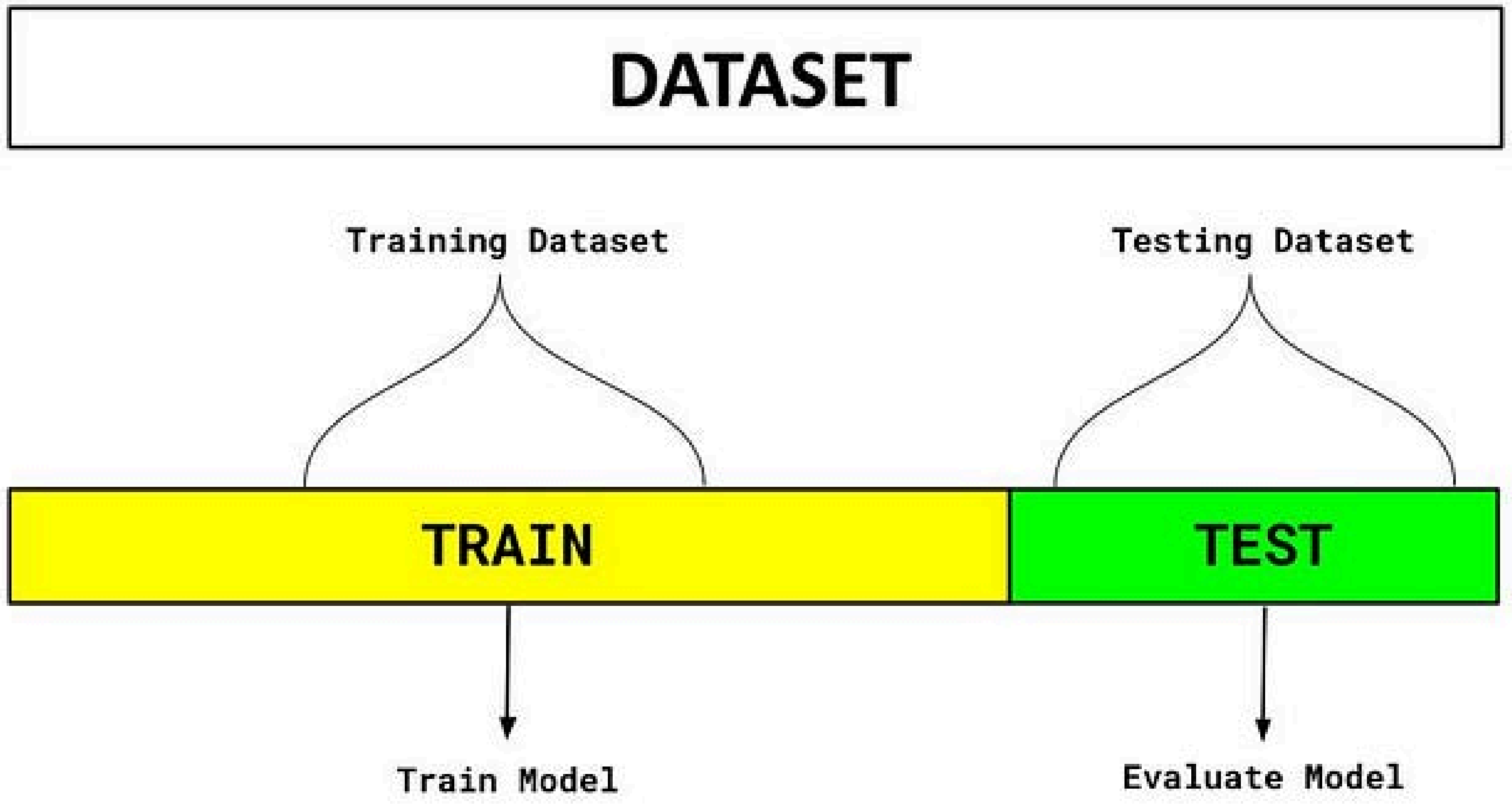
Model Evaluation

But Why Evaluate Models? Overfitting

- Model memorizes training data instead of learning patterns
- Performs well on training data but poorly on new data
- Fails to generalize to unseen examples

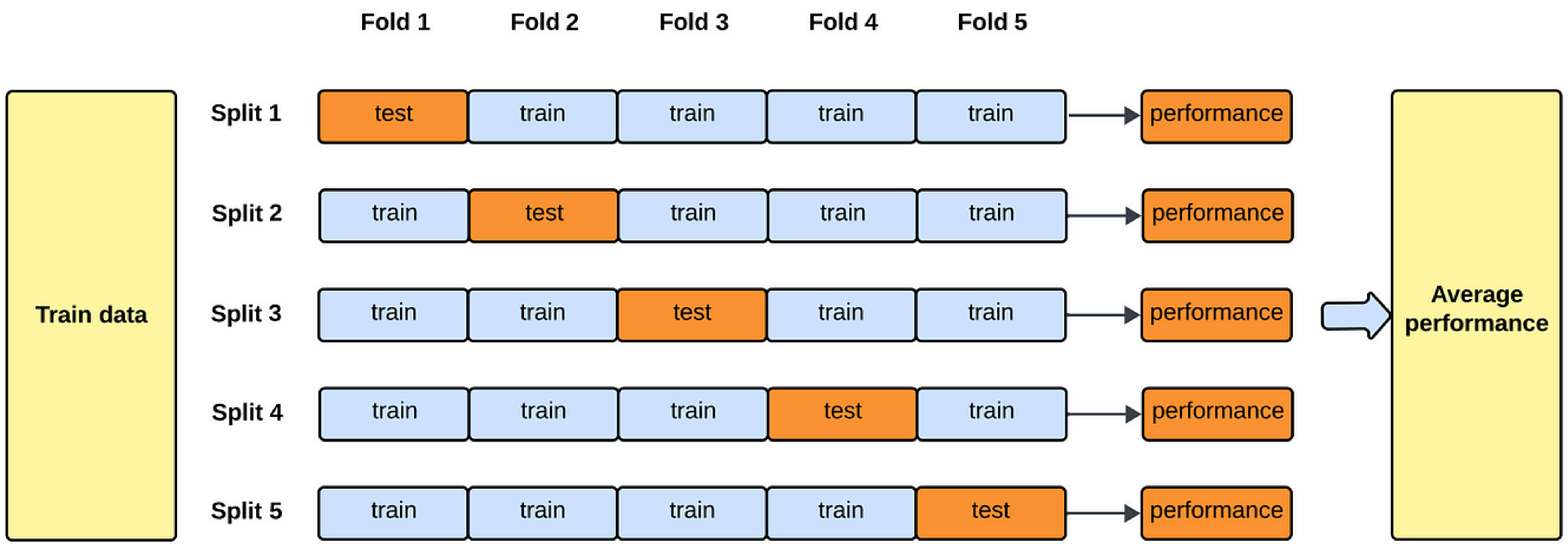
Train-Holdout Split

- Train on one part of the data
- Hold out another part to evaluate performance
- Test set should not be touched during training



K-fold Cross Validation

- Split into k parts (folds)
- Train on (k−1) folds, test on 1 fold
- Repeat k times and average results



Common Pitfall: Data Leakage

Information from outside training set leaks into model.
Model unknowingly cheats and gets fake good performance.
Leads to bad real-world results.

Why do we care?

- Leakage = False sense of security (fake good model).
- In real deployment, performance drops hard.
- Always split your data first, and only preprocess training set.

Preprocess BEFORE or AFTER splitting?

Split the data first into Train and Test sets.
Fit preprocessing (scaling, encoding) only on the Training set.
Apply the same transformation to the Test set.

Hands on Practise (Collab Notebook)

https://shorturl.at/UJFRN

Q&A

Thank You!!