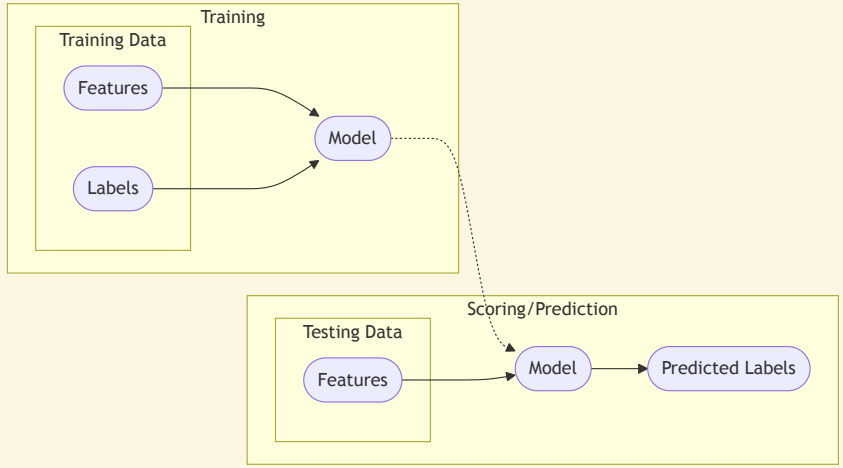
Supervised Machine Learning

Supervised Learning

Your "training" dataset is composed of examples of labeled examples:

- X - features
- y - labels

A supervised learning model learns to predict the label y from the features X.

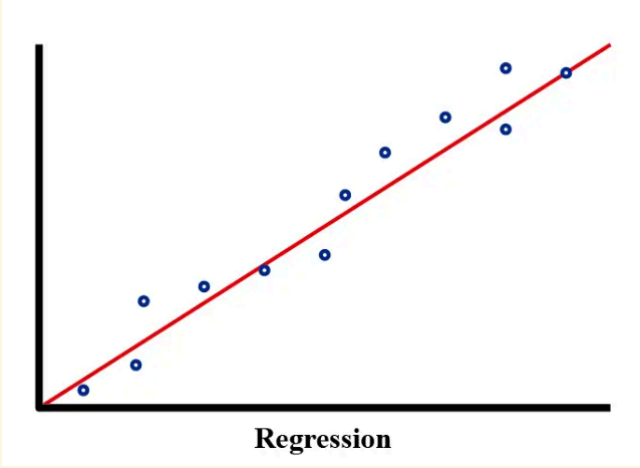


Supervised Learning

Regression

In regression, the label is a **continuous numerical value**.

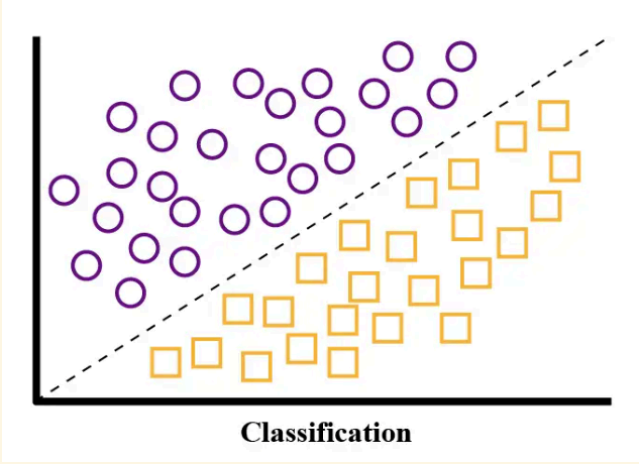
We approximate or predict a target value (like housing prices, stock prices, etc.).



Classification

In classification, the label is chosen from a **finite** set of **classes**.

We assign a category or class to an input (like spam detection, image recognition, etc.).



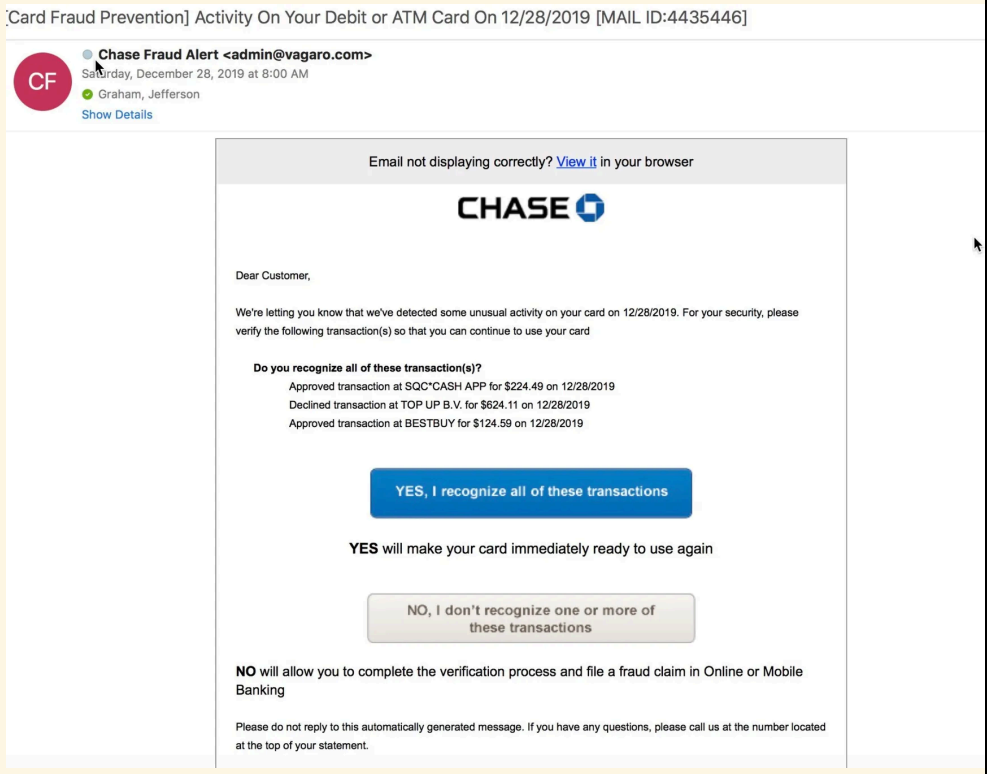
Supervised Learning Examples

Credit Card Fraud Detection

- **Features:** Vendor, location, time, distance from last transaction
- **Labels:** Chargebacks on previous transactions

Question: What kind of supervised learning?

1. Regression
2. Binary Classification
3. Multiclass Classification



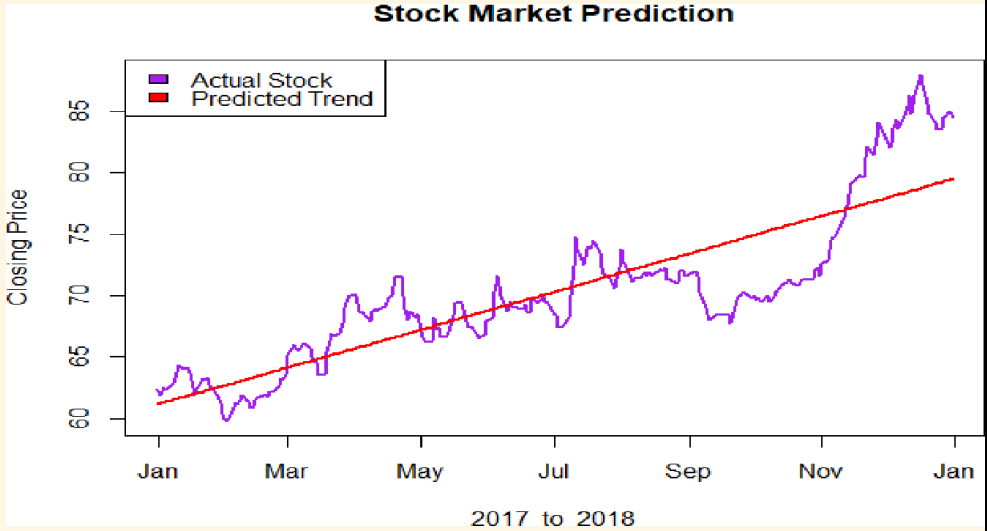
Supervised Learning Examples

Stock Market Prediction

- **Features:** Stock price from Feb 1st to March 1st
- **Labels:** Stock price on March 7th.

Question: What kind of supervised learning?

1. Regression
2. Binary Classification
3. Multiclass Classification



Supervised Learning Examples

Classification

- **Email Spam Filters**
  - Features: Words, sender, links
  - Label: Spam or Not Spam
- **Face ID / Fingerprint Unlock**
  - Features: Facial/fingerprint data
  - Label: You or Not You
- **Letter Recognition**
  - Features: Pixel values of images
  - Label: A-Z, 0-9, etc.

Regression

- **Weather Forecasting**
  - Features: Pressure, humidity, wind
  - Label: Temperature/rainfall amount
- **Uber/Lyft Pricing**
  - Features: Distance, time, demand
  - Label: Trip cost
- **YouTube View Count Predictions**
  - Features: Title, thumbnail, creator stats
  - Label: Expected views

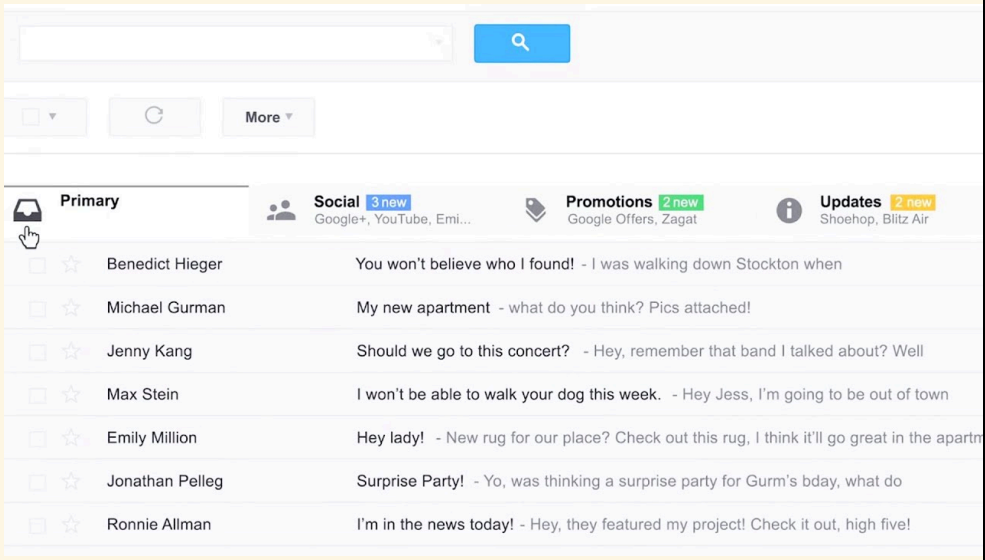
Supervised Learning Examples

Gmail Inbox Categories

- **Features:** Terms in the subject, body, origin email address
- **Labels:** "Primary", "Social", "Promotions", "Updates", "Forums"

Question: What kind of supervised learning?

1. Regression
2. Binary Classification
3. Multiclass Classification



Supervised Regression

What is Linear Regression?

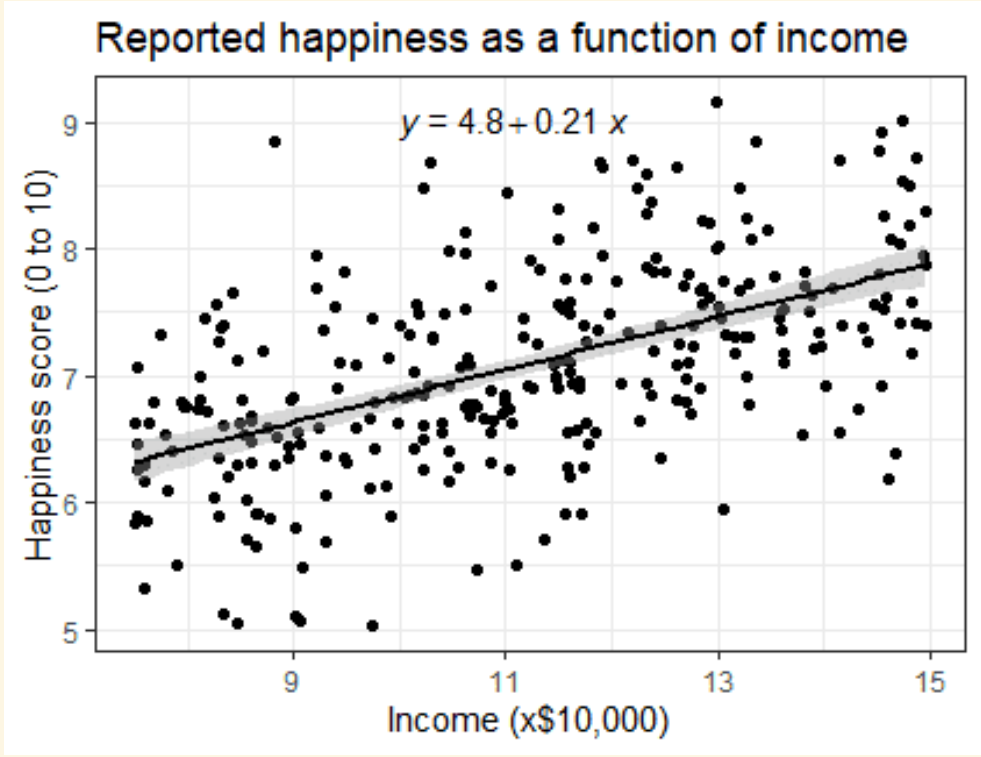
Linear regression is a **fundamental algorithm** in machine learning and can be thought of as simple supervised learning.

It models the relationship between a dependent variable y and one or more independent variables X by fitting a linear equation to the observed data.

For a simple model, we write it:

$$y = w_0 + w_1 x$$

- y: Dependent variable
- x: Independent variable
- w: A vector Coefficients



You've Seen This Before!

In high school, you learned about the equation of a line:

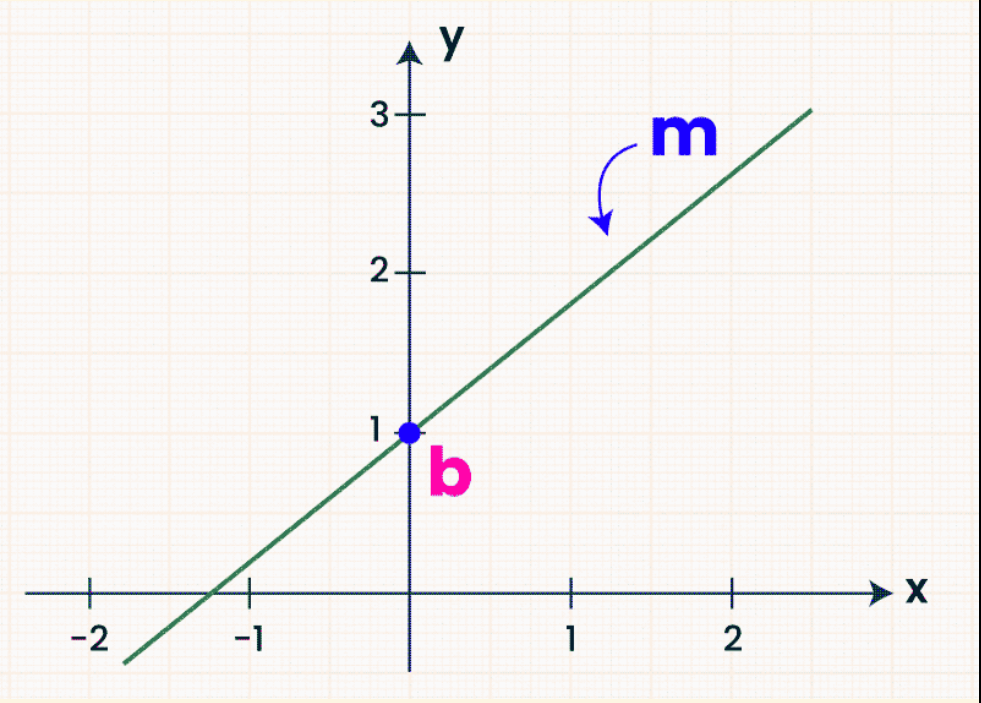
$$y = mx + b$$

The only difference is that in machine learning, we use w_0 and w_1 instead of m and b.

$$y = w_0 + w_1 x$$

That is...

$$\mathbf{w} = \begin{bmatrix} w_0 \\ w_1 \end{bmatrix} = \begin{bmatrix} m \\ b \end{bmatrix}$$



Let's Talk about that Notation

Math

$$y = w_0 + w_1 x$$

or

$$y = \mathbf{X} \cdot \mathbf{w}$$

Where

$$\mathbf{y} = \begin{bmatrix} y_1 \\ y_2 \\ \vdots \\ y_n \end{bmatrix}, \quad \mathbf{X} = \begin{bmatrix} 1 & x_1 \\ 1 & x_2 \\ \vdots & \vdots \\ 1 & x_n \end{bmatrix}, \quad \mathbf{w} = \begin{bmatrix} w_0 \\ w_1 \end{bmatrix}$$

Python (w/ Pandas)

```
y = X.dot(w)
```

Where:

```
y = pd.Series([
  y₁,
  y₂,
  ...,
  yₙ
])
X = pd.DataFrame([
    [1, x₁],
    [1, x₂],
    ...,
    [1, xₙ],
])
w = pd.Series([
  w₀,
  w₁
])
```

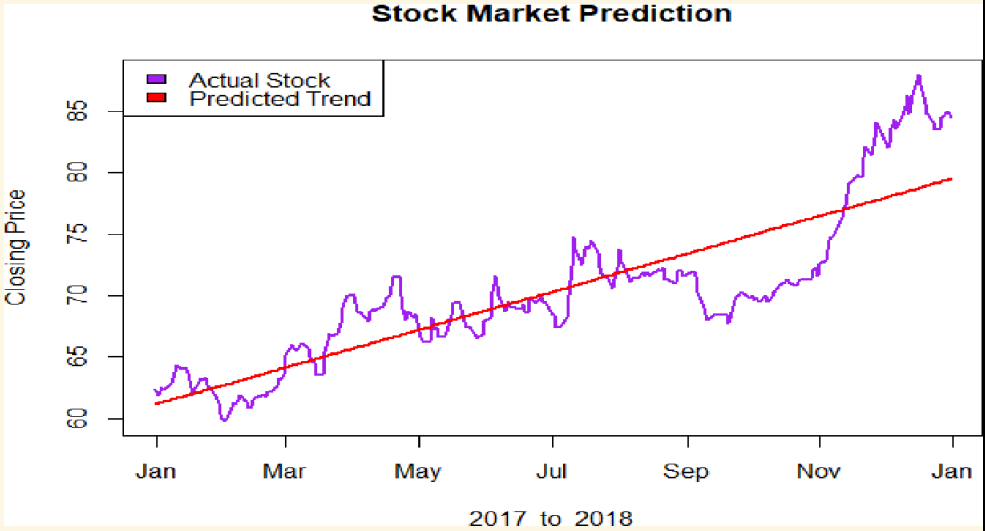
Linear Regression Example

Stock Market Prediction

- **Features:** Stock price from Feb 1st to March 1st
- **Labels:** Stock price on March 7th.

In the example above, the linear regression model would learn the relationship between the stock prices over time and use that to predict future prices.

- y: Closing Price
- x: Date
- w: The trained coefficients



Linear Doesn't Always Mean Line

We know that:

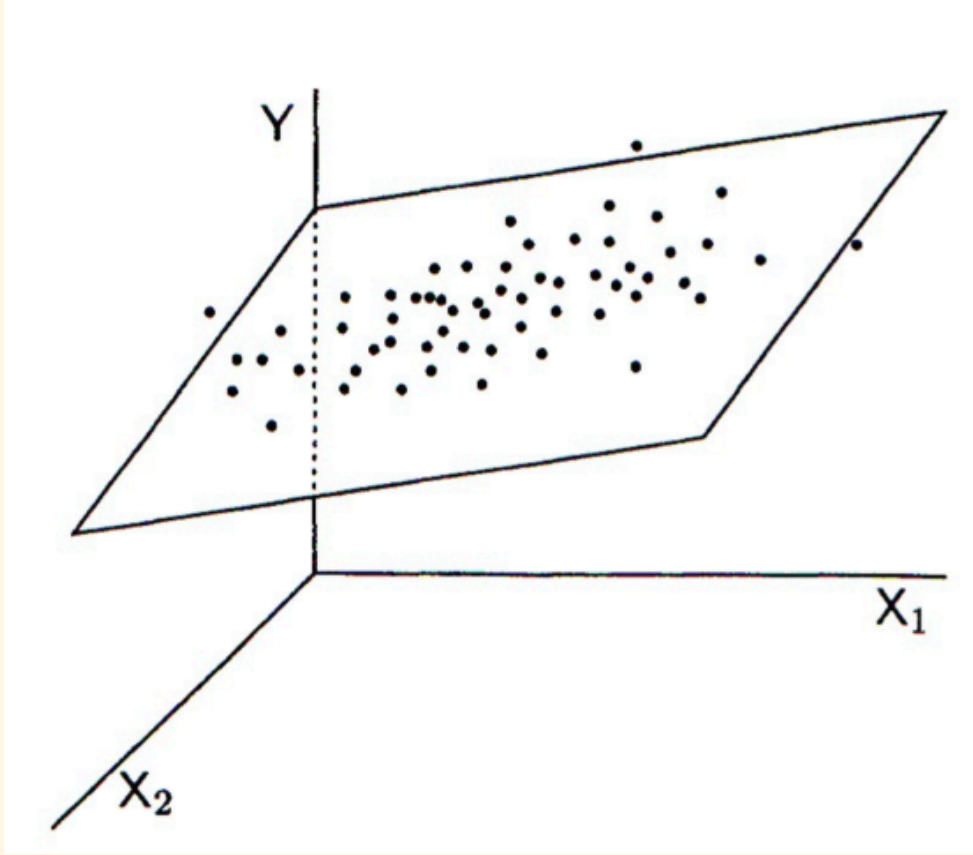
$$y = w_0 + w_1 x$$

But just as we think of w as a vector, we can think of y and x as vectors and matrices.

$$\mathbf{w} = \begin{bmatrix} w_0 \\ w_1 \end{bmatrix} \mathbf{y} = \begin{bmatrix} y_1 \\ y_2 \\ \vdots \\ y_n \end{bmatrix}, \quad \mathbf{X} = \begin{bmatrix} 1 & x_{11} \\ 1 & x_{21} \\ \vdots & \vdots \\ 1 & x_{n1} \end{bmatrix}$$

And now we can write the equation as:

$$\mathbf{y} = \mathbf{X} \cdot \mathbf{w}$$



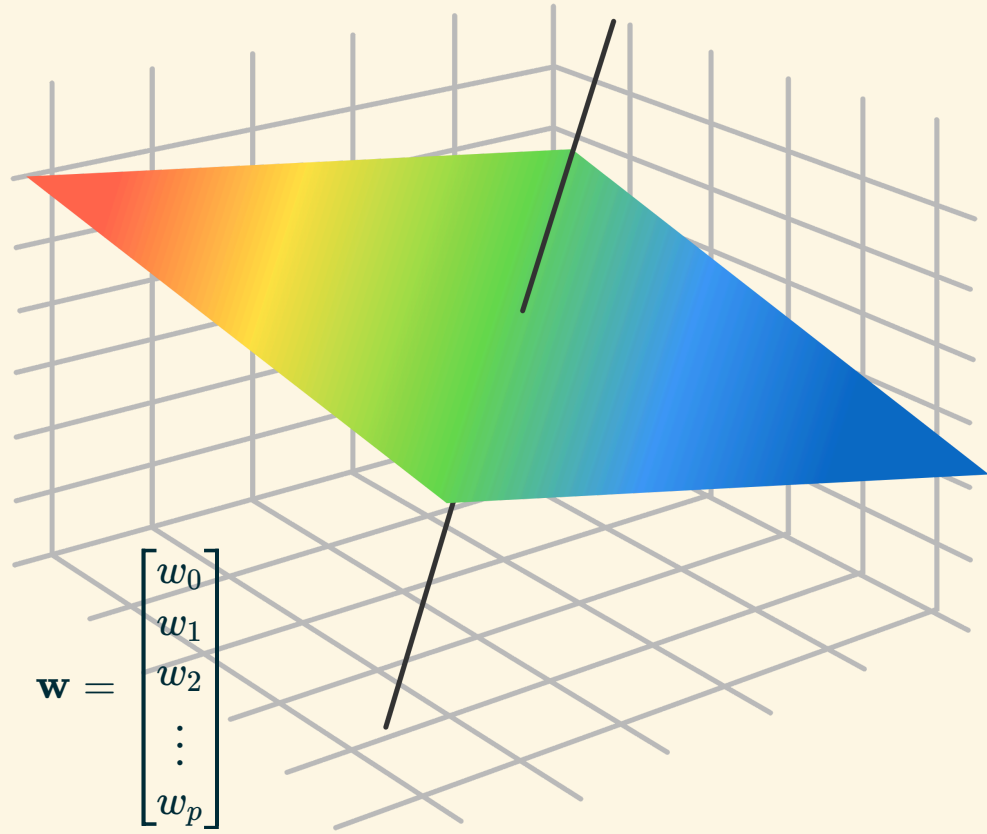
Multiple Linear Regression

- For multiple linear regression, the model includes multiple independent variables:

$$y = w_0 + w_1 x_1 + w_2 x_2 + \ldots + w_p x_p +$$

  - $x_1, x_2, \ldots, x_p$: Independent variables
  - $w_1, w_2, \ldots, w_p$: Coefficients

$$\mathbf{y} = \begin{bmatrix} y_1 \\ y_2 \\ \vdots \\ y_n \end{bmatrix}, \quad \mathbf{X} = \begin{bmatrix} 1 & x_{11} & x_{12} & \ldots & x_{1p} \\ 1 & x_{21} & x_{22} & \ldots & x_{2p} \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ 1 & x_{n1} & x_{n2} & \ldots & x_{np} \end{bmatrix}, \quad \mathbf{w} = \begin{bmatrix} w_0 \\ w_1 \\ w_2 \\ \vdots \\ w_p \end{bmatrix}$$



In Code

```
from sklearn.linear_model import LinearRegression

# Create a linear regression model
model = LinearRegression()

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

# Make predictions
y_pred = model.predict(X_test)
```

This is similar to saying:

$$\texttt{y\_pred} = \texttt{X\_pred} \cdot w$$

Visualizing Linear Regression

- The line of best fit minimizes the vertical distances (errors) between the observed points and the predicted line.
- The sum of these squared distances is what we aim to minimize using the loss function.

MSE Loss Function

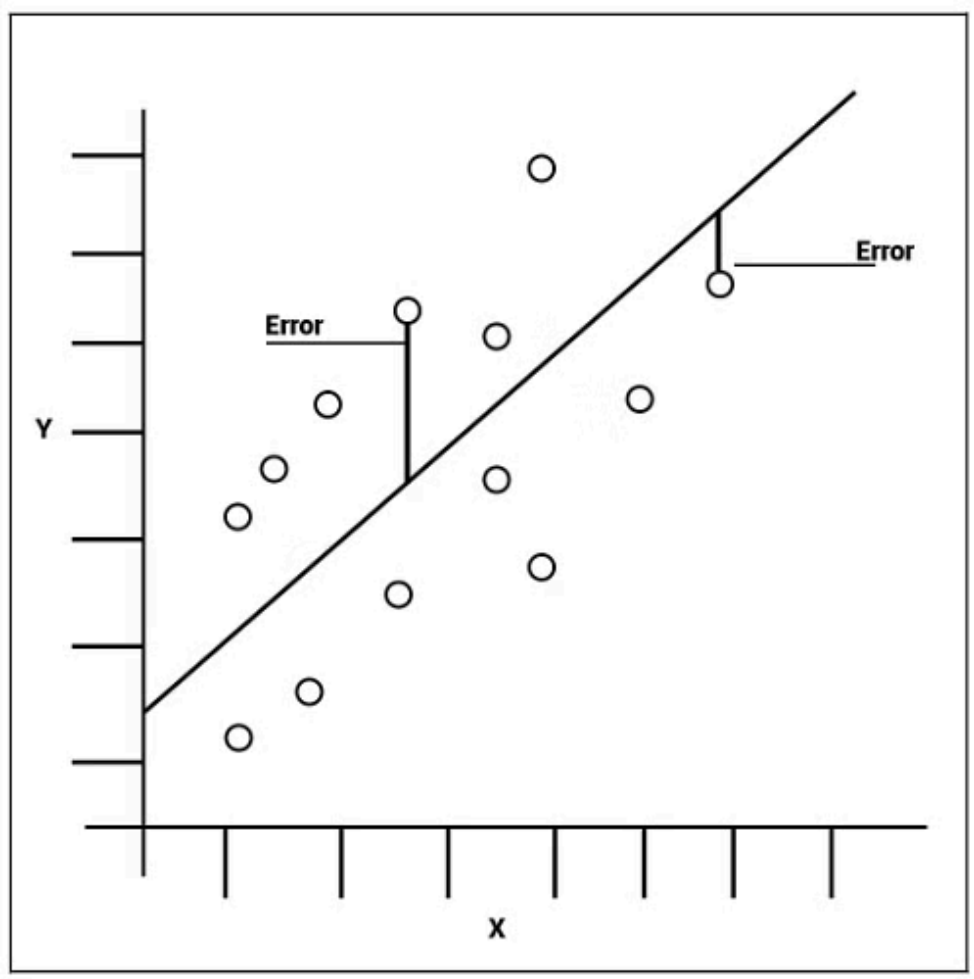
- The goal of linear regression is to find the values of w that minimize the difference between the observed and predicted values of y.

- We quantify this using a **loss function** called **Mean Squared Error (MSE)**, calculated as:

$$MSE = \frac{1}{N}\sum_{i=1}^{N}(y_i - \hat{y}_i)^2$$

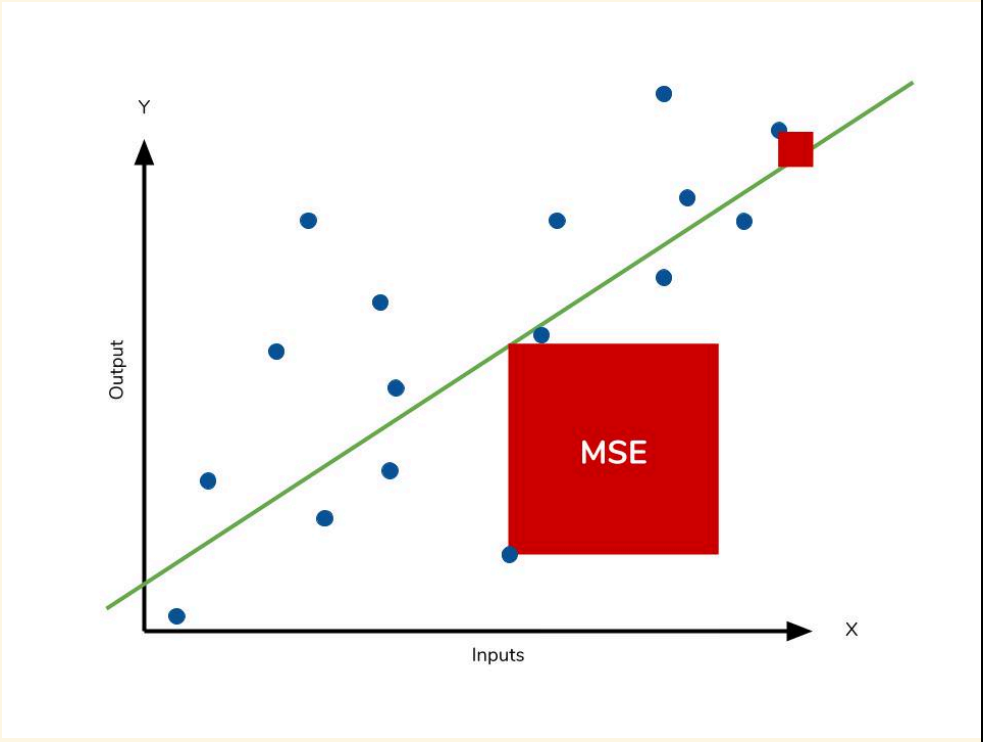
  - y_i: Actual value
  - $\hat{y}_i$: Predicted value
  - N: Number of observations

All models have a loss function, to "fit" a model is to minimize the loss function.



Why Squared Error Instead of Absolute Error?

- Mean Absolute Error (MAE) is an alternative to MSE.
- By squaring the errors, we ensure they are always positive.
- MSE is differentiable, which makes it easier to optimize using gradient-based methods.
- **Squaring the errors emphasizes larger discrepancies, making the model more sensitive to outliers.**



Fitting a Linear Regression

Normal Equation

For small to medium-sized datasets, linear regression can be solved using simple matrix math:

$$w = (X^T X)^{-1} X^T y$$

Where:

- X: Matrix of input features
- y: Vector of output values

For larger datasets (and for other algorithms we'll go over later), **Gradient Descent** is used.

For this class, we'll just use SKLearn.

In Python

```
import pandas as pd

def train_linear_regression(X_train, y_train):
    # Calculate means
    x_mean = x.mean(x_values)
    y_mean = y.mean(y_values)

    # Calculate slope (m)
    numerator = ((x - x_mean) * (y - y_mean)).sum()
    denominator = ((x - x_mean) ** 2).sum()
    m = numerator / denominator

    # Calculate intercept (b)
    b = y_mean - m * x_mean

    # w is the coefficients/weights vector
    w = pd.Series([b, m])

    return return w
```

What is Polynomial Regression?

Many real-world relationships are not purely linear; polynomial regression allows us to capture curvature in the data.

A basic polynomial relationship might be:

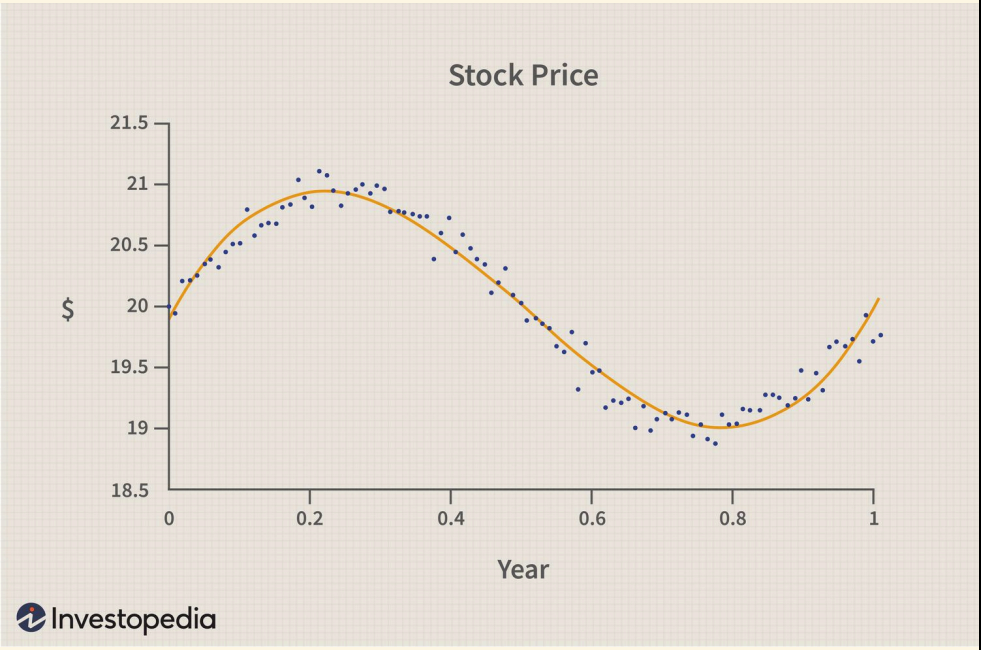
$$y = c_1 x + c_2 x^2 + \text{b}$$

We can rewrite this as:

$$y = w_0 + w_1 x + w_2 x^2$$

Notice that x^2 adds a non-linear effect.

But even though the relationship is polynomial in x, it remains linear in the coefficients w. **So we can solve it with the same techniques!**



From Multiple to Polynomial Regression

Polynomial regression is actually **multiple linear regression** using polynomial features.

Starting from multiple linear regression:

$$y = w_0 + w_1 x_1 + w_2 x_2 + \cdots + w_p x_p$$

If we substitute:

$$x_1 = x, \quad x_2 = x^2, \quad x_3 = x^3, \ldots$$

We get:

$$y = w_0 + w_1 x + w_2 x^2 + w_3 x^3 + \cdots + w_n x^n$$

Polynomial Regression Equation

The polynomial regression equation is:

$$y = w_0 + w_1 x + w_2 x^2 + \cdots + w_n x^n$$

We represent polynomial features as:

$$\mathbf{X} = \begin{bmatrix} 1 & x_1 & x_1^2 & \dots & x_1^n \\ 1 & x_2 & x_2^2 & \dots & x_2^n \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ 1 & x_m & x_m^2 & \dots & x_m^n \end{bmatrix}, \quad \mathbf{w} = \begin{bmatrix} w_0 \\ w_1 \\ w_2 \\ \vdots \\ w_n \end{bmatrix}$$

Plolynomial Regression Training

Similar to linear regression but with polynomial terms

Minimize the sum of squared errors (SSE)

$$\text{minimize} \sum_{i=1}^{n} (y_i - (w_0 + w_1 x_i + w_2 x_i^2 + \cdots + w_n x_i^n))^2$$

Beware Overfitting!

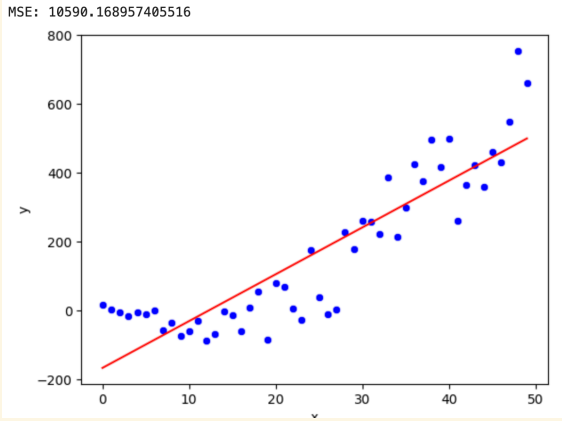
High-degree polynomials can fit the training data perfectly but generalize poorly to new data.

The best way to avoid overfitting is with **regularization**. This is when we add a penalty term to the loss function that discourages large coefficients.

Fitting a Regression

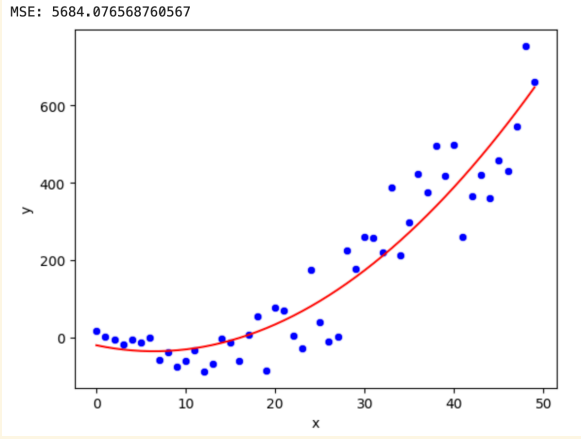
Underfit

Model is too simple and misses the underlying pattern



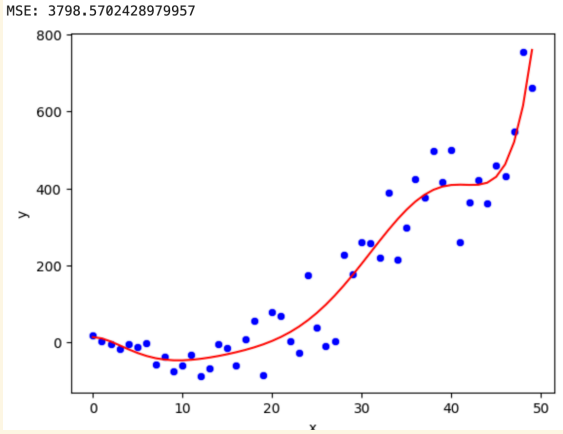
Good Fit

Model captures the underlying pattern without noise



Overfit

Model is too complex and captures noise



Real-World Example

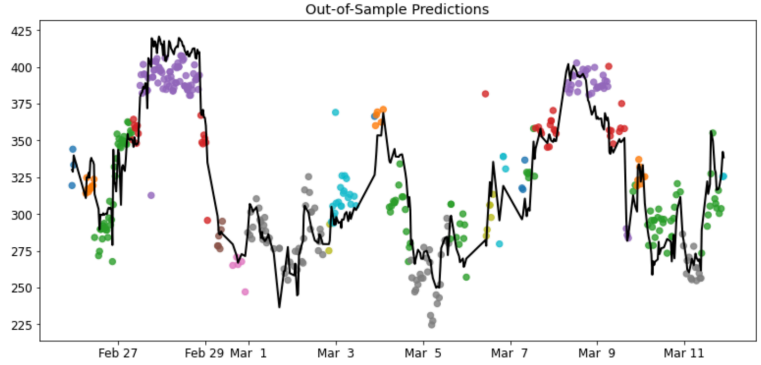
Outer Wall Thickness of an extruded vinyl profile is measured and recorded manually using a cut profile and a pair of calipers once every 12 hours.

- If this wall is **too thin**, the profile will create failure points.
- If this wall is **too thick**, the profile will be too heavy and expensive.
- Customer is **losing $400K/mo** in overage.

Metrics

- **Main Feed Speed**: Speed at which substrate material feeds into the main extruder.
- **Main Drive Amps**: Electrical load on the main extruder's motor.
- **Puller Speed**: Speed at which the product is pulled from the die.
- **Die and Barrel Zone Temperatures**: Temperature at the die and extruder barrel.

Dataset	Accuracy (>90%)	Correlation (>70%)	P95 error (<3 sigma)
Within-Sample	96%	94%	1.99 sigma
Out-of-Sample	95%	93%	1.73 sigma



Model

- Lasso Regression scored against a 60-minute rolling window
- Pre-aggregated data into 5-minute buckets

Exercise: Predicting Passengers

bigd103.link/linear-regression

Supervised Classification

Exercise

On average, which of the following Pokemon cards does the most damage? Write your answer in chat.

A Brief Interlude into Probability

Expected Value

Expected Value is a way to measure the average outcome of a random variable.

For example, “What’s the expected value of a standard die roll?” you’d add its possible outcomes and divide by 6:

$$\mathbb{E}[\text{die}] = \frac{1+2+3+4+5+6}{6} = 3.5$$

Using this, we can answer:

$$\begin{aligned}\mathbb{E}[\text{Quick Attack}] &= \left(\frac{1}{2}\right) \times 30 + \left(\frac{1}{2}\right) \times 10 \\ &= 15 + 5 \\ &= 20\end{aligned}$$

Multiplication Rule

The probability of something happening is denoted as:

Two independent events have a joint probability:

For example, if the probability of flipping a coin on heads once is 1/2 than the probability of flipping it twice and getting heads both times is:

Multiplication Rule

Considering the probabilities of flipping a coin *until* we get tails:

$$
\begin{aligned}
P(T) &= \tfrac{1}{2} \\
P(HT) &= \tfrac{1}{2} \times \tfrac{1}{2} = \tfrac{1}{4} \\
P(HHT) &= \dots
\end{aligned}
$$

We can sum these probabilities together until ∞ as a geometric series which converges to 1:

$$
\sum_{k=0}^{\infty} k \left(\frac{1}{2}\right)^{k+1} = 1.
$$

Therefore:

$$
\mathbb{E}[\text{Cont. Steps}] = 20 \times 1 = 20
$$

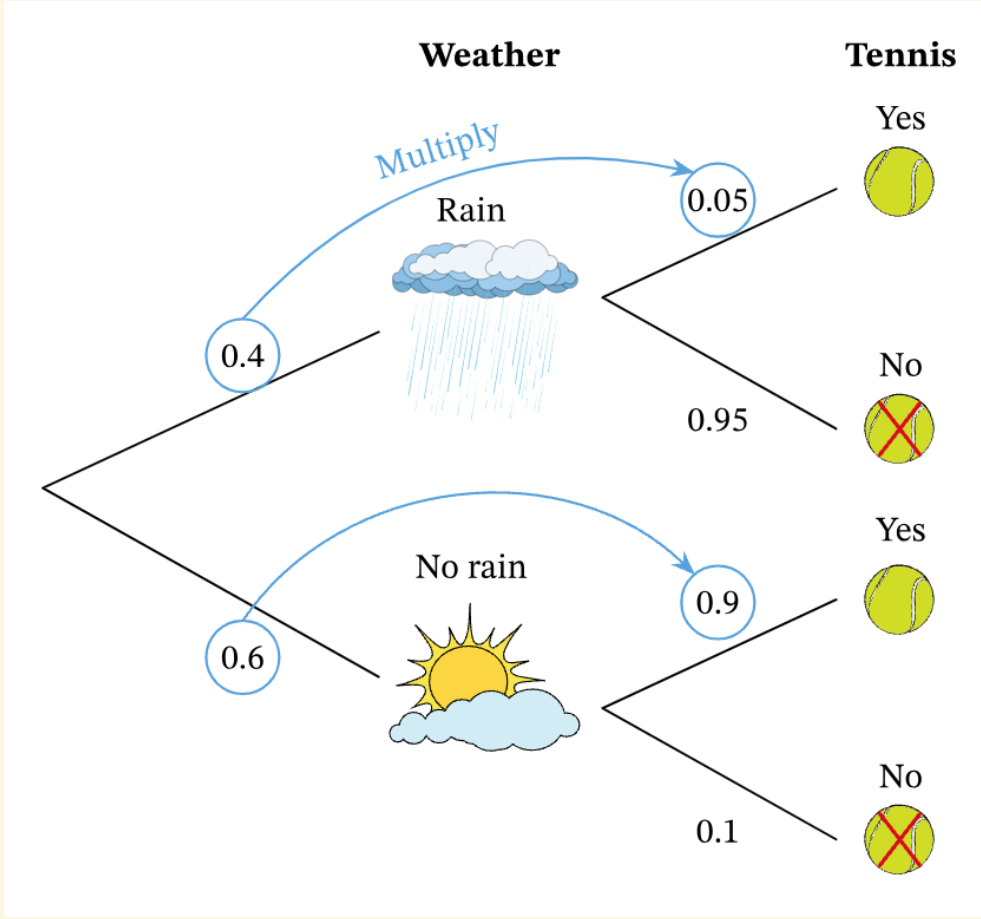
Conditional Probability

The probability of the positive class y given some non-independent event x is denoted as:

$$P(y = 1|x)$$

Bayes' theorem relates the probability of the positive class to the likelihood and prior probability:

$$P(y = 1|x) = \frac{P(x|y{=}1)P(y{=}1)}{P(x)}$$



COVID-19 Infection Example

Question: I have a cough, what's the likelihood I have COVID-19?

- $P(y = 1)$: Prior probability of having COVID-19 (e.g., infection rate in the population)
- $P(x|y = 1)$: Likelihood of having symptoms given that you have COVID-19
- $P(x)$: Probability of observing the symptoms (e.g., general probability of being sick)

Using Bayes' theorem:

$$P(\text{COVID}|\text{symptoms}) = \frac{P(\text{symptoms}|\text{COVID}) \cdot P(\text{COVID})}{P(\text{symptoms})}$$

Example Calculation

Suppose:

- $P(\text{COVID}) = 0.05$ (5% of the population is infected)
- $P(\text{symptoms}|\text{COVID}) = 0.90$ (90% of infected people show symptoms)
- $P(\text{symptoms}) = 0.20$ (20% of the population shows symptoms)

Then:

$$P(\text{COVID}|\text{symptoms}) = \frac{0.90 \times 0.05}{0.20} = \frac{0.045}{0.20} = 0.225$$

Interpretation:

*Given that you have symptoms, there is a **22.5% probability that you have COVID-19**.*

Logistic Regression

Logistic Regression

Logistic regression is used for **binary classification** problems (e.g., is this email spam or not?).

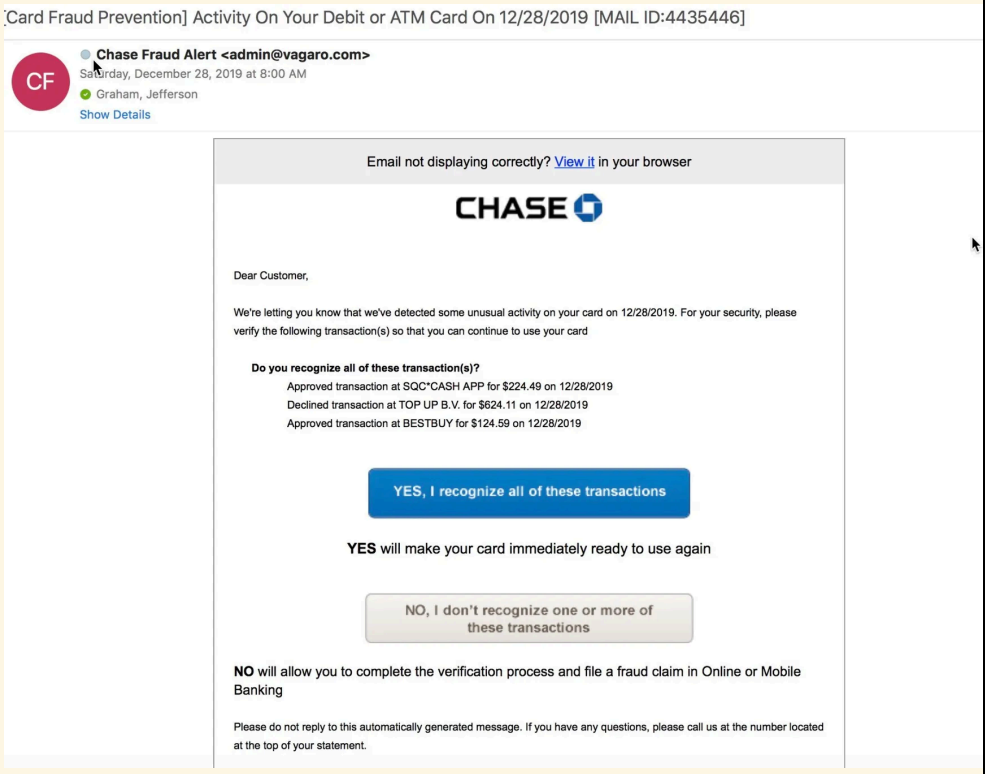
It models the probability that a given input x belongs to a particular category (often "positive" vs. "negative").

Credit Card Fraud Detection

Features: Vendor, location, time, distance from last transaction

Labels: Chargebacks on previous transactions

Modeled Relationship: The probability of a transaction being fraudulent.



From Bayes to Regression

Bayes' theorem gives us the probability of class given features:

$$P(y=1|x) = \frac{P(x|y=1)P(y=1)}{P(x)}$$

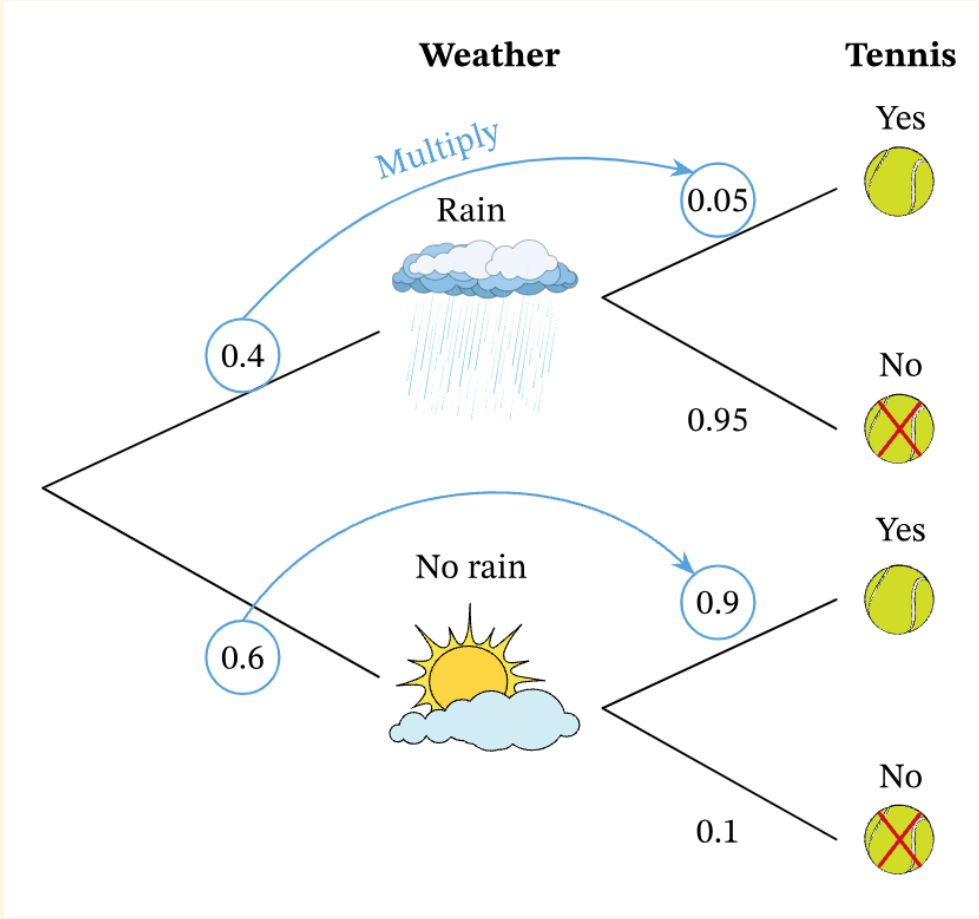
But what if we don't explicitly know these probabilities?

Logistic regression helps us learn these directly from data:

Instead of estimating probabilities explicitly, we model the **log-odds** as a linear function:

$$\log \frac{P(y=1|x)}{1-P(y=1|x)} = w_0 + w_1 x_1 + \cdots + w_p x_p$$

Then, the logistic function converts log-odds to a probability.



Logistic Function

We transform linear combinations of features to probabilities using the **logistic (sigmoid) function**:

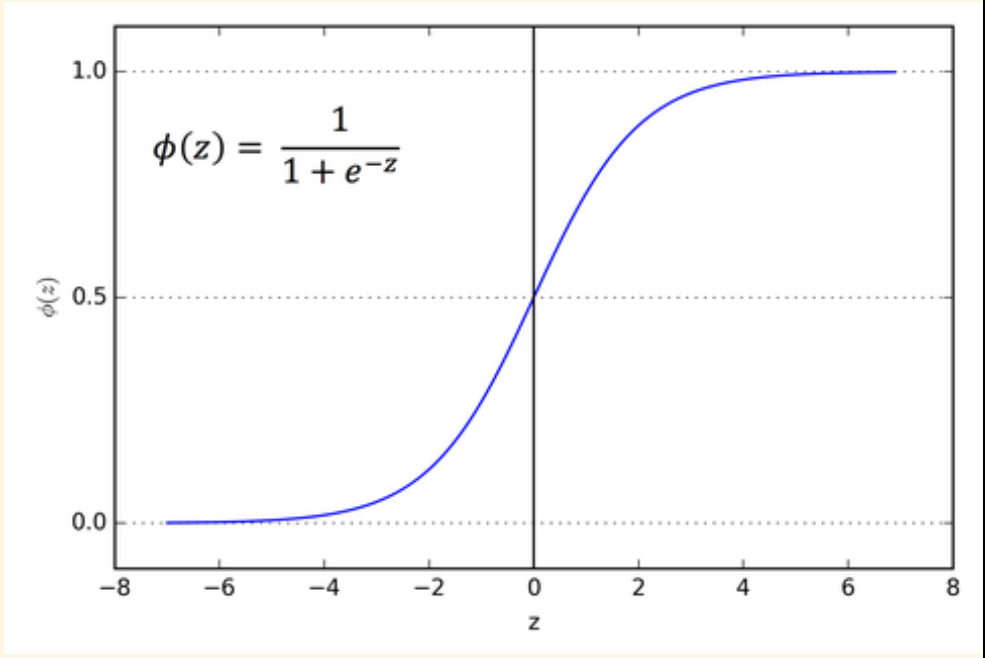
$$P(y = 1 \mid x) = \sigma(z) = \frac{1}{1+e^{-z}}$$

Where:

$$z = w_0 + w_1 x_1 + \cdots + w_p x_p$$

- z: Linear combination of features.
- $\sigma(z)$: Probability between 0 and 1.

While z can take any value, the output of $\sigma(z)$ is always between 0 and 1.



Decision Boundary

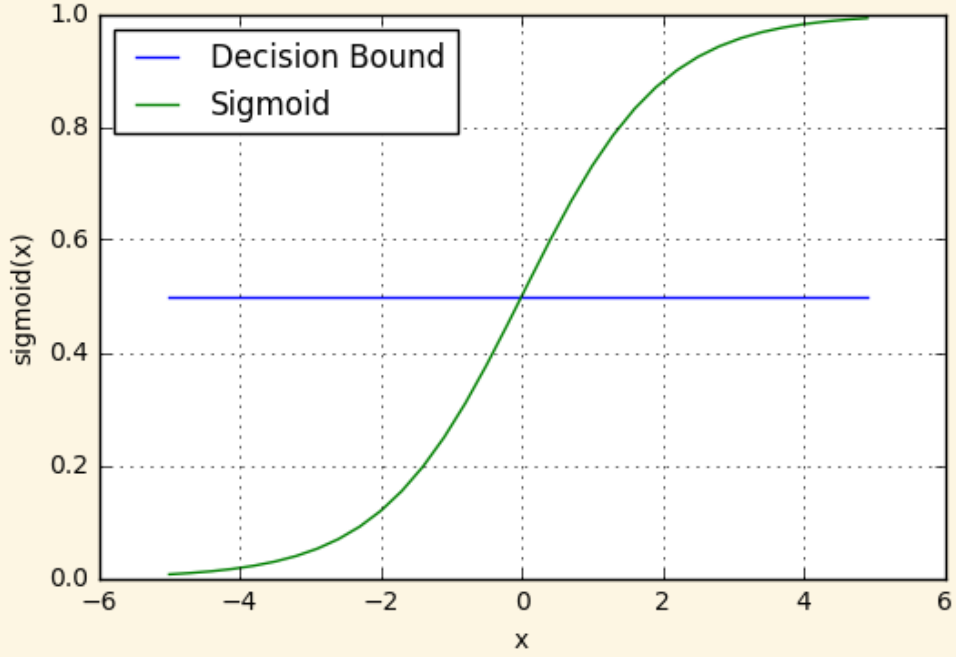
The decision boundary is defined where the model predicts $P(y = 1 \mid x) = 0.5$.

Equivalently, $\sigma(z) = 0.5 \rightarrow z = 0$.

Loss Function and Optimization

Uses the **Cross-Entropy Loss** (also called Log Loss).

Parameters (w_i) are optimized using **gradient descent** to minimize this loss.



Odds and Log-Odds

Odds

The ratio of the probability of an event to the probability of not the event.

$$\text{Odds} = \frac{P(y=1|x)}{1-P(y=1|x)}$$

Log-Odds (Logit)

The natural logarithm of the odds.

$$\text{Log-Odds} = \log\left(\frac{P(y=1|x)}{1-P(y=1|x)}\right)$$

Logistic regression is linear in the **log-odds** space:

$$\log\left(\frac{P(y=1|x)}{1-P(y=1|x)}\right) = w_0 + w_1x_1 + \cdots + w_px_p$$

Interpretation of Coefficients

Each coefficient w_i represents how the **log-odds** of the positive class changes with respect to a one-unit change in x_i.

Coefficient	Interpretation
$w_i > 0$	Increasing x_i increases the log-odds (increasing probability)
$w_i < 0$	Increasing x_i decreases the log-odds (decreasing probability)

Measuring Performance

Accuracy

Layman friendly. Can be misleading for imbalanced datasets.

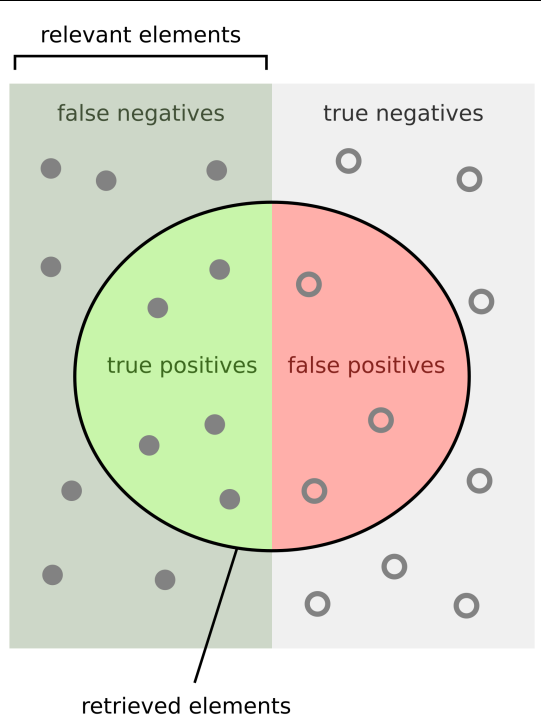
$$\text{Accuracy} = \frac{\text{Correct Predictions}}{\text{Total Predictions}}$$

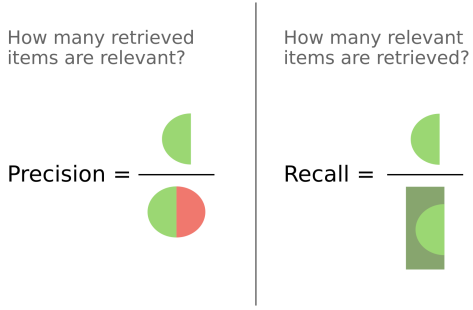
Precision and Recall

Less intuitive but more informative for imbalanced datasets.

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

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





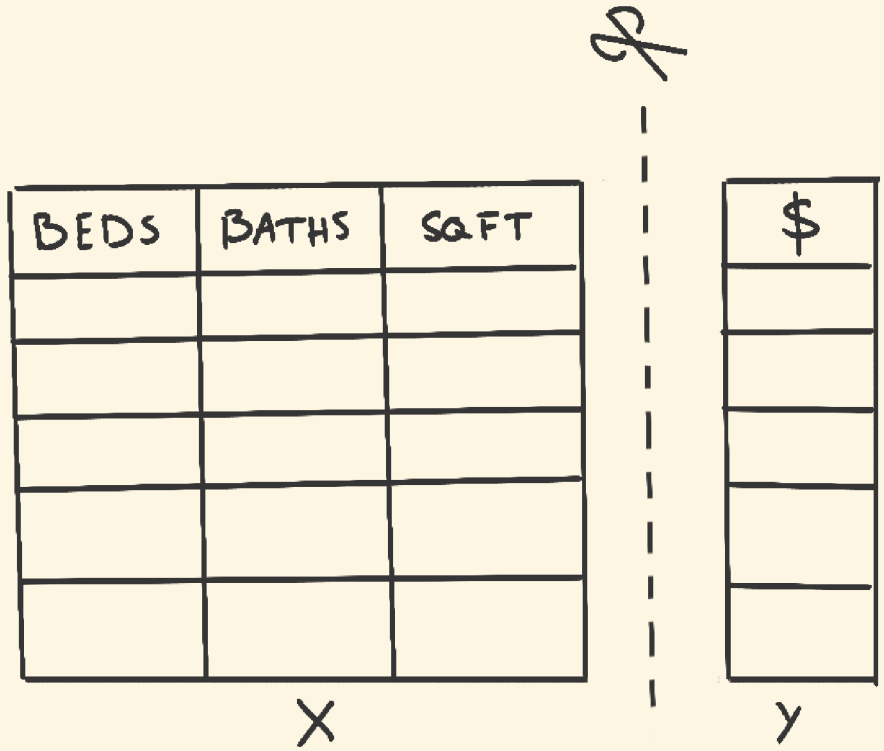
Training a Supervised Learning Model

(1) Feature / Label Split

- **Features**: Input data used to make predictions (usually multiple columns).
- **Labels**: Output data to predict (usually a single column).

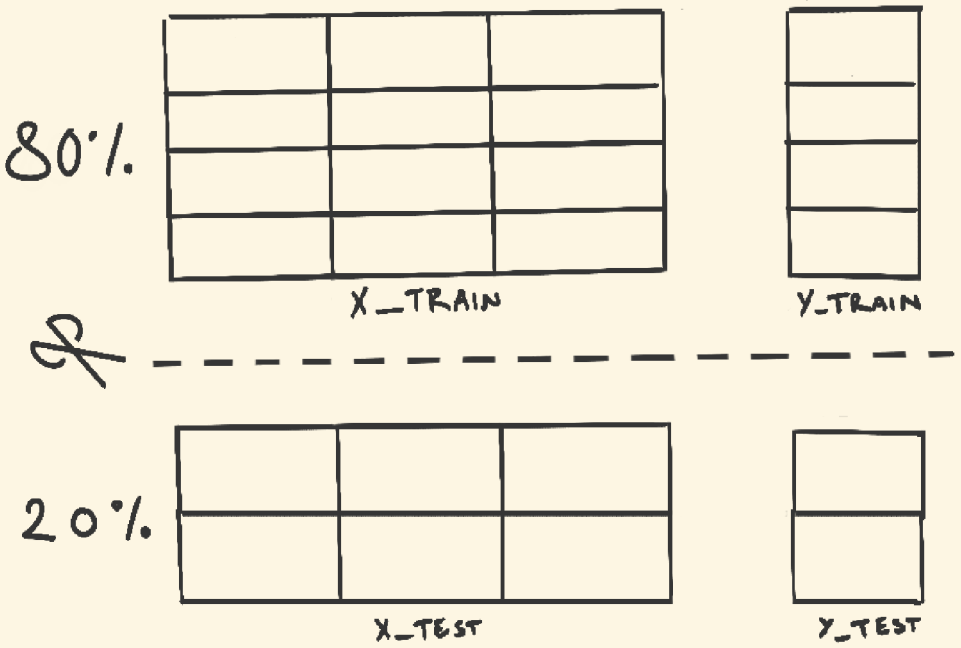
```
# Load data
df = pd.read_csv('data.csv')

# Split features and labels
X = df.drop(columns=['label'])
y = df['label']
```



(2) Train / Test Split

- **Training Set**: Used to train the model.
- **Test Set**: Used to evaluate the model's performance.



With Pandas

```
# Create a train/test split mask
is_train = np.random.rand(len(X)) < 0.8

# Split data
X_train, y_train = X[is_train], y[is_train]
X_test, y_test = X[~is_train], y[~is_train]
```

With Sklearn

```
from sklearn.model_selection import \
    train_test_split

X_train, X_test, y_train, y_test = \
    train_test_split(X, y, test_size=0.2)
```

Deciding the Split Method

Random Split

Shuffle the dataset randomly, then allocate a percentage for training and the remaining for testing.

When to Use: Independent and identically distributed (i.i.d.) data.

Time-Based Split

The model must predict future events based past data. We split chronologically (e.g., train on 2000 to 2020, test on 2020 to 2025.)

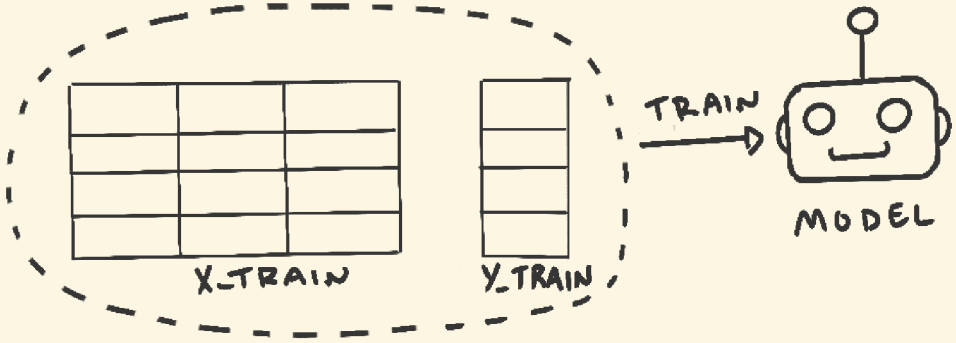
When to Use: Time-series, sequential data.

Other Splits

Stratified Split, Group Split, Cross-Validation

(3) Model Training

- **Choose a Model**: Select an algorithm that fits the problem.
- **Fit the model**: Train the model on the training data.
- **Exclude the Test Set**: The model should not see the test set during training.




```
from sklearn.linear_model import \
    LinearRegression

model = LinearRegression()
model.fit(X_train, y_train)
```

(4) Model Evaluation

- **Predict**: Use the model to make predictions on the test features.
- **Evaluate**: Compare predicted labels to the test labels with an evaluation metric.

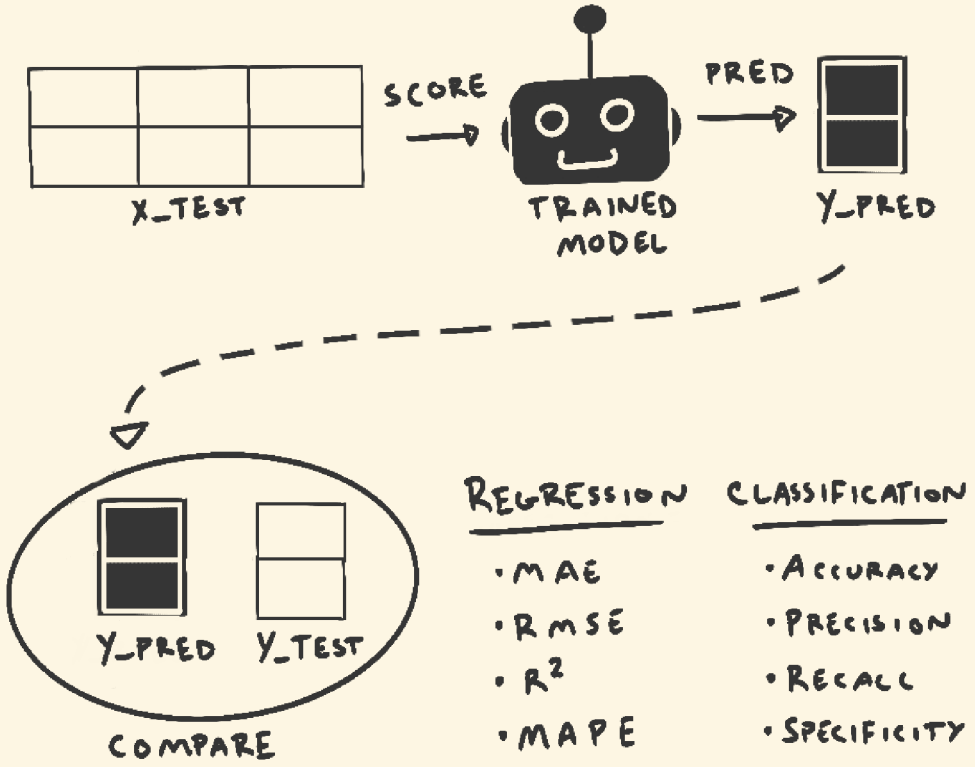
Regression Evaluation

```
from sklearn.metrics import mean_squared_error
y_pred = model.predict(X_test)
mse = mean_squared_error(y_test, y_pred)
print(f'Regression Mean Squared Error: {mse}')
```

Classification Evaluation

```
from sklearn.metrics import accuracy_score
y_pred = model.predict(X_test)
accuracy = accuracy_score(y_test, y_pred)
print(f'Classification Accuracy: {accuracy}')
```

Different evaluation metrics are used for regression and classification problems.



Exercise: Predicting College Admissions

bigd103.link/logistic-regression

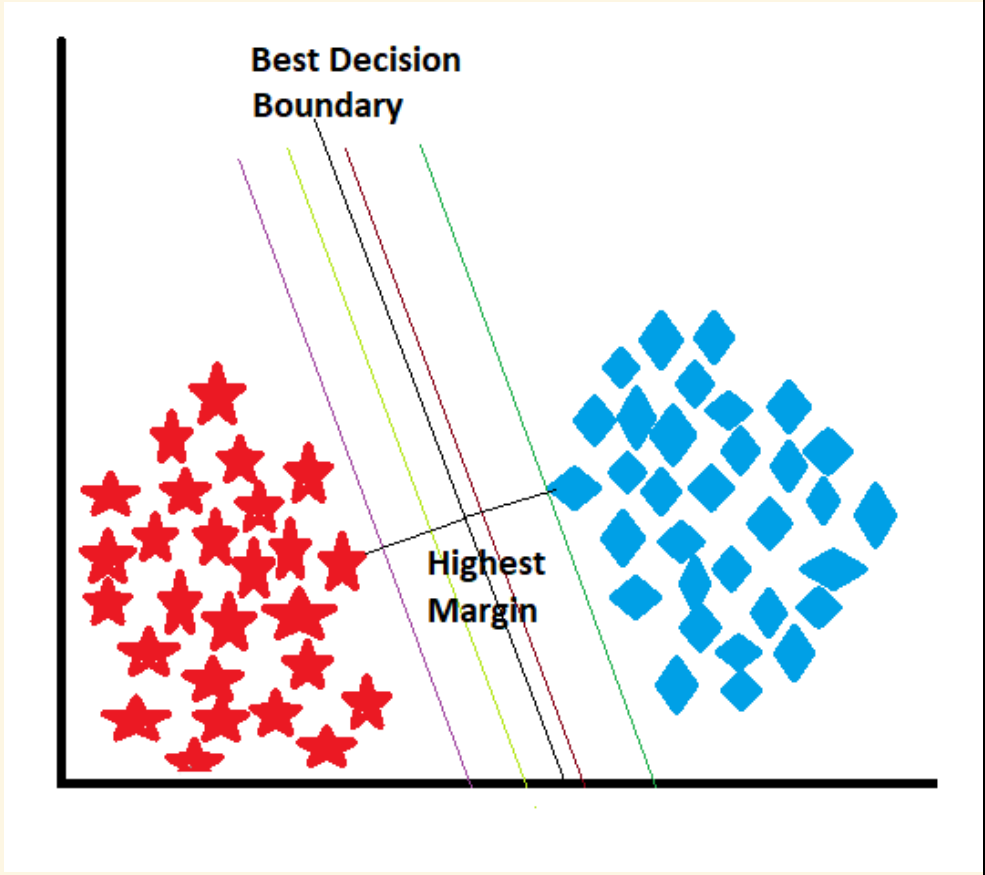
Support Vector Machines (SVM)

What is an SVM?

Support Vector Machines are powerful classifiers that find the **optimal decision boundary** between classes.

Key idea: Don’t just find any line that separates the classes - find the line with the **maximum margin**.

Margin: The distance between the decision boundary and the nearest data points from each class.



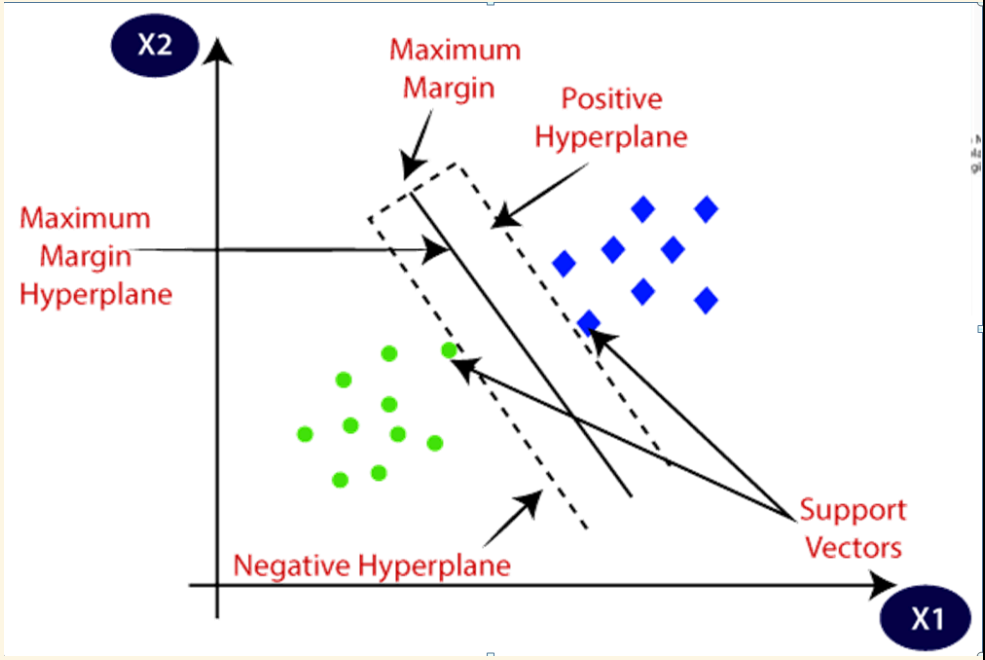
The Maximum Margin Principle

Question: Which line is better?

All three lines correctly separate the classes, but the SVM chooses the one with the largest margin.

Why? A larger margin means:

- More confidence in predictions
- Better generalization to new data
- More robust to small changes



Support Vectors

Support vectors are the data points that:

- Lie closest to the decision boundary
- Actually determine where the boundary is placed
- Would change the boundary if removed

This is very different from logistic regression, which uses all points.

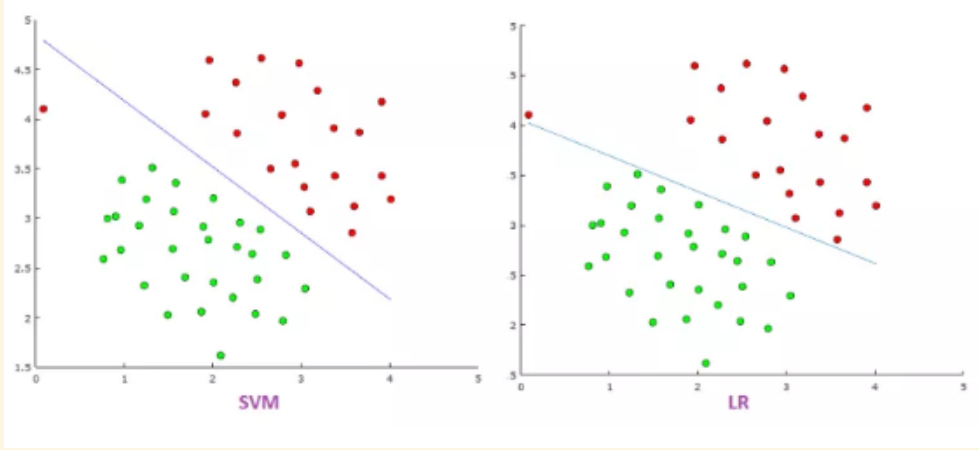
SVM vs Logistic Regression

Logistic Regression

- Considers **all** data points
- Outputs probabilities
- Minimizes log loss

SVM

- **Focuses on points near the boundary**
- Outputs class labels (-1 or +1)
- Maximizes the margin



Linear SVM Math

For a linear SVM, we're trying to find a line (or hyperplane):

$$w^T x + b = 0$$

Such that:

- Points of class +1 satisfy: $w^T x + b \geq +1$
- Points of class -1 satisfy: $w^T x + b \leq -1$
- The margin width is: $\frac{2}{||w||}$

Goal: Minimize $||w||$ (which maximizes the margin)

In Python (using Scikit-learn)

```
from sklearn.svm import SVC

# Create a linear SVM classifier
model = SVC(kernel='linear')

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

# Make predictions
y_pred = model.predict(X_test)

# Check the support vectors
print(f"Number of support vectors: {len(model.support_vect
print(f"Total training points: {len(X_train)}")
# Usually, only a small fraction are support vectors!
```

The Kernel Trick

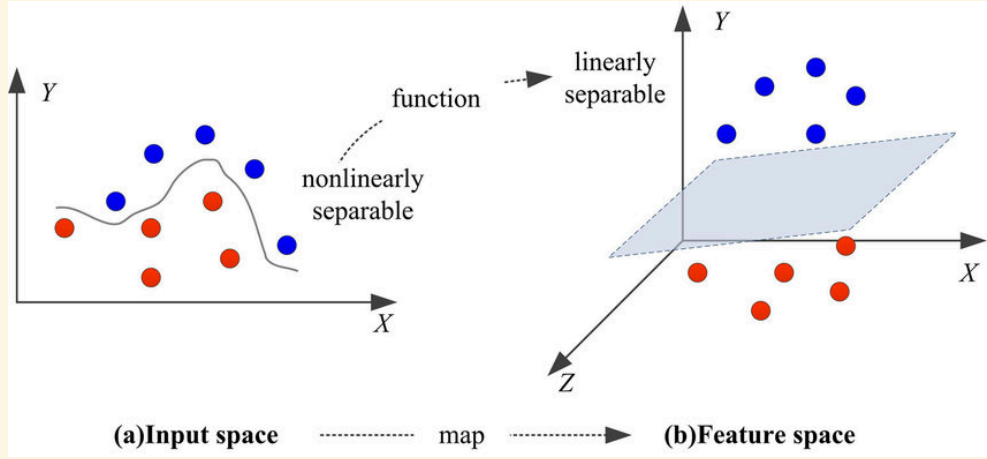
Problem: What if the data isn't linearly separable?

Solution: The kernel trick allows SVMs to:

1. Map data to a higher-dimensional space
2. Find a linear boundary in that space
3. Which appears non-linear in the original space!

Common Kernels

- **Linear**: no transformation
- **RBF**: Radial Basis Function
- **Polynomial**: polynomial transformation



SVM in Practice

```
from sklearn.model_selection import train_test_split
from sklearn.svm import SVC
from sklearn.preprocessing import StandardScaler

# Split the data
X_train, X_test, y_train, y_test = train_test_split(X, y,

# IMPORTANT: Scale features for SVM!
scaler = StandardScaler()
X_train_scaled = scaler.fit_transform(X_train)
X_test_scaled = scaler.transform(X_test)
```

```
# Train SVM with RBF kernel
svm = SVC(kernel='rbf', C=1.0, gamma='scale')
svm.fit(X_train_scaled, y_train)

# Evaluate
y_pred = svm.predict(X_test_scaled)
tp, tn, fp, fn = calculate_confusion_matrix(y_test, y_pred
accuracy = (tp + tn) / (tp + tn + fp + fn)
print(f"Accuracy: {accuracy:.1%}")
```

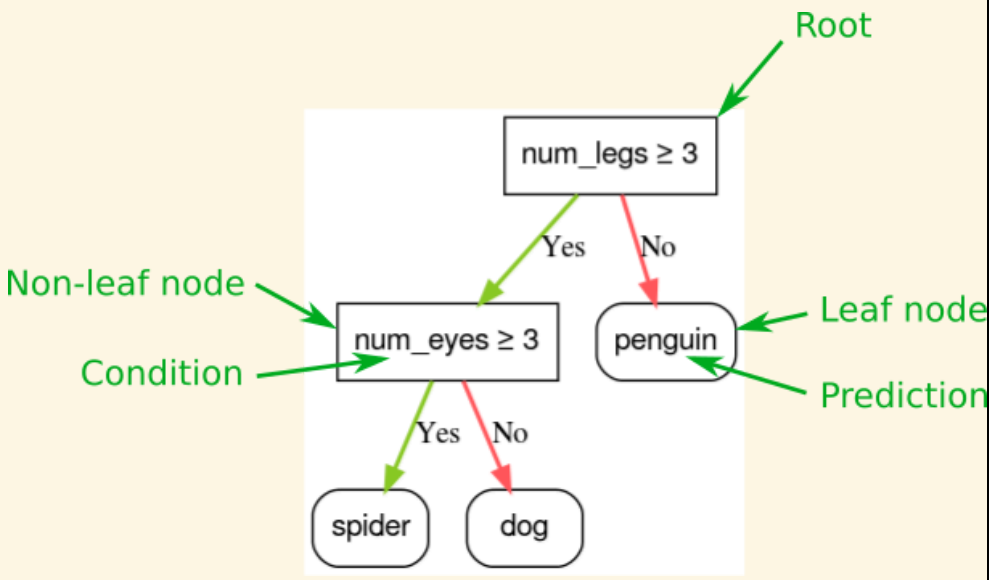
Note: SVMs are sensitive to feature scales! Always **normalize** your data.

Decision Trees

What is a Decision Tree?

A decision tree is a flowchart-like structure used for classification and regression tasks.

It recursively splits the dataset into subsets based on feature values, forming a tree of decisions.

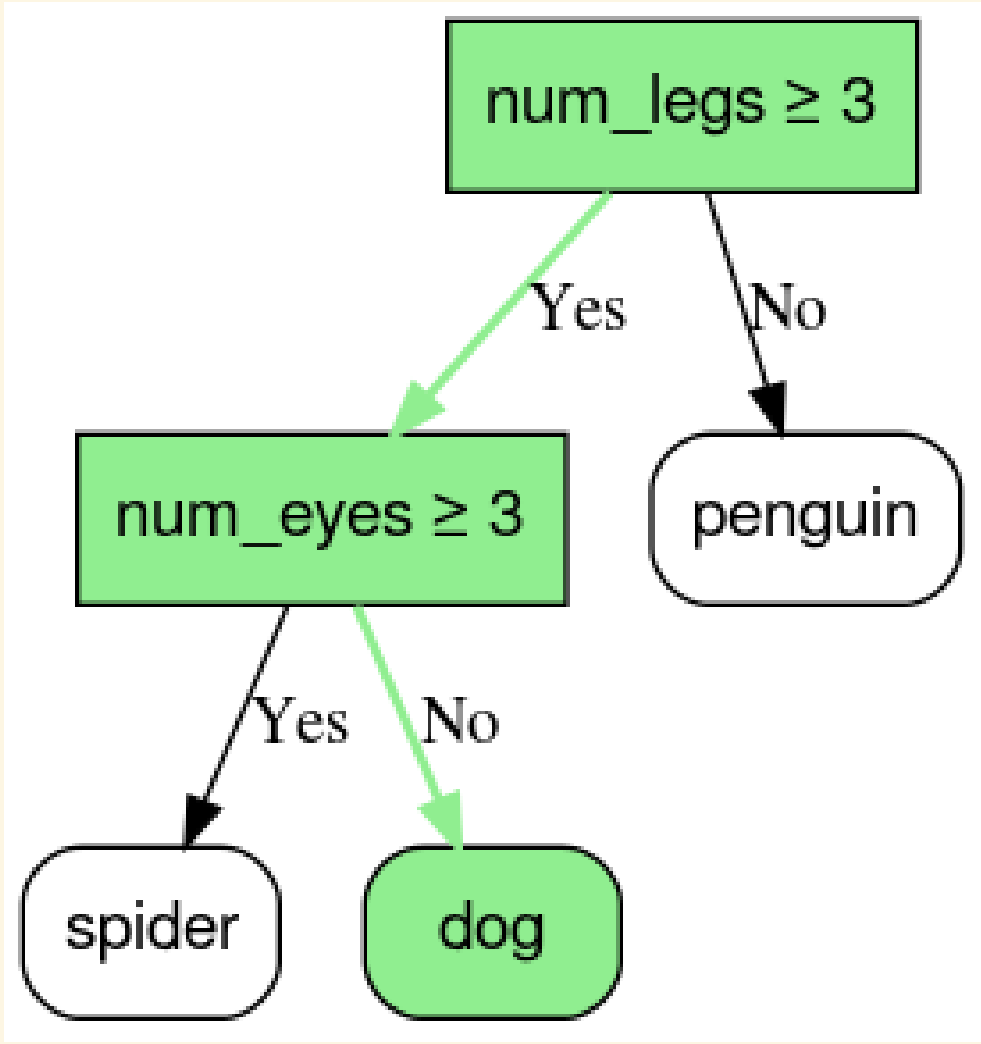


Decision Tree Prediction

- **Objective**: Predict the class label of an instance based on its feature values.
- **Input**: Numerical or categorical features and class labels.
- **Output**: A decision tree that classifies instances into predefined classes.

```
X = pd.DataFrame({
  "num_legs": [4],
  "num_eyes": [2],
})
print(model.score(X))
```

```
dog
```

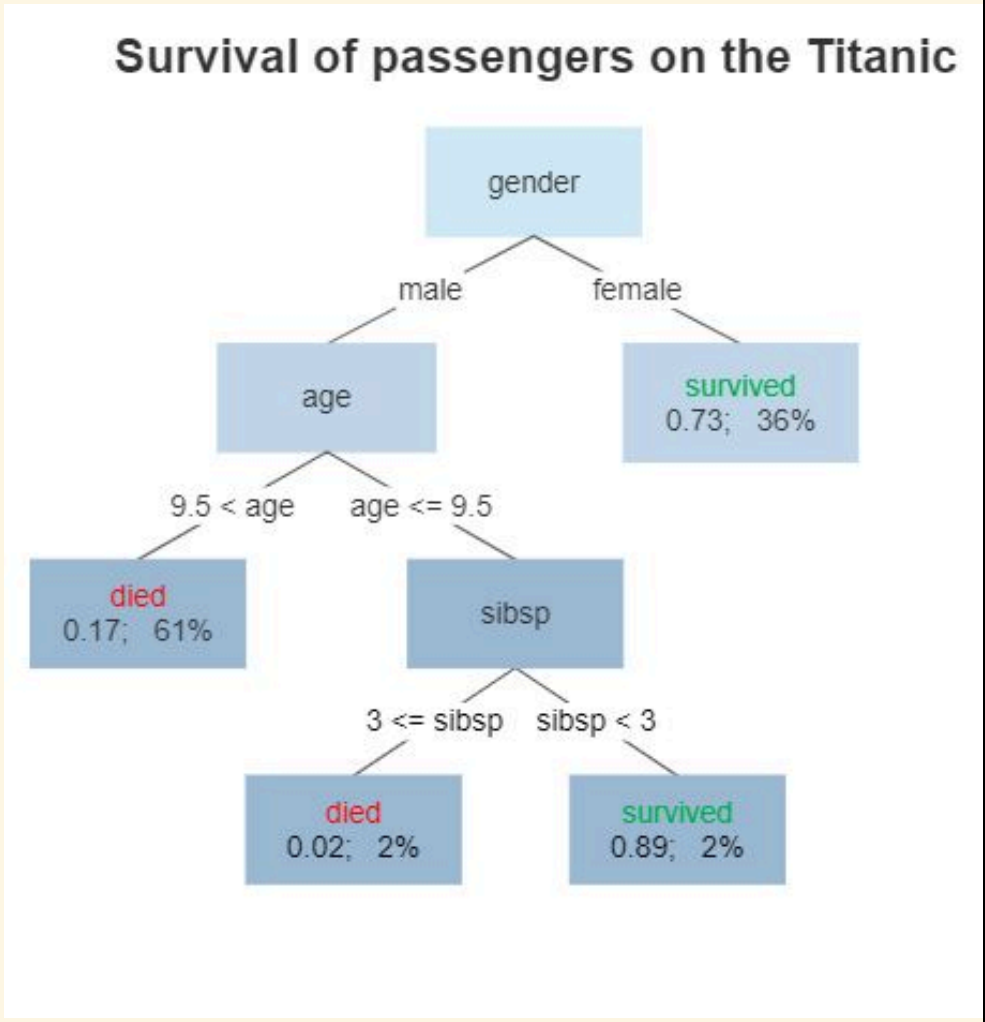


Decision Trees are Intuitive

- **Interpretability**: Easy to understand and visualize.
- **Feature Selection**: Automatically selects important features.
- **Non-Parametric**: No assumptions about the underlying data distribution.

Titanic Example

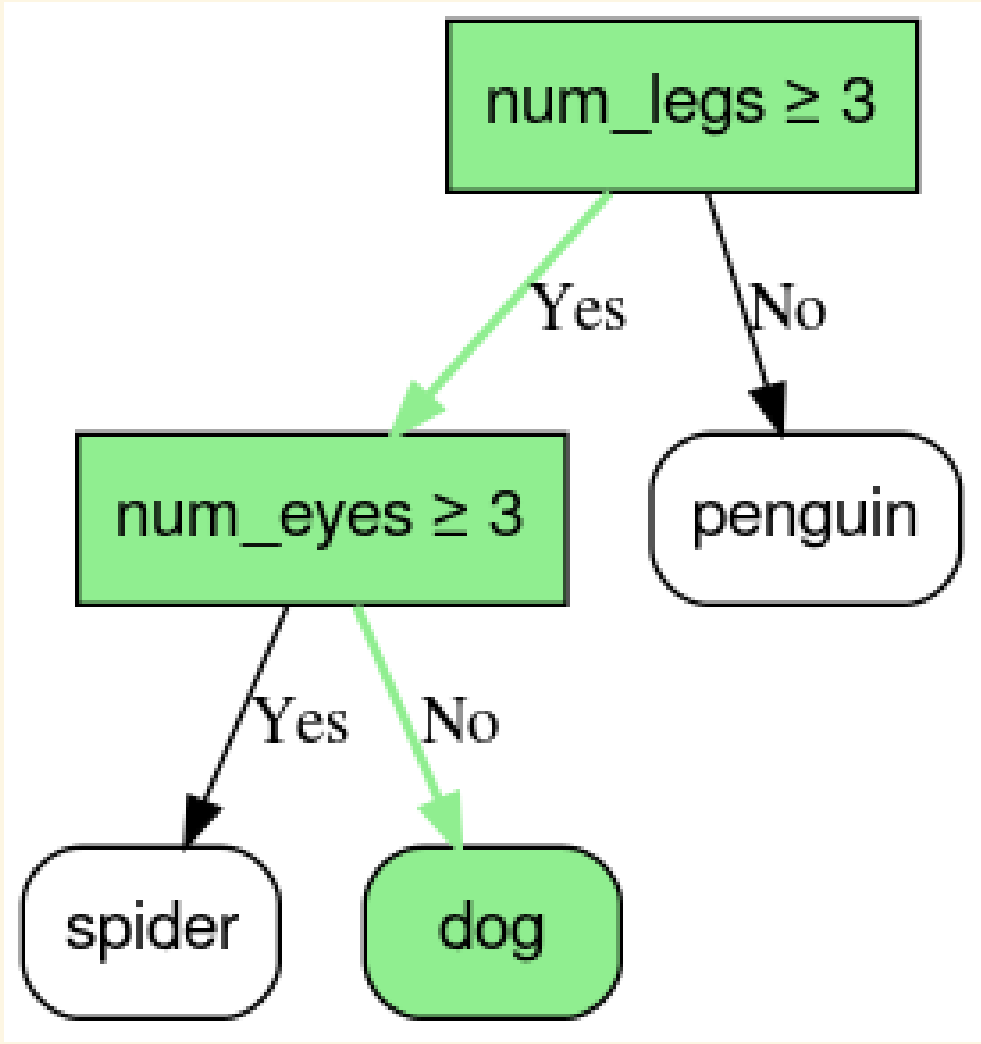
- Identifies Gender as most important feature
- Following the tree predicts the survival probability
- Leaf nodes contain the class label

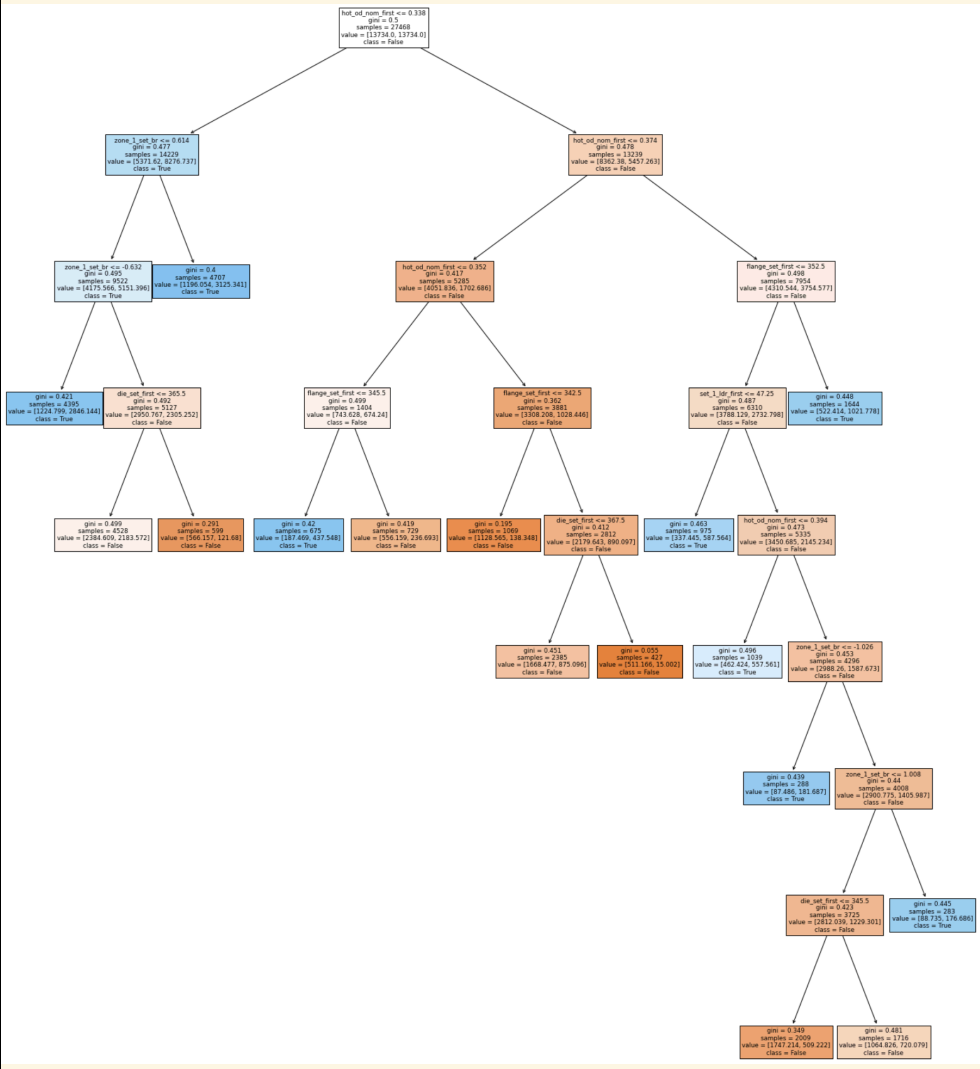


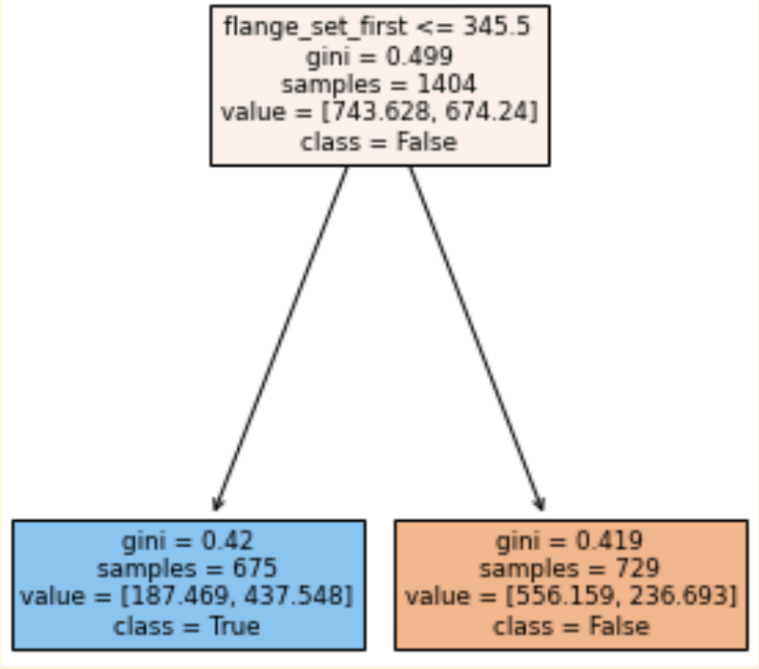
Portability

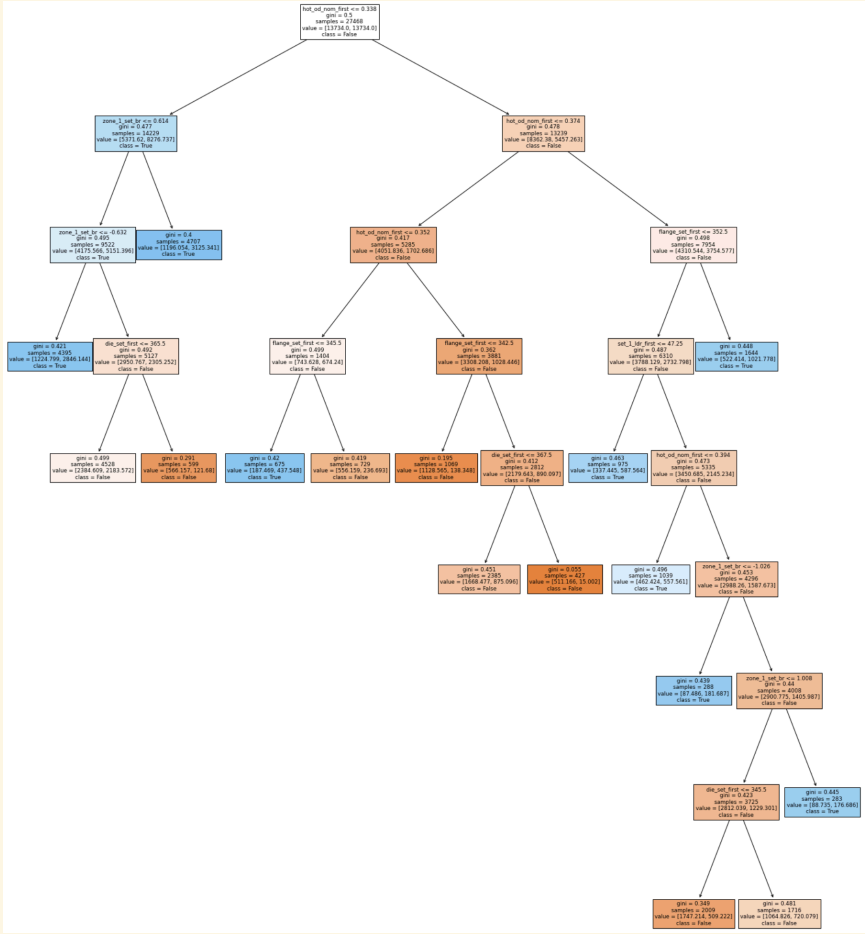
Decision trees can be easily transferred to other formats (e.g., rules) for use in other systems.

```
def predict(x):
    num_legs = x["num_legs"]
    num_eyes = x["num_eyes"]
    if num_legs >= 3:
        if num_eyes >= 3:
            return "spider"
        else:
            return "dog"
    else:
        return "penguin"
```




hot_od_nom_first <= 0.338
gini = 0.5
samples = 27468
value = [13734.0, 13734.0]
class = False
zone_1_set_br <= 0.614
gini = 0.477
samples = 14229
value = [5371.62, 8276.737]
class = True
hot_od_nom_first <= 0.374
gini = 0.478
samples = 13239
value = [8362.38, 5457.263]
class = False
zone_1_set_br <= -0.632
gini = 0.495
samples = 9522
value = [4175.566, 5151.396]
class = True
gini = 0.4
samples = 4707
value = [1196.054, 3125.341]
class = True
hot_od_nom_first <= 0.352
gini = 0.417
samples = 5285
value = [4051.836, 1702.686]
class = False
flange_set_first <= 352.5
gini = 0.498
samples = 7954
value = [4310.544, 3754.577]
class = False
gini = 0.421
samples = 4395
value = [1224.799, 2846.144]
class = True
die_set_first <= 365.5
gini = 0.492
samples = 5127
value = [2950.767, 2305.252]
class = False
flange_set_first <= 345.5
gini = 0.499
samples = 1404
value = [743.628, 674.24]
class = False
flange_set_first <= 342.5
gini = 0.362
samples = 3881
value = [3308.208, 1028.446]
class = False
set_1_ldr_first <= 47.25
gini = 0.487
samples = 6310
value = [3788.129, 2732.798]
class = False
gini = 0.448
samples = 1644
value = [522.414, 1021.778]
class = True
gini = 0.499
samples = 4528
value = [2384.609, 2183.572]
class = False
gini = 0.291
samples = 599
value = [566.157, 121.68]
class = False
gini = 0.42
samples = 675
value = [187.469, 437.548]
class = True
gini = 0.419
samples = 729
value = [556.159, 236.693]
class = False
gini = 0.195
samples = 1069
value = [1128.565, 138.348]
class = False
die_set_first <= 367.5
gini = 0.412
samples = 2812
value = [2179.643, 890.097]
class = False
gini = 0.463
samples = 975
value = [337.445, 587.564]
class = True
hot_od_nom_first <= 0.394
gini = 0.473
samples = 5335
value = [3450.685, 2145.234]
class = False
gini = 0.451
samples = 2385
value = [1668.477, 875.096]
class = False
gini = 0.055
samples = 427
value = [511.166, 15.002]
class = False
gini = 0.496
samples = 1039
value = [462.424, 557.561]
class = True
zone_1_set_br <= -1.026
gini = 0.453
samples = 4296
value = [2988.26, 1587.673]
class = False
gini = 0.439
samples = 288
value = [87.486, 181.687]
class = True
zone_1_set_br <= 1.008
gini = 0.44
samples = 4008
value = [2900.775, 1405.987]
class = False
die_set_first <= 345.5
gini = 0.423
samples = 3725
value = [2812.039, 1229.301]
class = False
gini = 0.445
samples = 283
value = [88.735, 176.686]
class = True
gini = 0.349
samples = 2009
value = [1747.214, 509.222]
class = False
gini = 0.481
samples = 1716
value = [1064.826, 720.079]
class = False


flange_set_first <= 345.5
gini = 0.499
samples = 1404
value = [743.628, 674.24]
class = False
gini = 0.42
samples = 675
value = [187.469, 437.548]
class = True
gini = 0.419
samples = 729
value = [556.159, 236.693]
class = False



```
WHERE
  (
    (hot_od_nom_first < 0.339) AND (zone_1_set_br >
  ) -- class: True (proba: 72.32%) based on 4,707 sa
  OR (
    (hot_od_nom_first < 0.339) AND (zone_1_set_br <
  ) -- class: True (proba: 69.91%) based on 4,395 sa
  OR (
    (hot_od_nom_first > 0.338) AND (hot_od_nom_first
  ) -- class: True (proba: 66.17%) based on 1,644 sa
  OR (
    (hot_od_nom_first > 0.338) AND (hot_od_nom_first
    AND (flange_set_first < 352.501) AND (set_1_ldr_
    AND (hot_od_nom_first < 0.395)
  ) -- class: True (proba: 54.66%) based on 1,039 sa
  OR (
    (hot_od_nom_first > 0.338) AND (hot_od_nom_first
    AND (flange_set_first < 352.501) AND (set_1_ldr_
  ) -- class: True (proba: 63.52%) based on 975 samp
  OR (
    (hot_od_nom_first > 0.338) AND (hot_od_nom_first
    AND (hot_od_nom_first < 0.353) AND (flange_set_
  ) -- class: True (proba: 70.01%) based on 675 samp
  OR (
    (hot_od_nom_first > 0.338) AND (hot_od_nom_first
    AND (flange_set_first < 352.501) AND (set_1_ldr_
    AND (hot_od_nom_first > 0.394) AND (zone_1_set_b
  ) -- class: True (proba: 67.5%) based on 288 samp
```

Tree Construction

- **Step 1**: Start with the entire dataset at the root.
- **Step 2**: Select the best feature to split the data based on **the chosen criterion**.
- **Step 3**: Split the data into subsets.
- **Step 4**: Recursively apply steps 2 and 3 to each subset.
- **Step 5**: Stop splitting when a stopping condition is met (e.g., maximum depth, minimum instances per node).

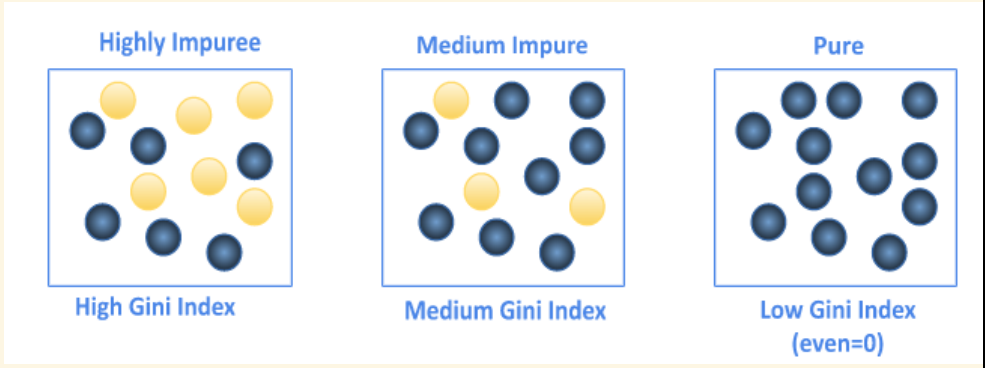
Splitting Criteria - Gini Impurity

- Measures the impurity of a node. Lower values indicate purer nodes.
- Gini impurity for a node with classes $1, 2, \ldots, C$:

$$G = \sum_{i=1}^{C} p_i(1 - p_i)$$

where p_i is the proportion of instances of class i in the node.

Intuition: If I randomly selected two items, how likely is it that they're different classes?



Splitting Criteria - Entropy

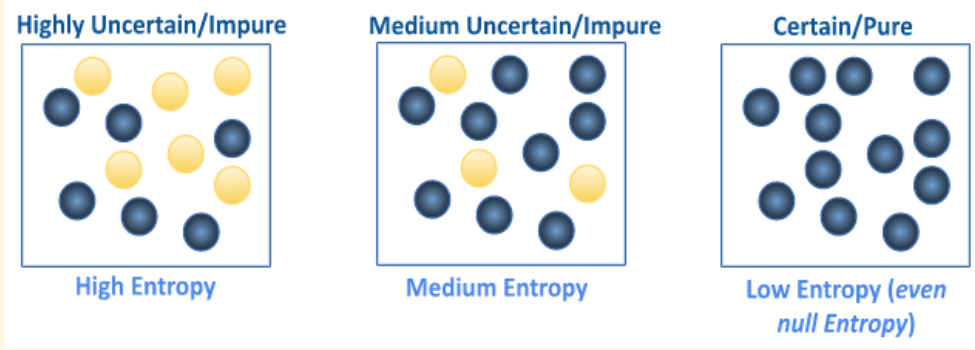
- Measures the randomness in the node.

  Lower values indicate less randomness.
- Entropy for a node with classes $1, 2, \ldots, C$:

$$H = -\sum_{i=1}^{C} p_i \log(p_i)$$

  where p_i is the proportion of instances of class i in the node.

> **Intuition:** How surprised am I on average when I see the class label?



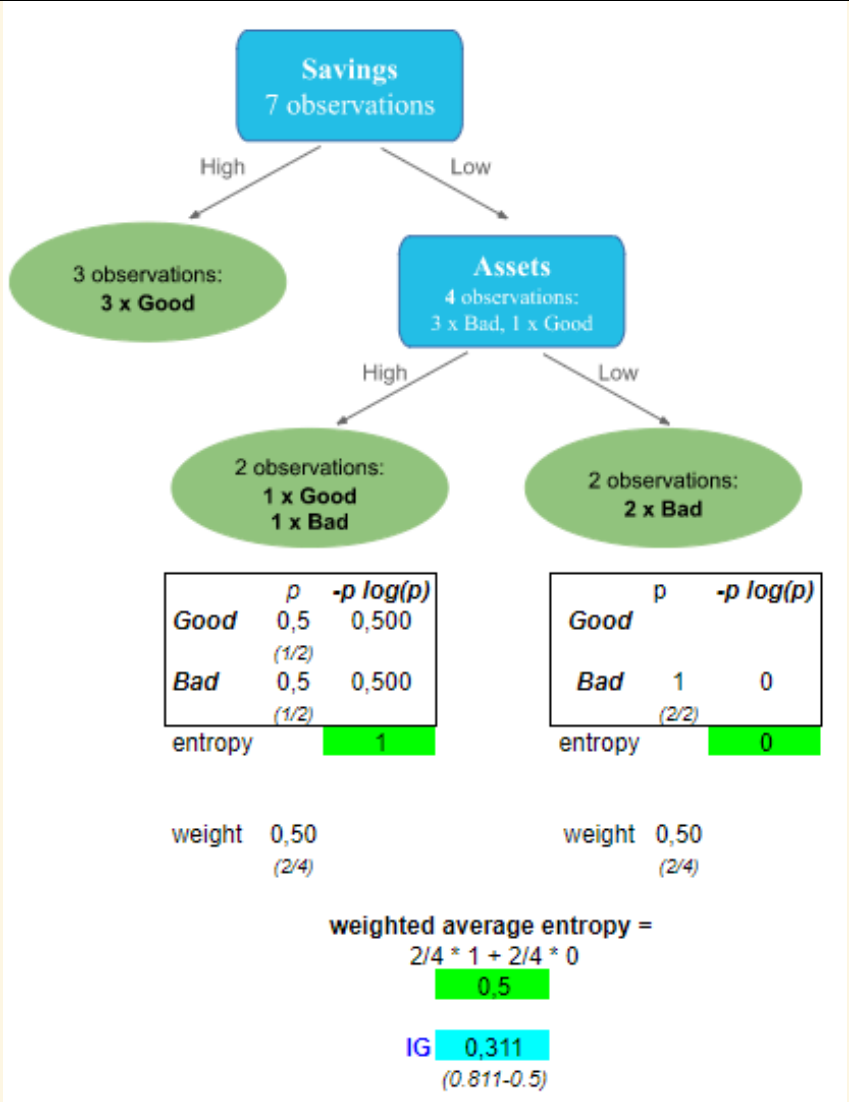
Splitting Criteria - Information Gain

- Measures the reduction in entropy after a split.
- Information Gain for a split S:

$$IG(S) = H(\text{parent}) - \sum_j \frac{|S_j|}{|S|} H(S_j)$$

where $H(\text{parent})$ is the entropy of the parent node, S_j are the subsets formed by the split, and $|S_j|$ is the number of instances in subset S_j.

Intuition: How much less surprised am I after the split?



Pruning

Reduce overfitting by removing branches.

Pre-pruning: Stop growing the tree early.

- Maximum depth.
- Minimum instances per node.

Post-pruning: Grow the tree but then remove branches that don't provide significant power.

- **Cost Complexity:** Add a penalty for tree complexity to the cost function:

  $$R_\alpha(T) = R(T) + \alpha|T|$$

  Where $R(T)$ is the misclassification rate of tree T, α is a complexity parameter, and $|T|$ is the number of leaves.

Exercise: Predicting Admissions v2

bigd103.link/decision-trees

Comparing Classifiers

Algorithm	Pros	Cons
Logistic Regression	Fast, probabilistic, interpretable	Linear only
SVM	Powerful, kernels, works in high-D	Slow, hard to interpret
Decision Trees	Interpretable, non-linear, fast	Can overfit easily

In practice: Try multiple algorithms and see what works best for your data and problem!

Machine Learning in Production

Lifecycle of an Updating Predictive Model

1. **Model Development**: Train a model on historical train set.
2. **Model Evaluation**: Assess the model's performance on historical test set.
3. **Model Deployment**: Integrate the model into a system or application.
4. **Model Monitoring**: Track the model's performance over time.
5. **Model Updates**: Update the model as needed.