About me

Associate prof. dr. Sven Degroeve (sven.degroeve@ugent.be), Staff Scientist at CompOmics, VIB-CMB

Master in Information Technology (1999)
Title of thesis: "Classification of skin lesions with neural networks"

PhD in Computer Science (2004)
Title of thesis: "Design and evaluation of a linear classification strategy for gene structural element recognition"

Member of the study programme and the IT committee

https://sdgroeve.github.io/

About me

Associate prof. dr. Sven Degroeve (sven.degroeve@ugent.be), Staff Scientist at CompOmics, VIB-CMB

Master in Information Technology (1999)
Title of thesis: "Classification of skin lesions with neural networks"

PhD in Computer Science (2004)
Title of thesis: "Design and evaluation of a linear classification strategy for gene structural element recognition"

Member of the study programme and the IT committee

https://sdgroeve.github.io/

About the course

Full calendar on Ufora

2/4 and 3/4 practicum in the afternoon: assignment 1 part 1 (due 3/4)
https://www.kaggle.com/competitions/splice-site-prediction-contest
Link to join: https://www.kaggle.com/t/e3d467bfa7db464199bcbcccced8433

6/5 assignment 1 part 2 due

16/5 assignment 2 due

Microteaching: 16/5 full day presentations

7/5 and 8/5 major internship systems biology project (due 16/5)

About the course

https://github.com/sdgroeve/Machine_Learning_course_UGent_D012554_2025

About the course

coursera

coursera.org/specializations/machine-learning-introduction

3Blue1Brown

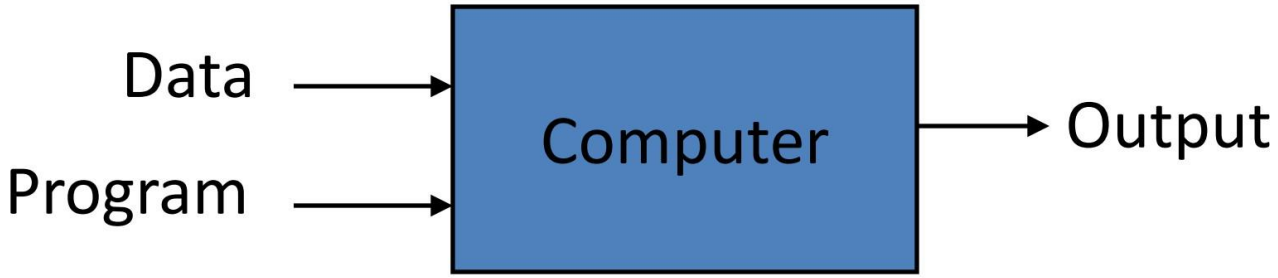
3blue1brown.com

(Classical) Machine Learning

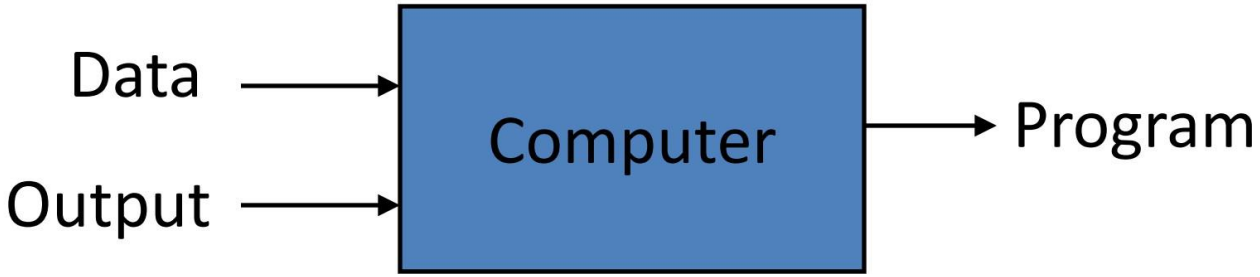
Xkdc

Machine Learning

Traditional Programming



Machine Learning



Slide: Pedro Domingos

Machine Learning

*"A computer program is said to learn from **experience** E with respect to some class of **tasks** T and **performance measure** P, if its performance at tasks in T, as measured by P, improves with experience E."*

-- Tom Mitchell

Machine Learning

*"A computer program is said to learn from **experience** E with respect to some class of **tasks** T and **performance measure** P, if its performance at tasks in T, as measured by P, improves with experience E."*

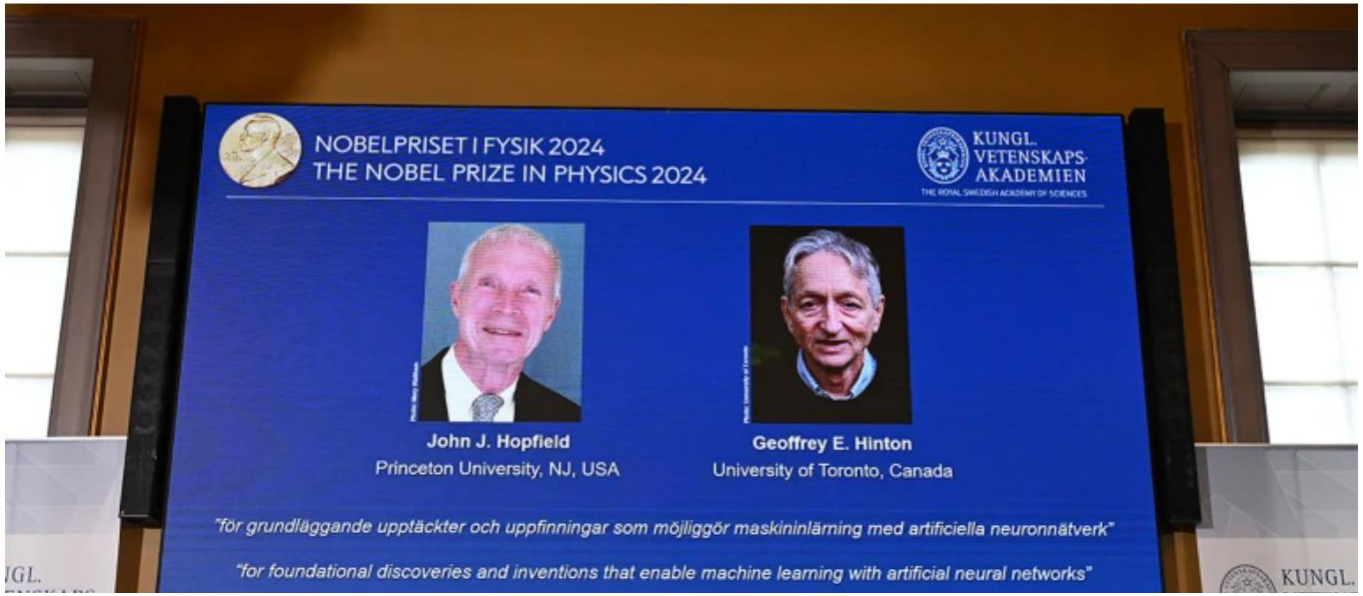
-- Tom Mitchell

NEWS | 08 October 2024

Physics Nobel scooped by machine-learning pioneers

John Hopfield and Geoffrey Hinton pioneered computational methods that enabled the development of neural networks.

By Elizabeth Gibney & Davide Castelvecchi



Biomedical Applications

Medical Imaging Analysis: Machine learning enhances the detection of diseases in medical images (e.g., X-rays, MRIs, CT scans) by identifying patterns indicative of conditions like cancer, fractures, and neurological disorders.

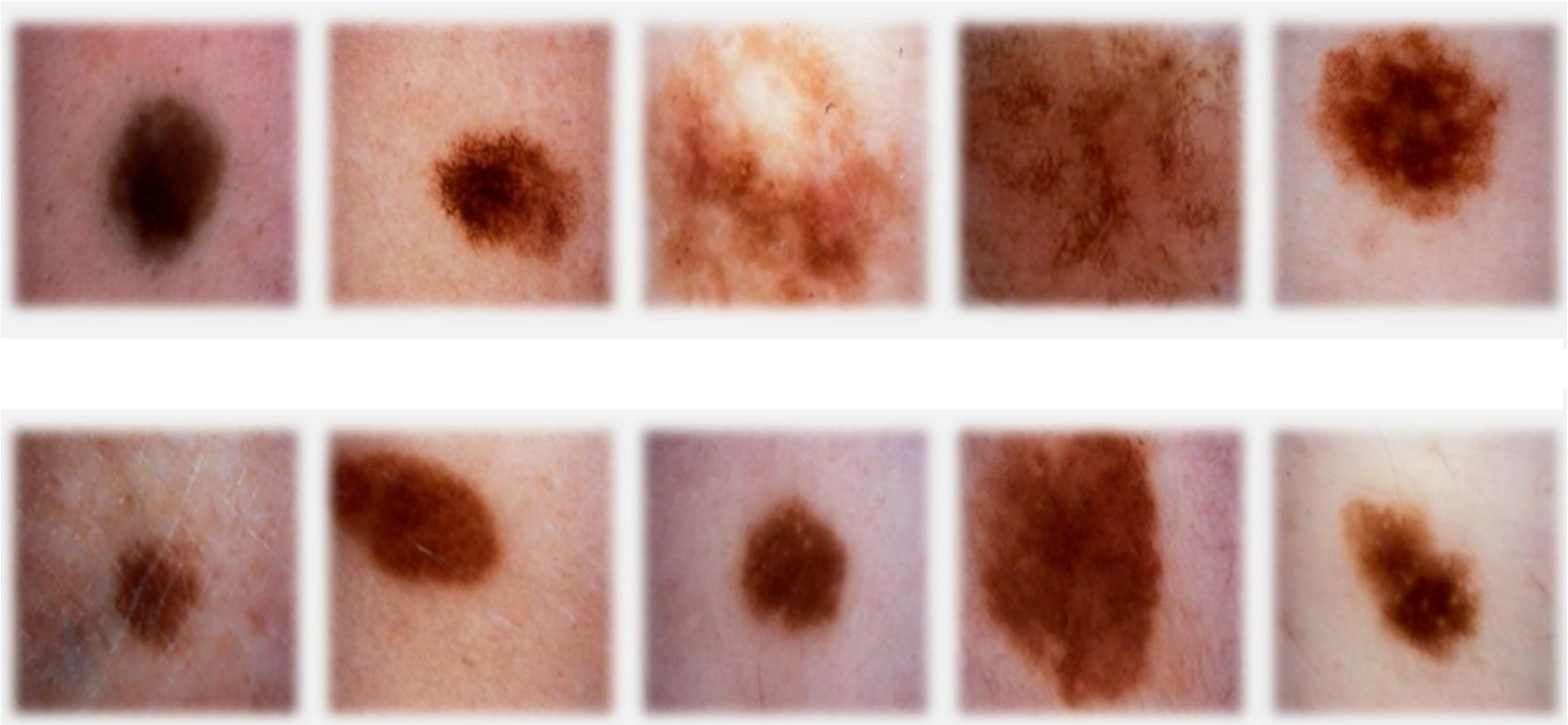
Genomics and Drug Discovery: ML models analyse vast genomic data to identify genetic mutations, predict disease susceptibility, and accelerate drug discovery by simulating molecular interactions.

Personalized Medicine: Machine learning helps tailor treatments based on an individual's genetic profile, medical history, and response to previous treatments, improving patient outcomes.

Predictive Diagnostics: AI models analyse patient data (e.g., electronic health records, biomarkers) to predict disease onset, progression, and potential complications, enabling early intervention.

Healthcare Automation: ML-powered chatbots, virtual assistants, and administrative tools streamline tasks like patient triaging, medical documentation, and hospital resource allocation.

Classification

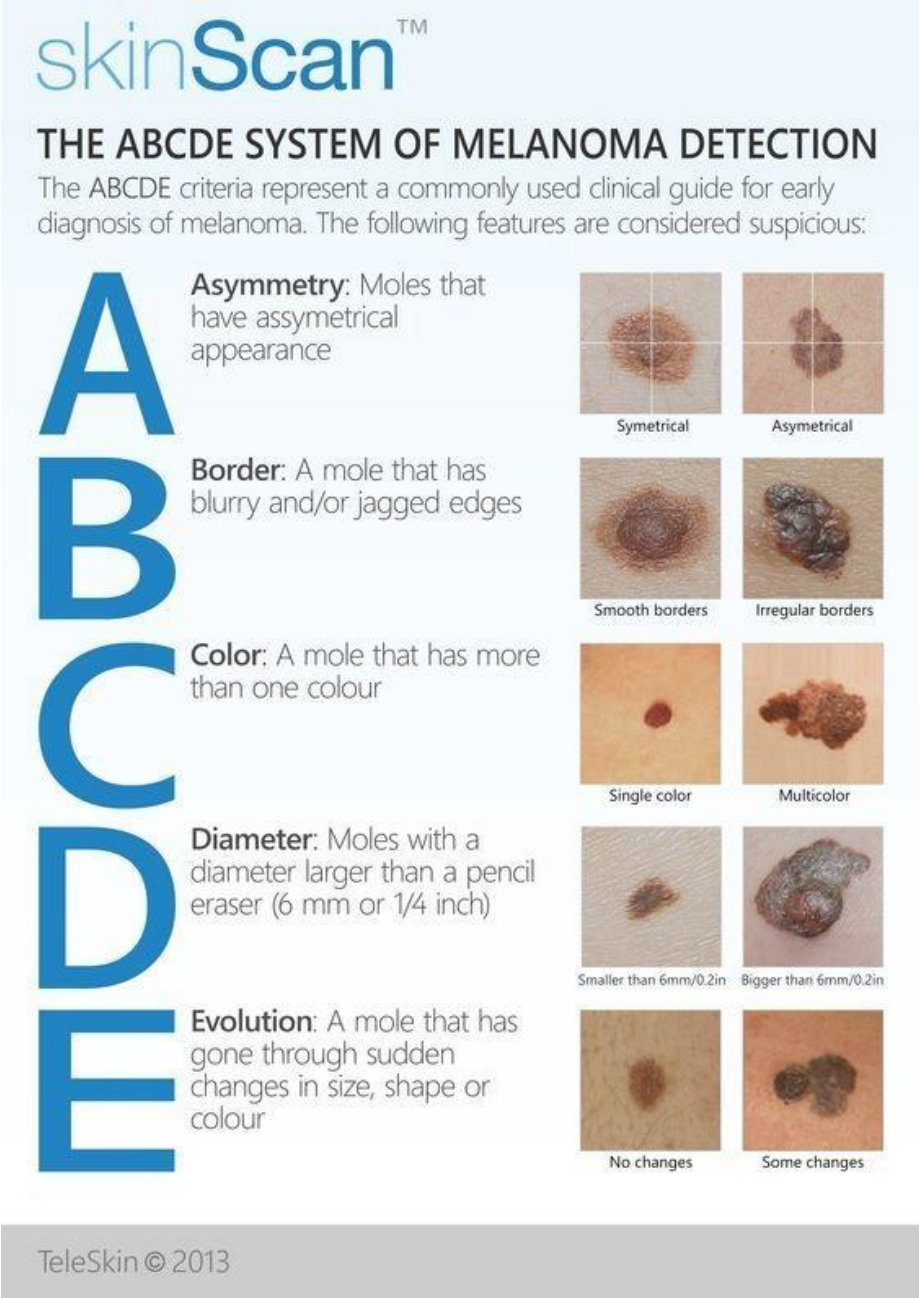


sign of cancer

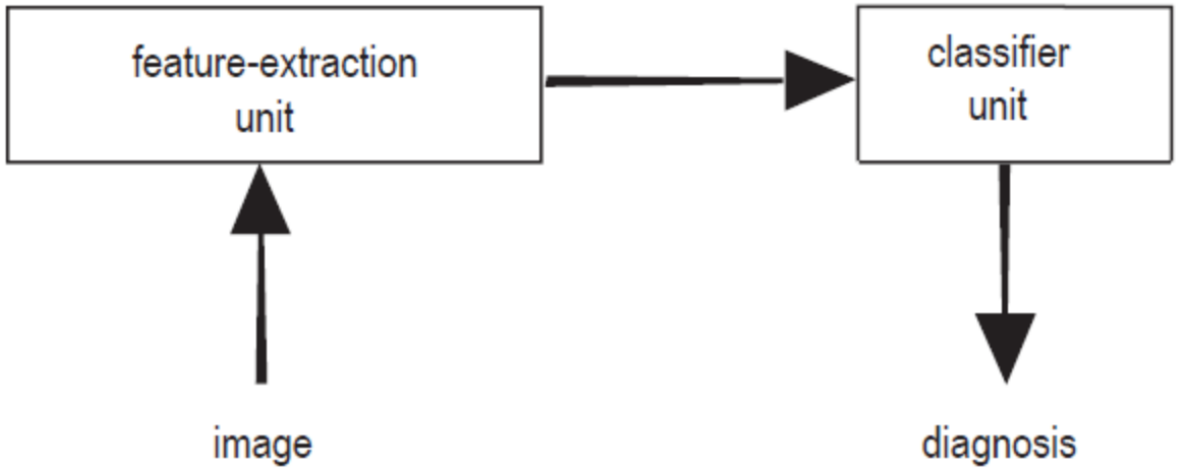
top row malignant

bottom row benign

Classification



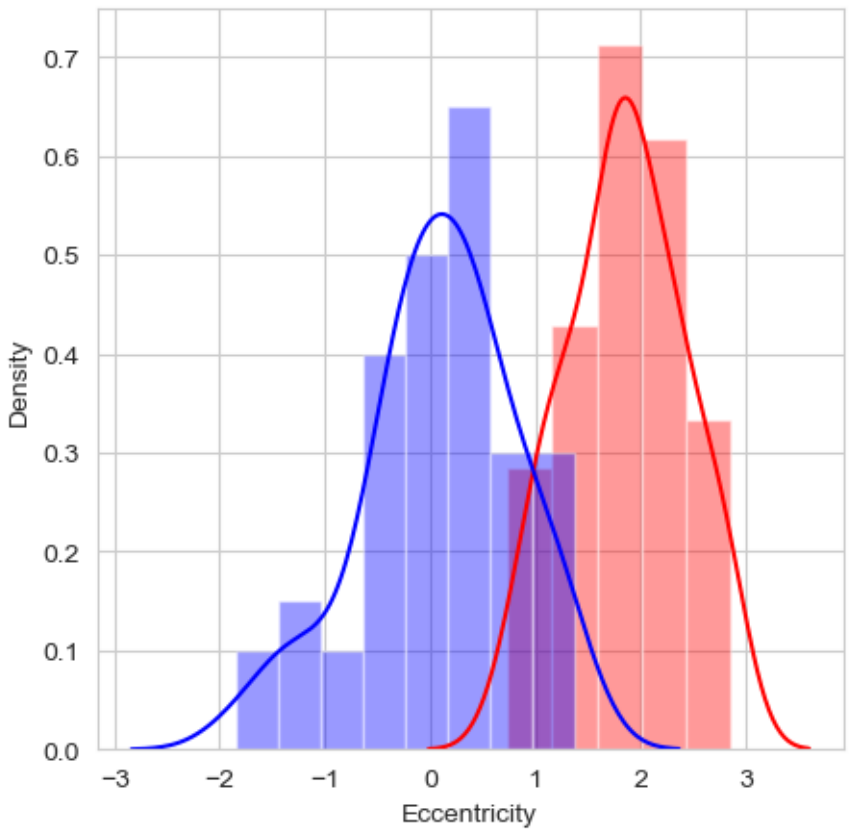
Classification: terminology



feature extraction: features (a.k.a. properties or attributes)

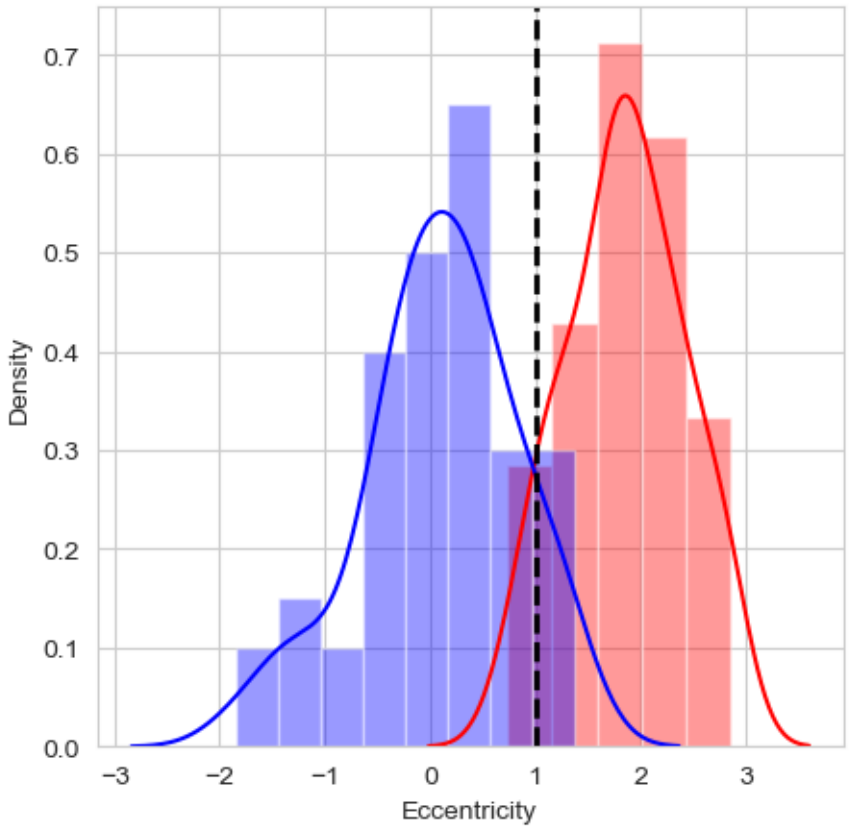
data set, sample (a.k.a. example, instance or data point), label (a.k.a. target)

Classification: a feature



feature: eccentricity of lesion (how nearly circular the lesion is)

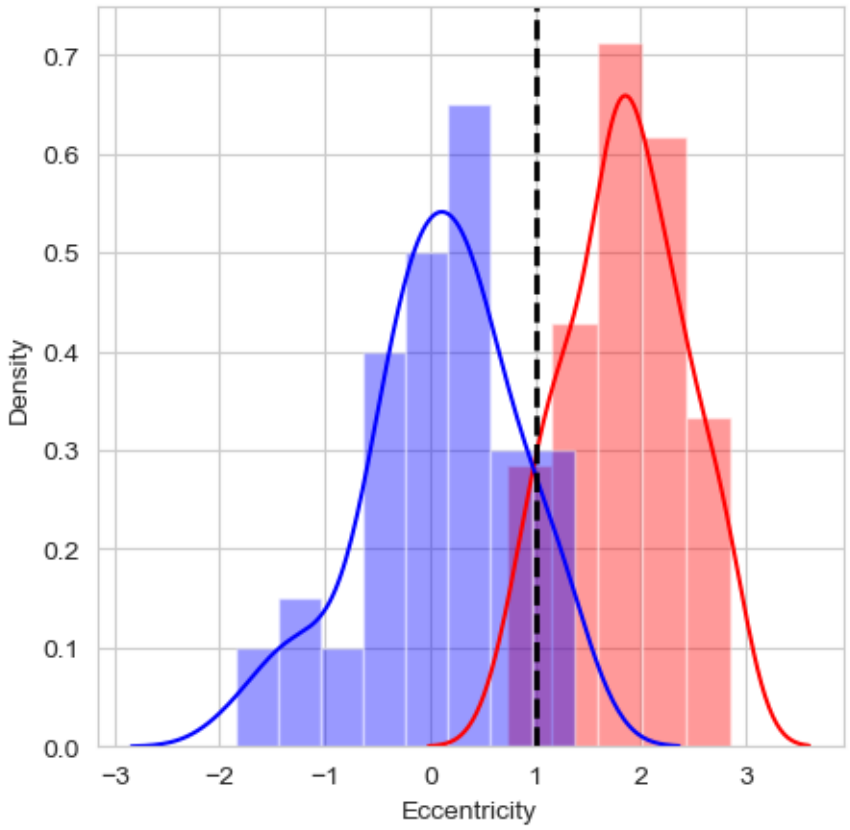
Classification: the model

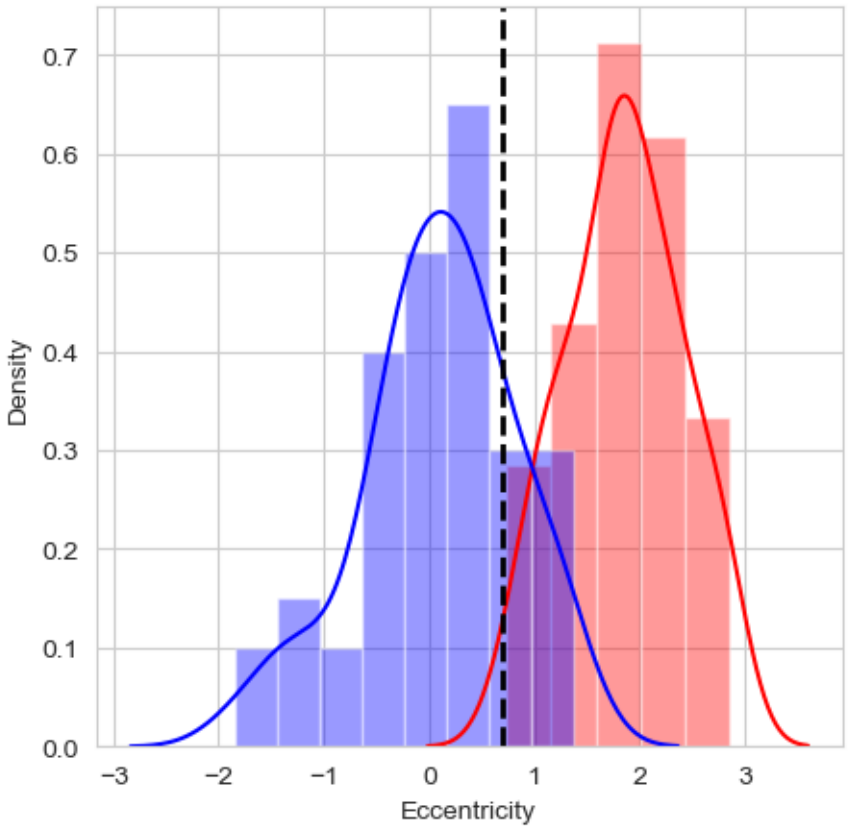


feature: eccentricity of lesion (how nearly circular the lesion is)

model: threshold ***t***

Classification: the model





feature: eccentricity of lesion (how nearly circular the lesion is)

model: threshold ***t:*** consequence of the predictions

Classification: prediction errors

malignant: **positive** class
benign: **negative** class

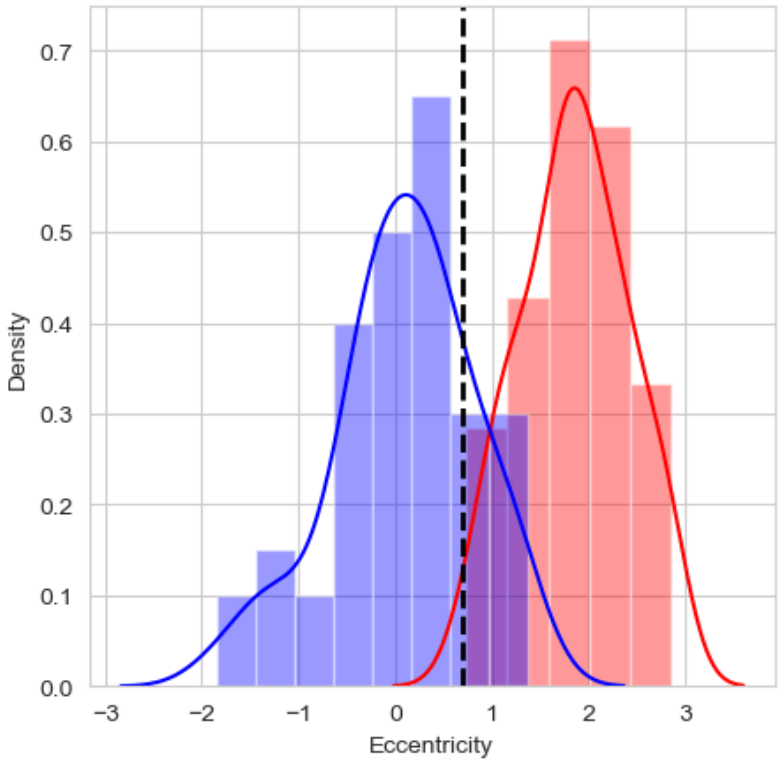
count the number of malignant images with eccentricity value $\geq t$: **true positive** predictions (TP)

count the number of malignant images with eccentricity value $< t$: **false negative** predictions (FN)

count the number of benign images with eccentricity value $\geq t$: **false positive** predictions (FP)

count the number of benign images with eccentricity value $< t$: **true negative** predictions (TN)

	Classified as: Positive	Classified as: Negative
Really is: Positive	True Positive	False Negative
Really is: Negative	False Positive	True Negative



Classification: prediction errors

	Classified as: Positive	Classified as: Negative
Really is: Positive	True Positive	False Negative
Really is: Negative	False Positive	True Negative

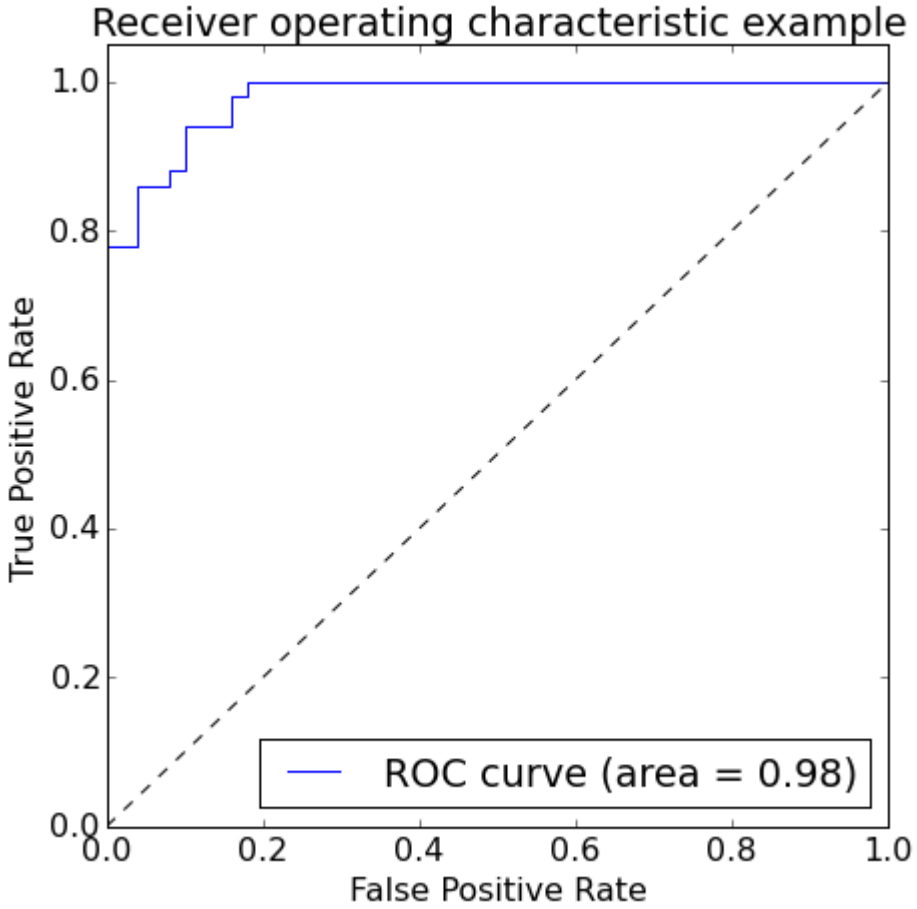
$$\text{accuracy} = \frac{TP + TN}{TP + FP + TN + FN}$$

$$\text{TPR} = \frac{TP}{TP + FN}$$

$$\text{FPR} = \frac{FP}{FP + TN}$$

Classification: prediction errors

$$\mathrm{TPR} = \frac{TP}{TP + FN}$$



$$\mathrm{FPR} = \frac{FP}{FP + TN}$$

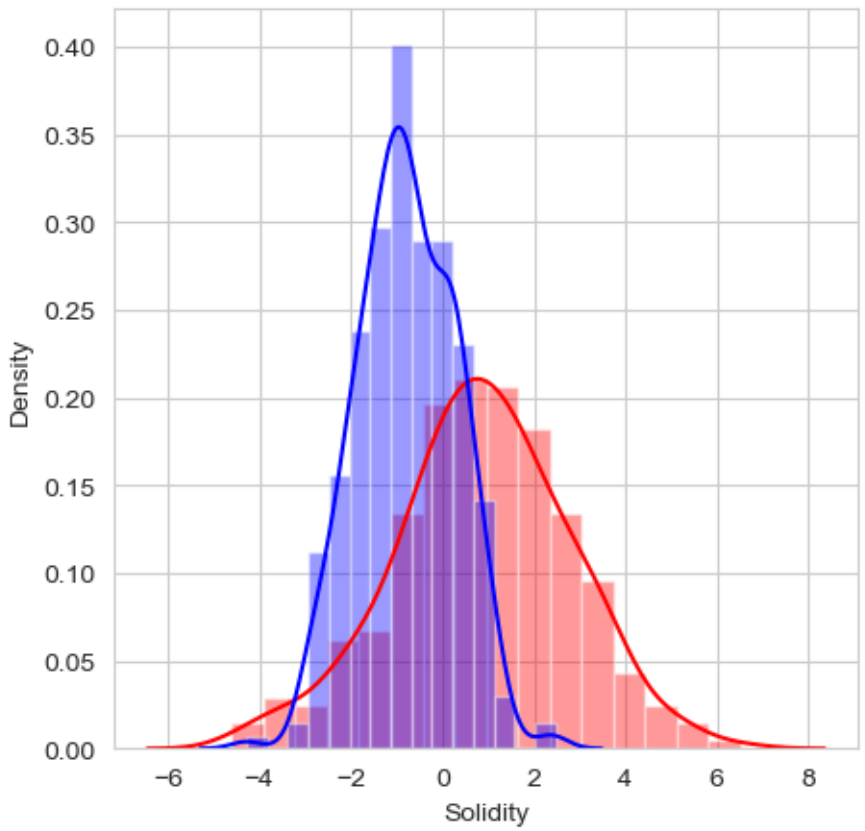
model that classifies all images as malignant:
TPR=1 and FPR=1

model that classifies all images a benign:
TPR=0 and FPR=0

vary threshold *t*

Area Under the Curve (AUC)

Classification: multi-dimensional

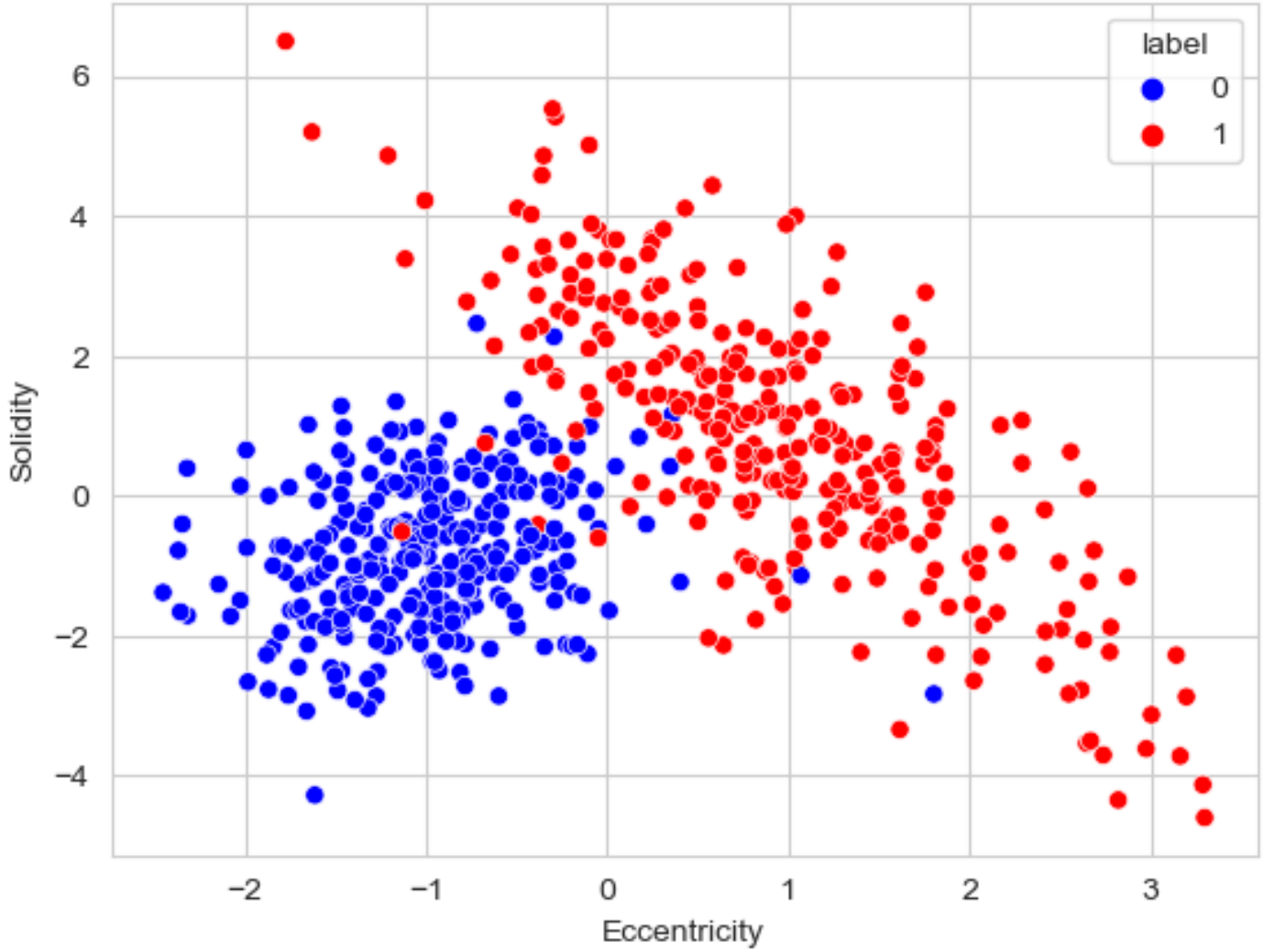


add another feature?

feature vector X

Euclidean vector space

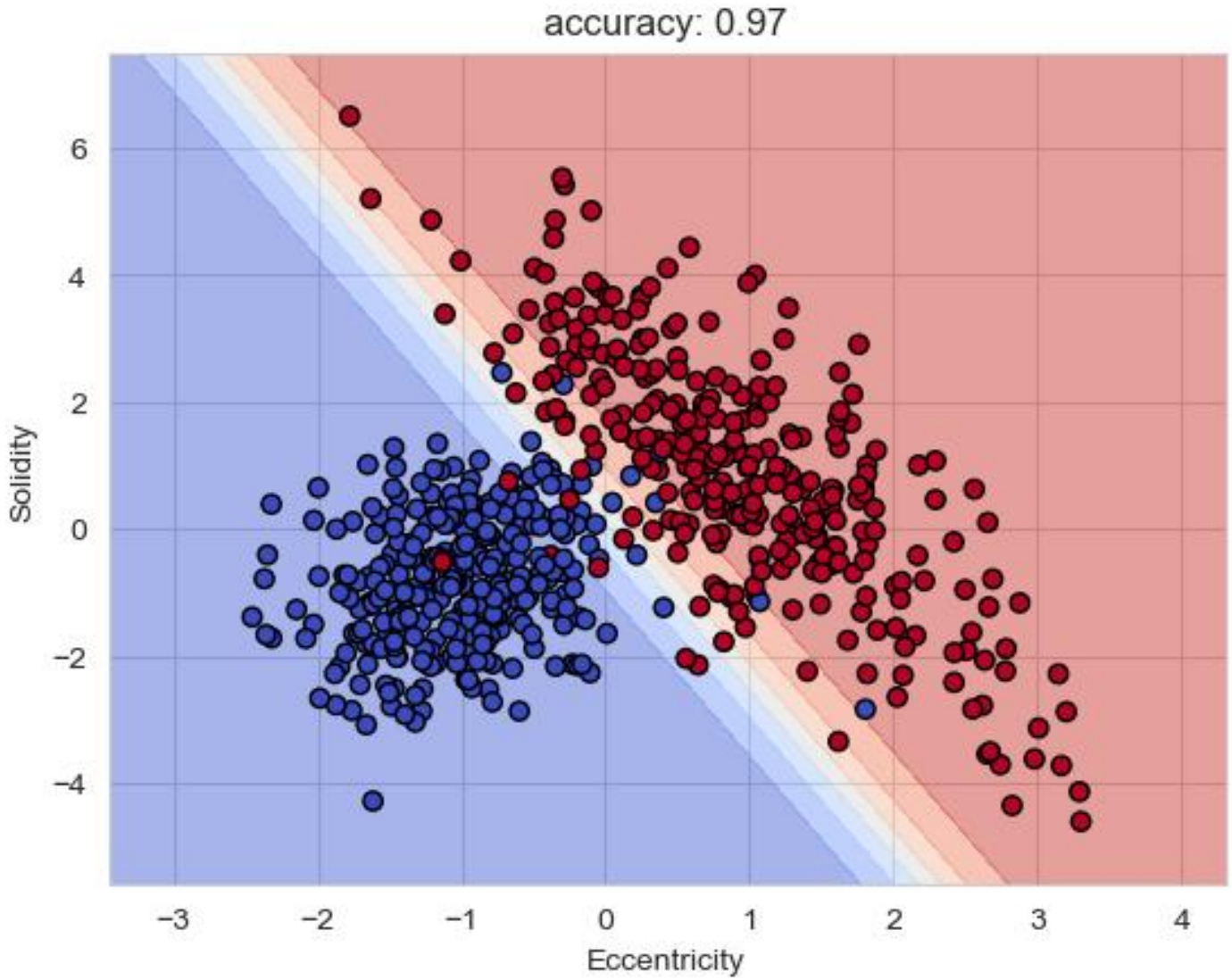
Classification: multi-dimensional



feature vector X

Euclidean vector space

Classification: multi-dimensional



linear decision boundary

blue region malignant, red region benign

yet more features

can’t look at the decision boundary

more complex

Decision tree

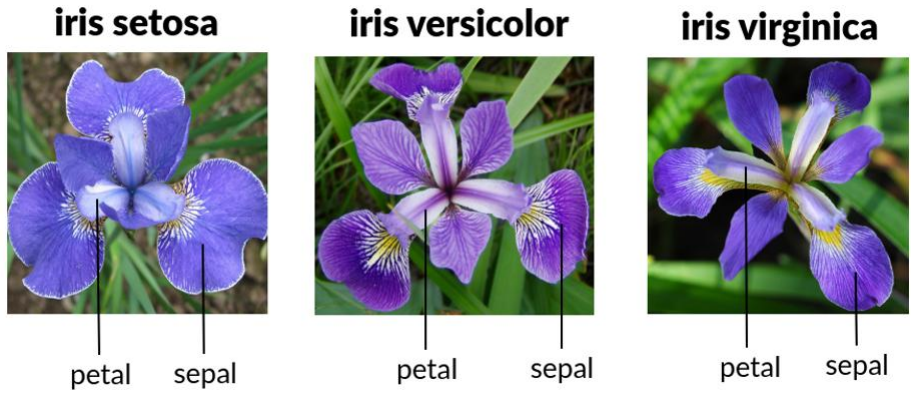
A **decision tree** is a supervised machine learning algorithm used for classification and regression tasks. It models decisions and their possible consequences in a tree-like structure, where nodes represent features, branches represent decision rules, and leaves represent outcomes.

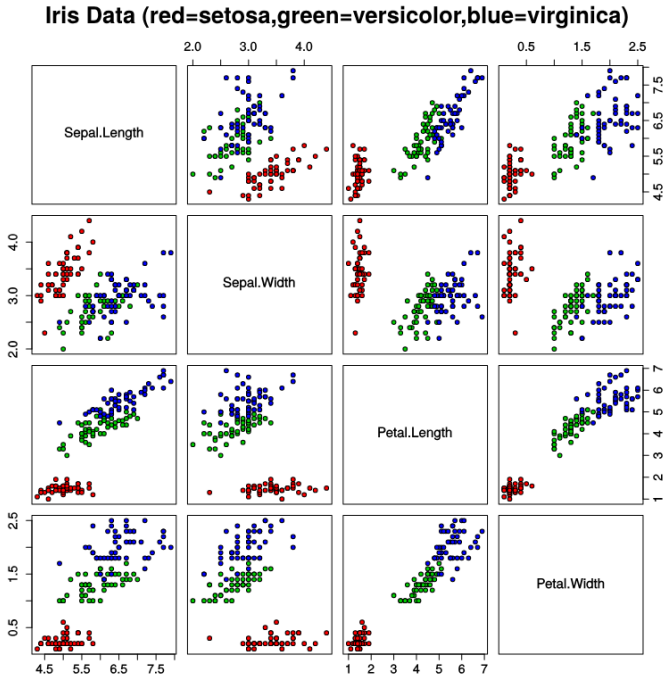
Decision trees **split data based on feature values**, using criteria like Gini impurity or entropy (information gain) in classification, and mean squared error (MSE) in regression. They recursively divide the dataset into subsets to make predictions.

They are **interpretable** and easy to understand but prone to **overfitting**, which can be mitigated by techniques like pruning or max depth limitation.

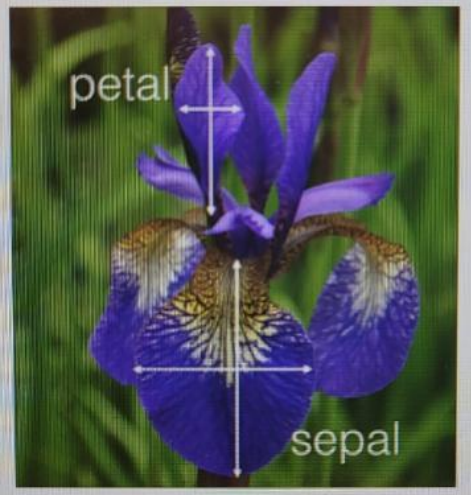
The concept of decision trees dates back to the 1960s, with ID3 being a notable early algorithm introduced by Ross Quinlan in 1986.

Iris dataset





4. In 1936 Ronald Fisher published a famous paper on classification titled "The use of multiple measurements in taxonomic problems." In the paper, Fisher study the problem of classifying iris flowers based on measurements of the sepal and petal widths and lengths, depicted in the image below.



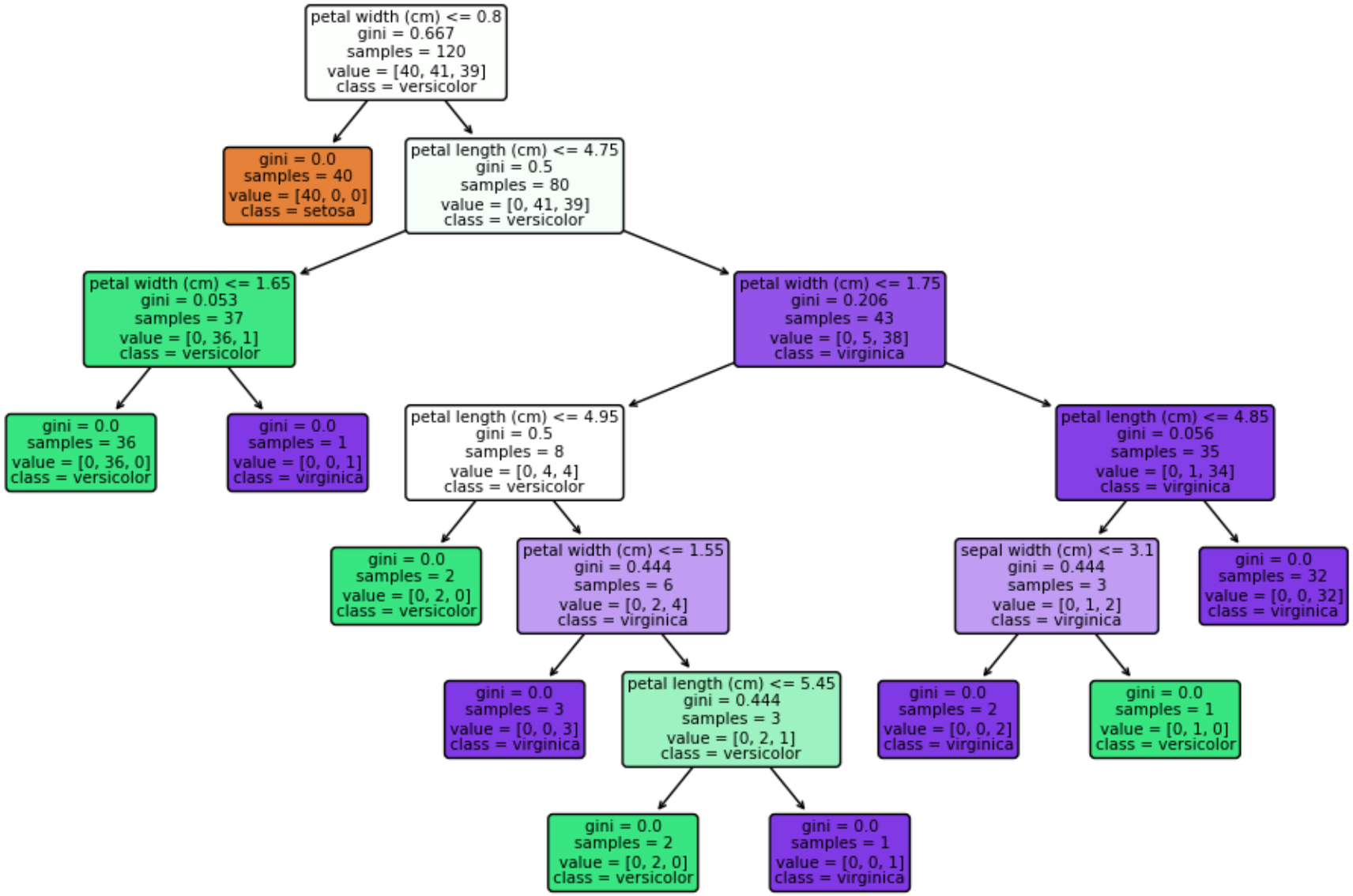
Fisher's dataset is available in Matlab (`fisheriris.mat`) and is widely available on the web (e.g., Wikipedia). The dataset consists of 50 examples of three types of iris flowers. The sepal and petal measurements can be used to classify the examples into the three types of flowers. Approach the iris classification problem using a *generative modeling* approach. Assume that each class-conditional density is MVN.

(a) Fit the MVN models to the training data using estimates of means and covariances. Design a classifier based on these fitted models.

(b) Constrain the MVN models so that the decision boundaries between pairs of classes are linear.

Fisher, R.A. (1936) The Use of Multiple Measurements in Taxonomic Problems. Annals Eugenics, 7, 179-188.

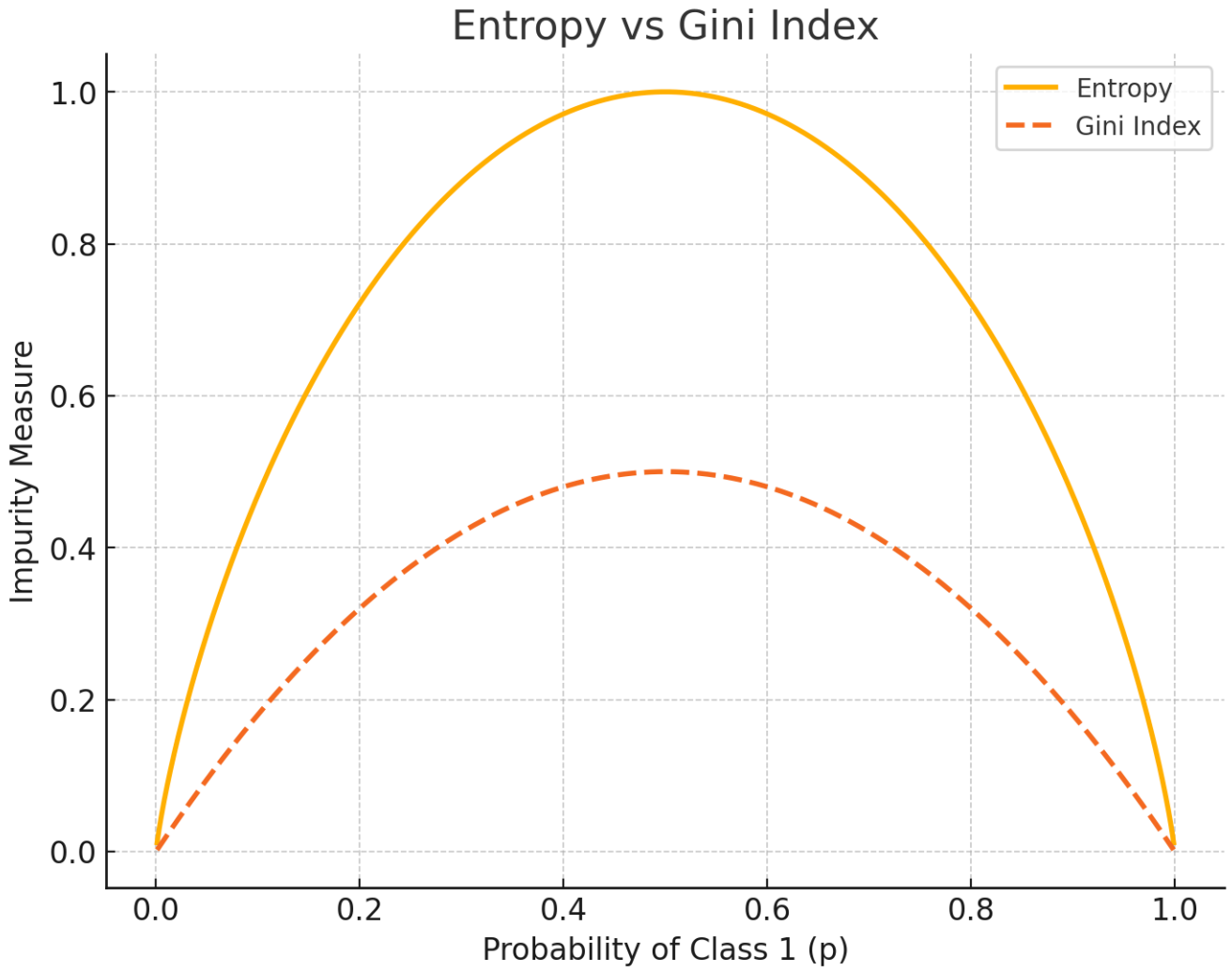
Decision tree



Decision tree

$$\text{Entropy} = -\sum_{i=1}^{C} p_i \log_2(p_i)$$

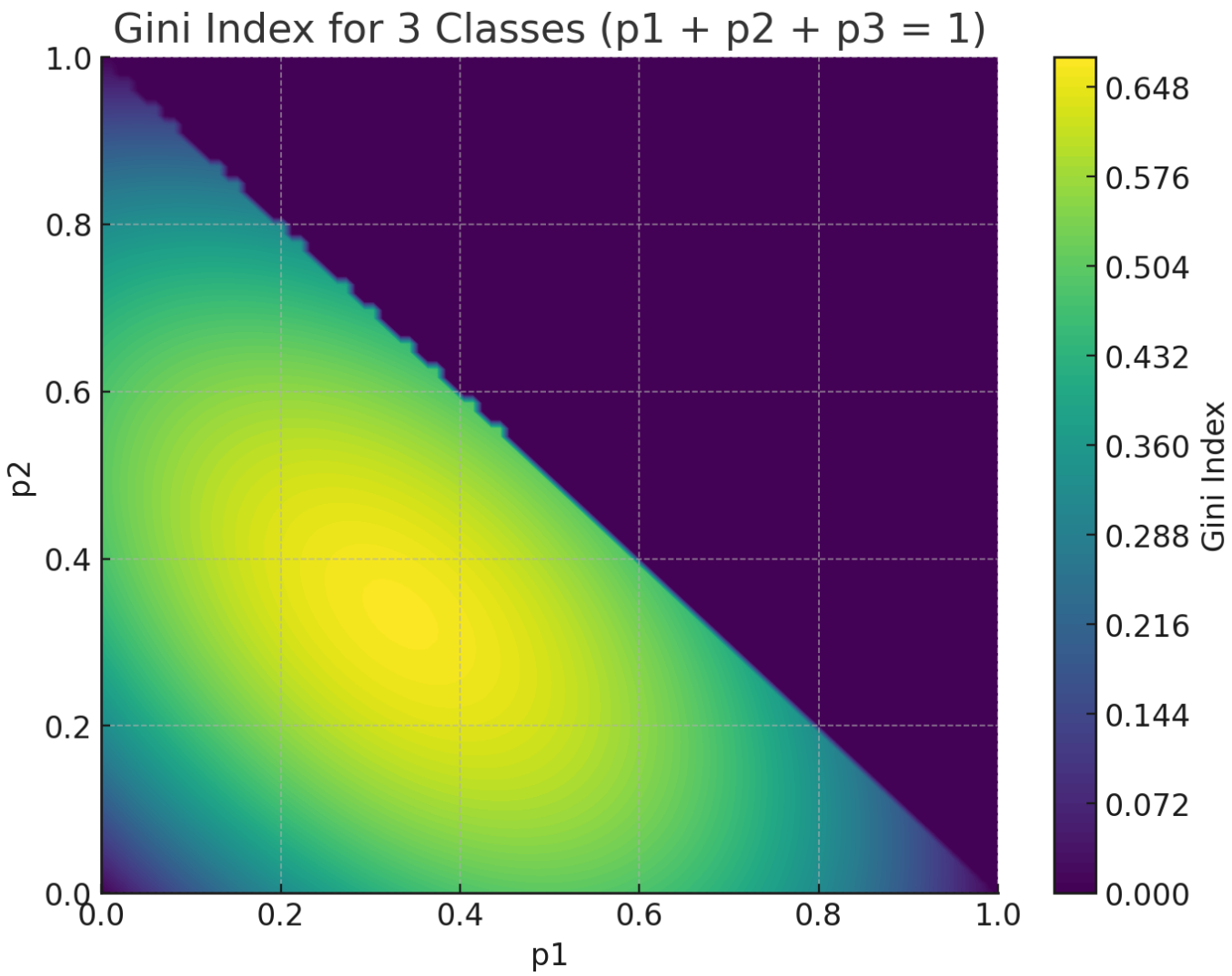
$$\text{Gini Index} = 1 - \sum_{i=1}^{C} p_i^2$$



Decision tree

$$\text{Entropy} = -\sum_{i=1}^{C} p_i \log_2(p_i)$$

$$\text{Gini Index} = 1 - \sum_{i=1}^{C} p_i^2$$

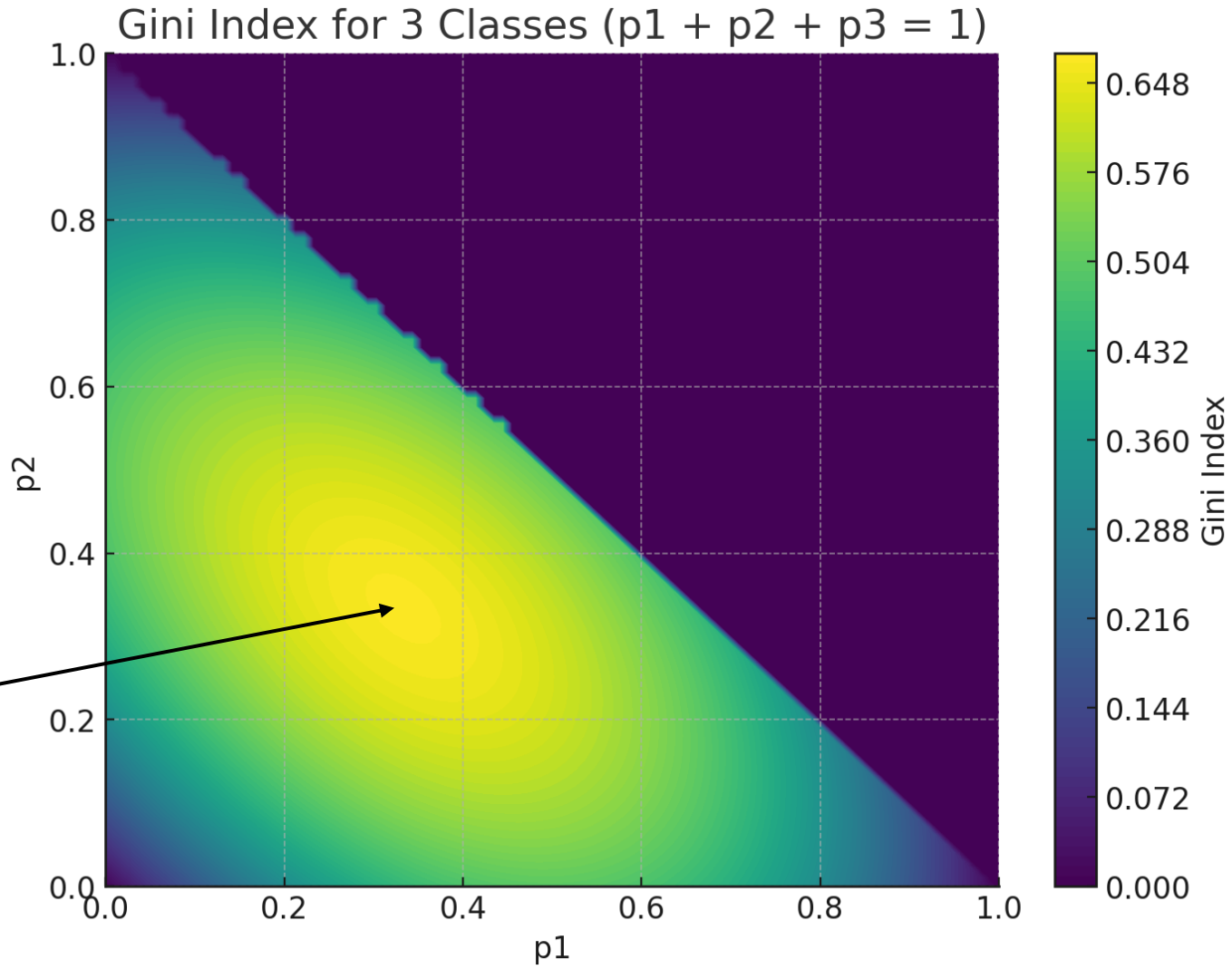


Decision tree

$$\text{Entropy} = -\sum_{i=1}^{C} p_i \log_2(p_i)$$

$$\text{Gini Index} = 1 - \sum_{i=1}^{C} p_i^2$$

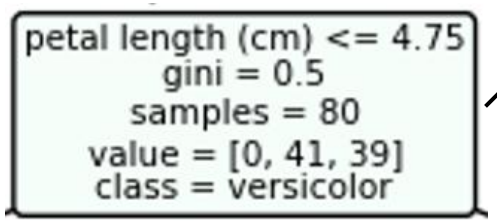
petal width (cm) <= 0.8
gini = 0.667
samples = 120
value = [40, 41, 39]
class = versicolor

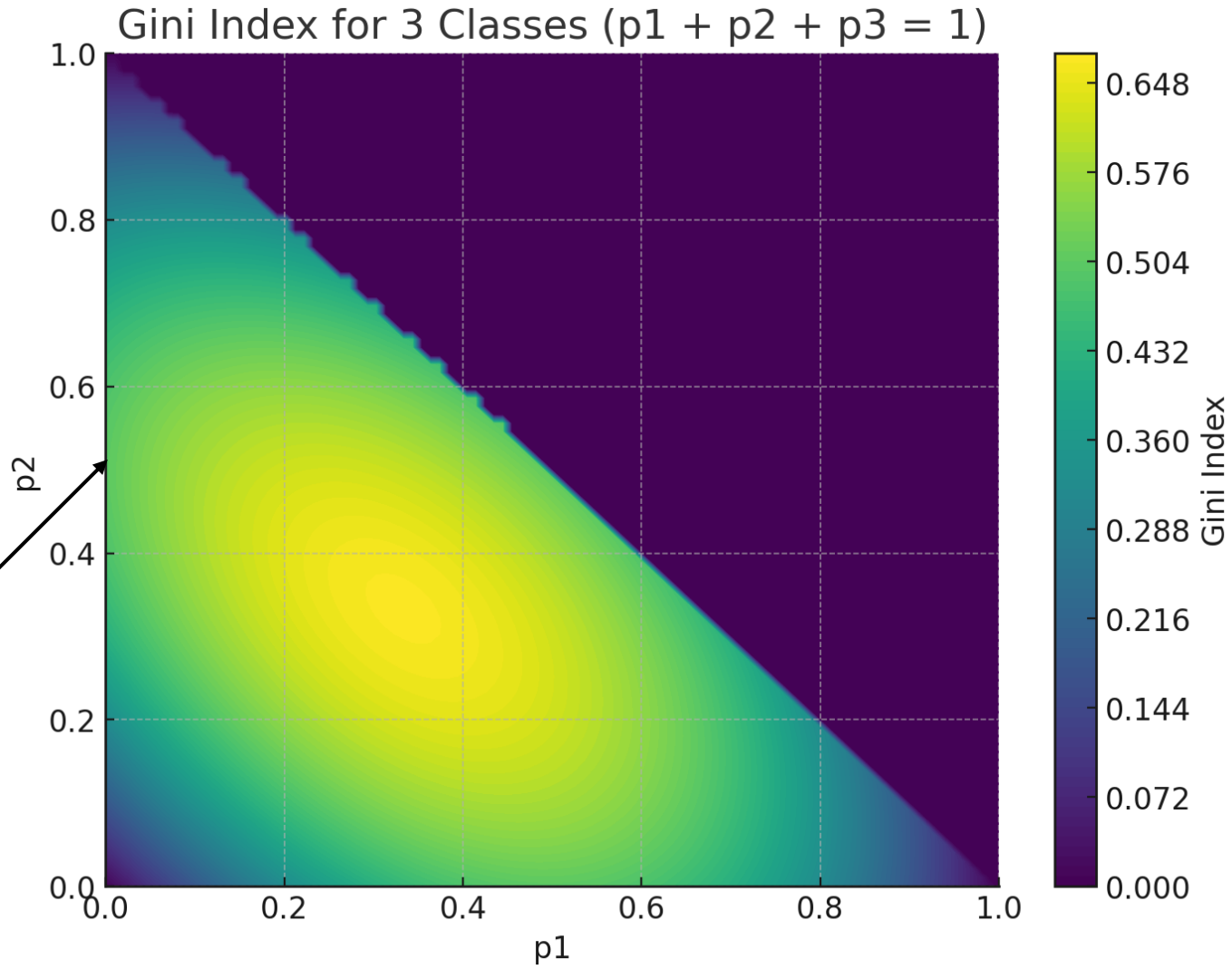


Decision tree

$$\text{Entropy} = -\sum_{i=1}^{C} p_i \log_2(p_i)$$

$$\text{Gini Index} = 1 - \sum_{i=1}^{C} p_i^2$$

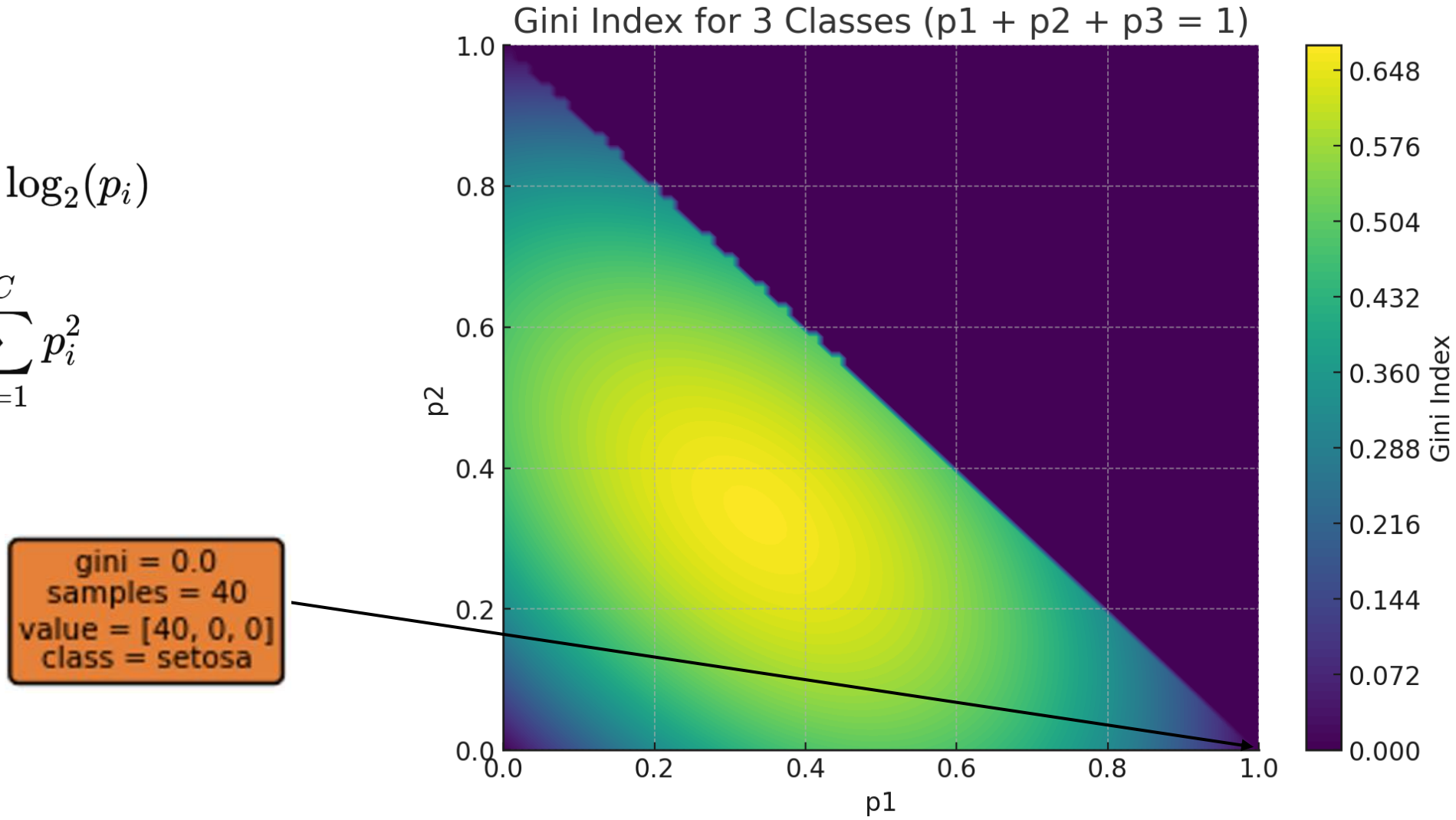




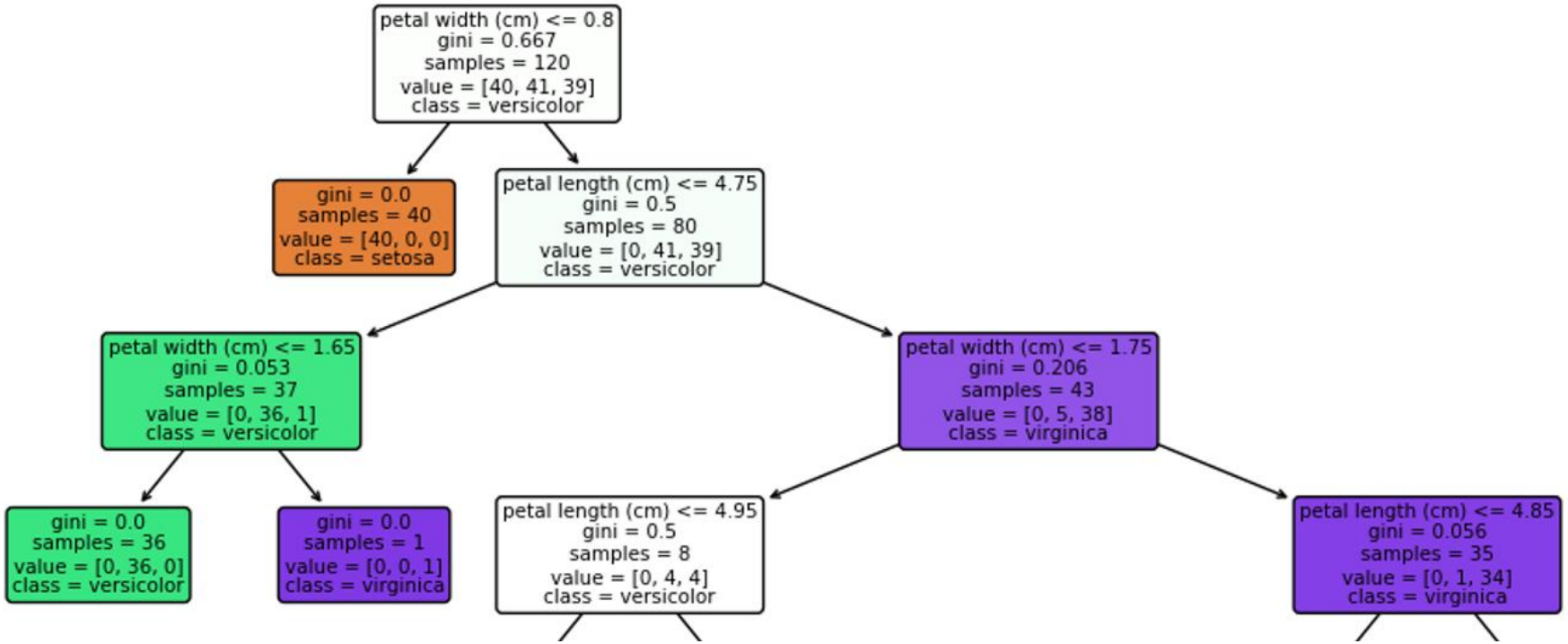
Decision tree

$$\text{Entropy} = -\sum_{i=1}^{C} p_i \log_2(p_i)$$

$$\text{Gini Index} = 1 - \sum_{i=1}^{C} p_i^2$$



Decision tree



For a dataset D split into two subsets D_{left} and D_{right}, the Gini criterion is:

$$\text{Gini}_{\text{split}} = \frac{|D_{\text{left}}|}{|D|} \cdot \text{Gini}(D_{\text{left}}) + \frac{|D_{\text{right}}|}{|D|} \cdot \text{Gini}(D_{\text{right}})$$

Model evaluation: train-test split

By separating data into **training** and **testing** sets, the model learns patterns from training data while being evaluated on **unseen data**, ensuring it generalizes well to new inputs.

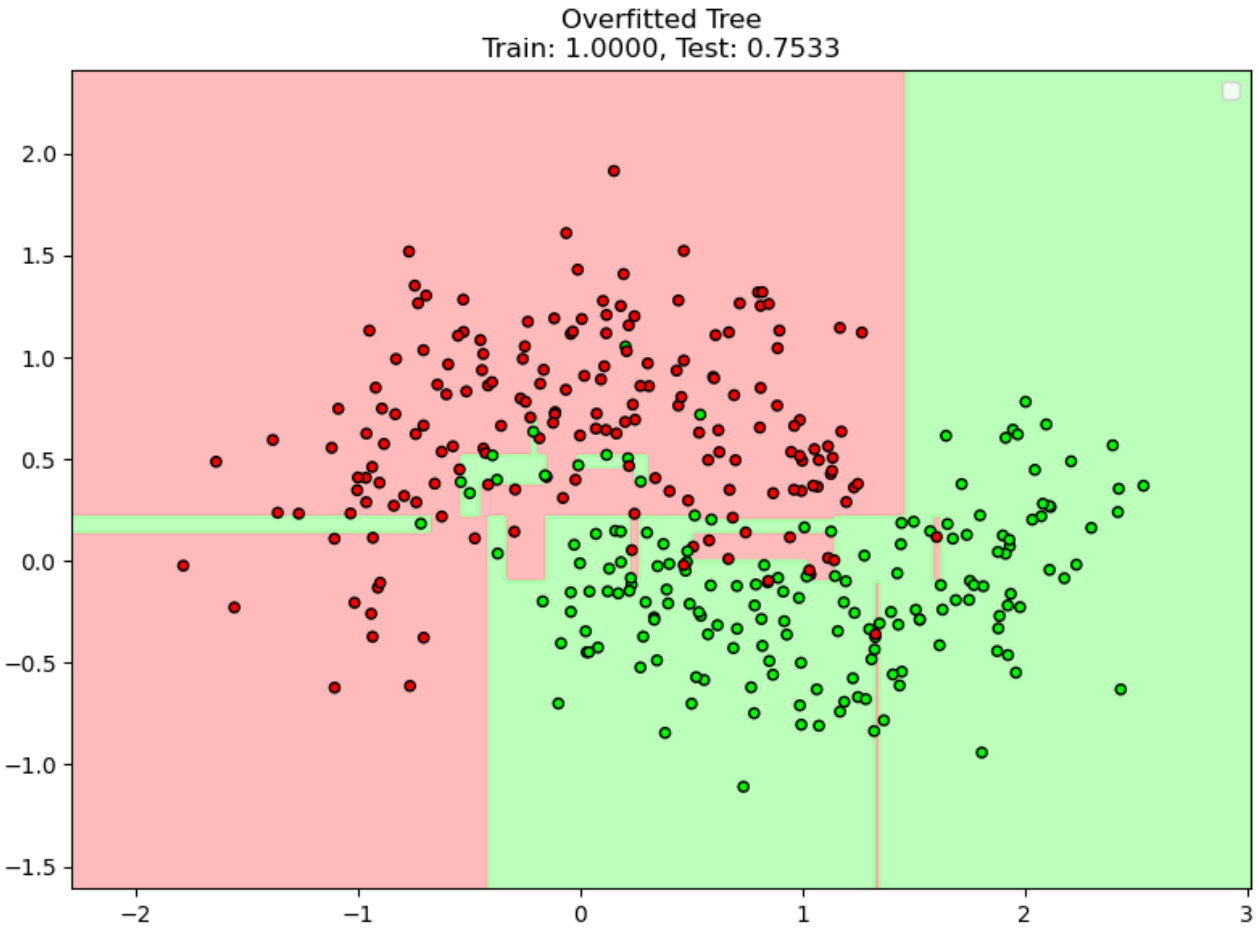
The test set provides an **independent dataset** to measure model performance, helping to assess accuracy, precision, recall, and other metrics objectively.

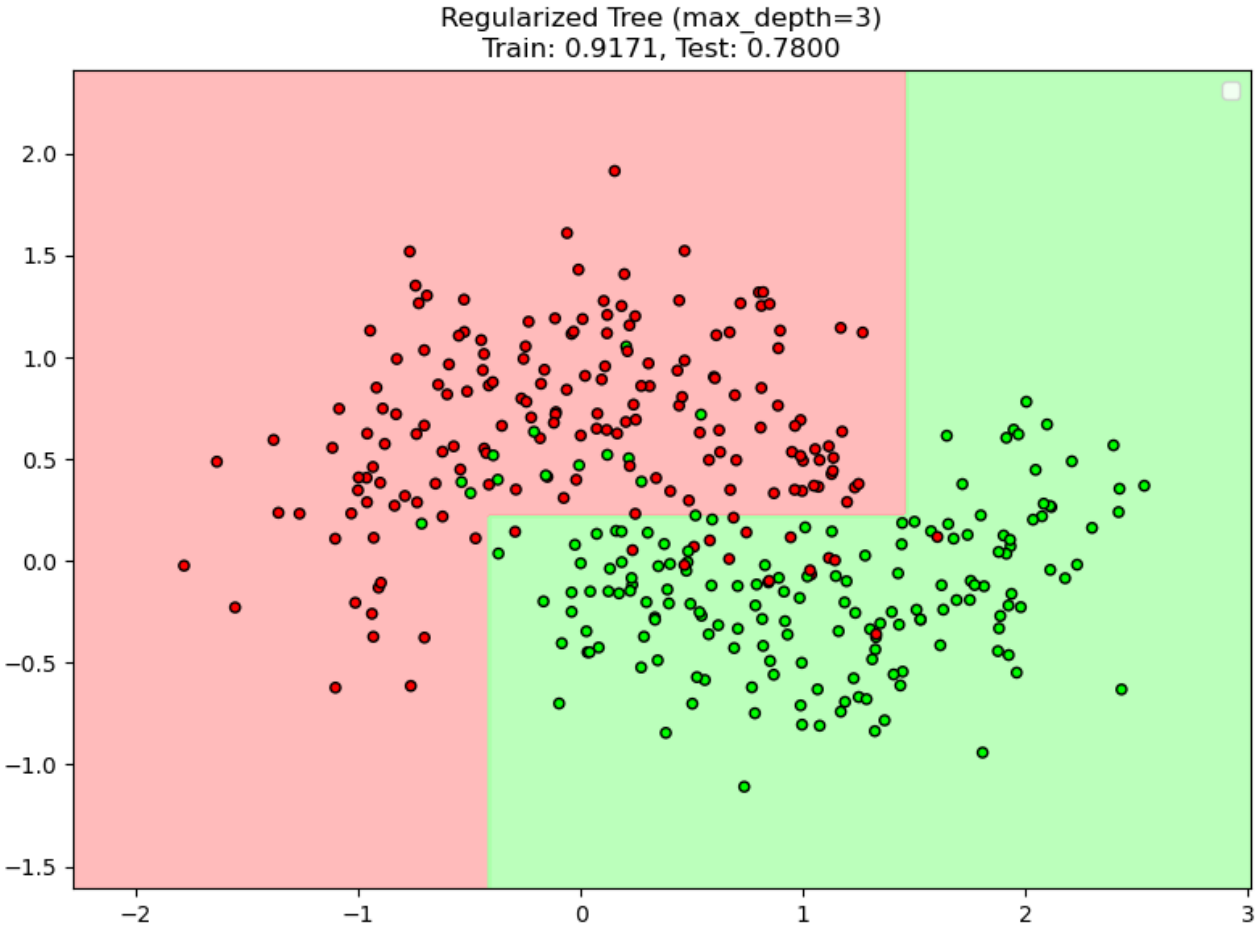
Model evaluation: train-test split

GitHub

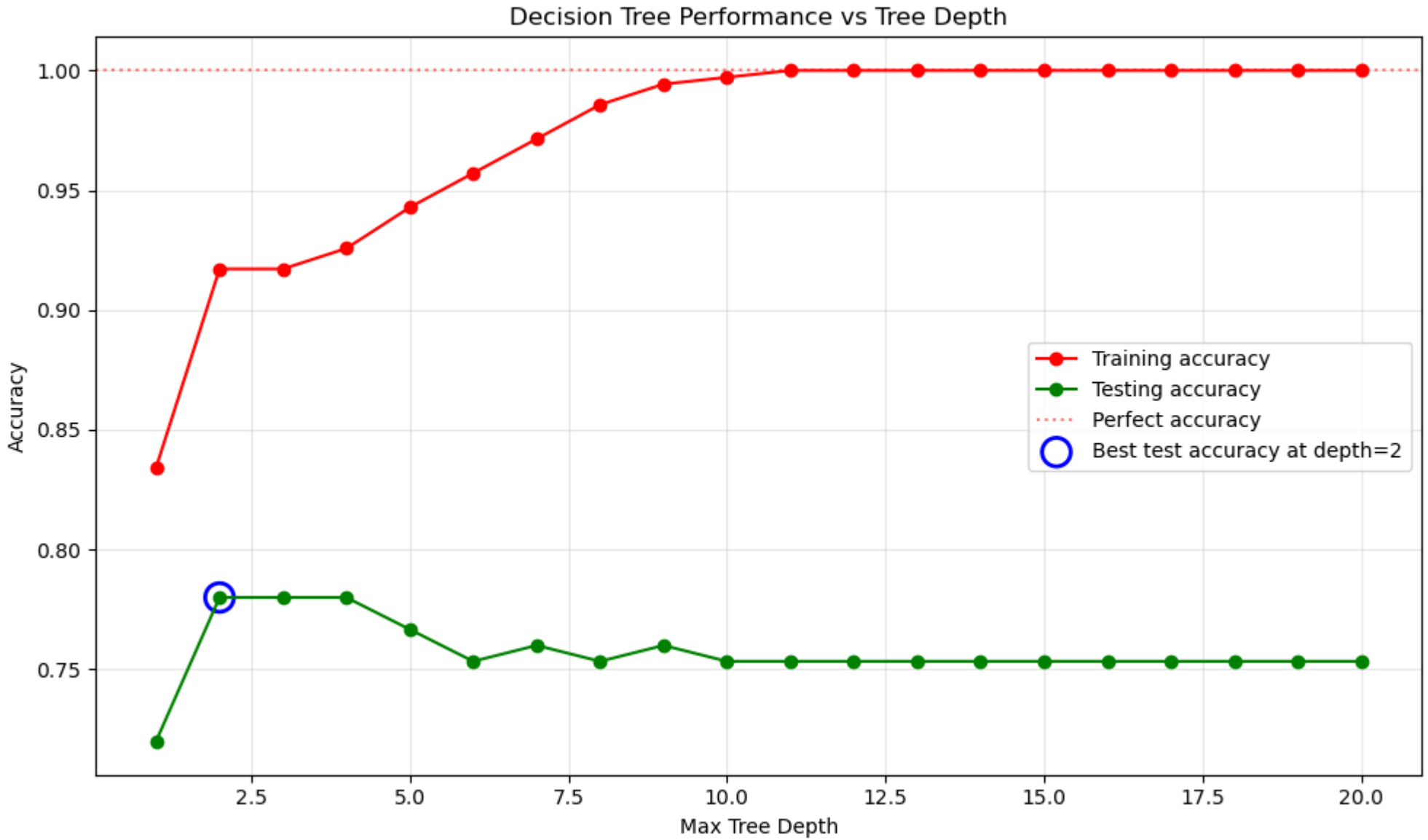
overfitting.ipynb

Classification: model complexity

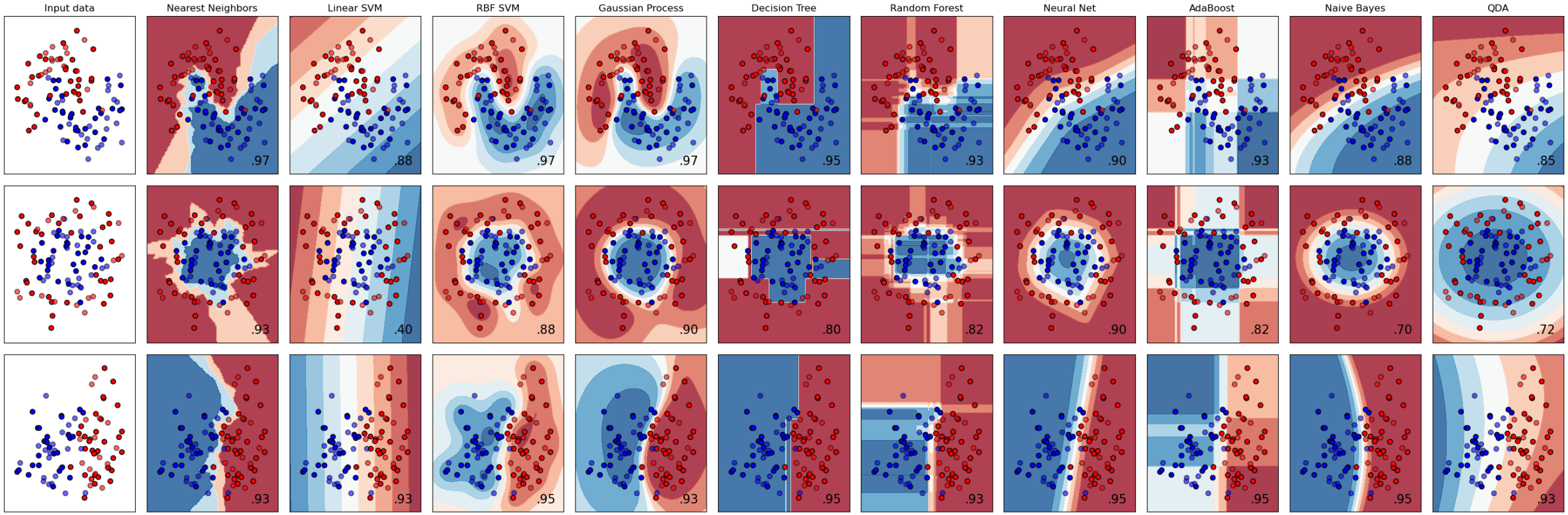




Classification: model complexity



Scikit-learn



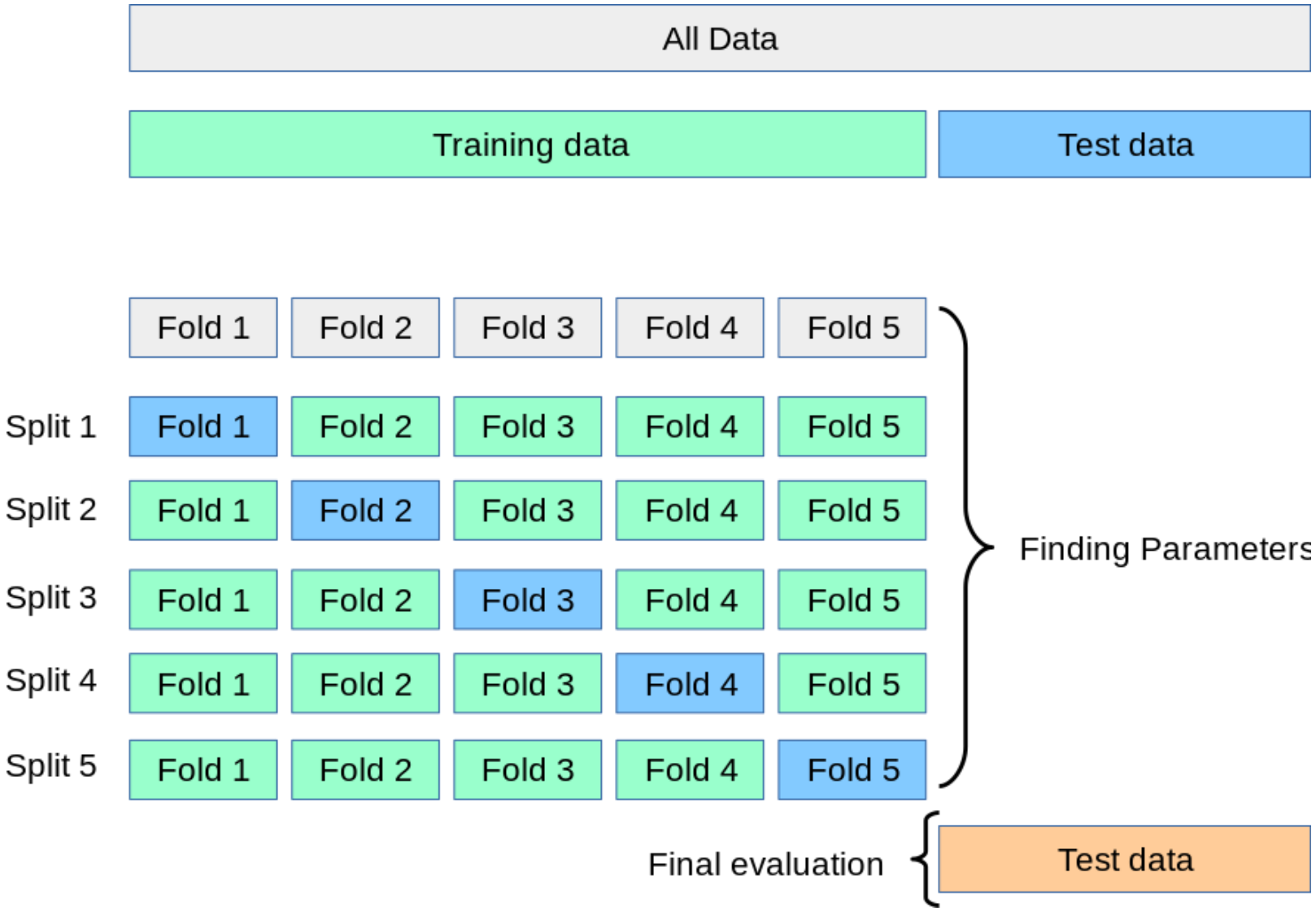
Model evaluation: *k*-fold cross-validation

Cross-validation is a technique to evaluate model performance by splitting the dataset into multiple training and validation sets, reducing bias from a single train-test split.

K-fold cross-validation is the most common method, where the dataset is divided into k folds, and the model is trained and validated k times, with each fold serving as a validation set once.

It helps prevent overfitting and improves model selection, especially in hyperparameter tuning, by providing a more reliable estimate of generalization performance.

Model evaluation: *k*-fold cross-validation



Data normalization

Without normalization, features with larger numerical ranges can dominate model training, leading to biased weight updates.

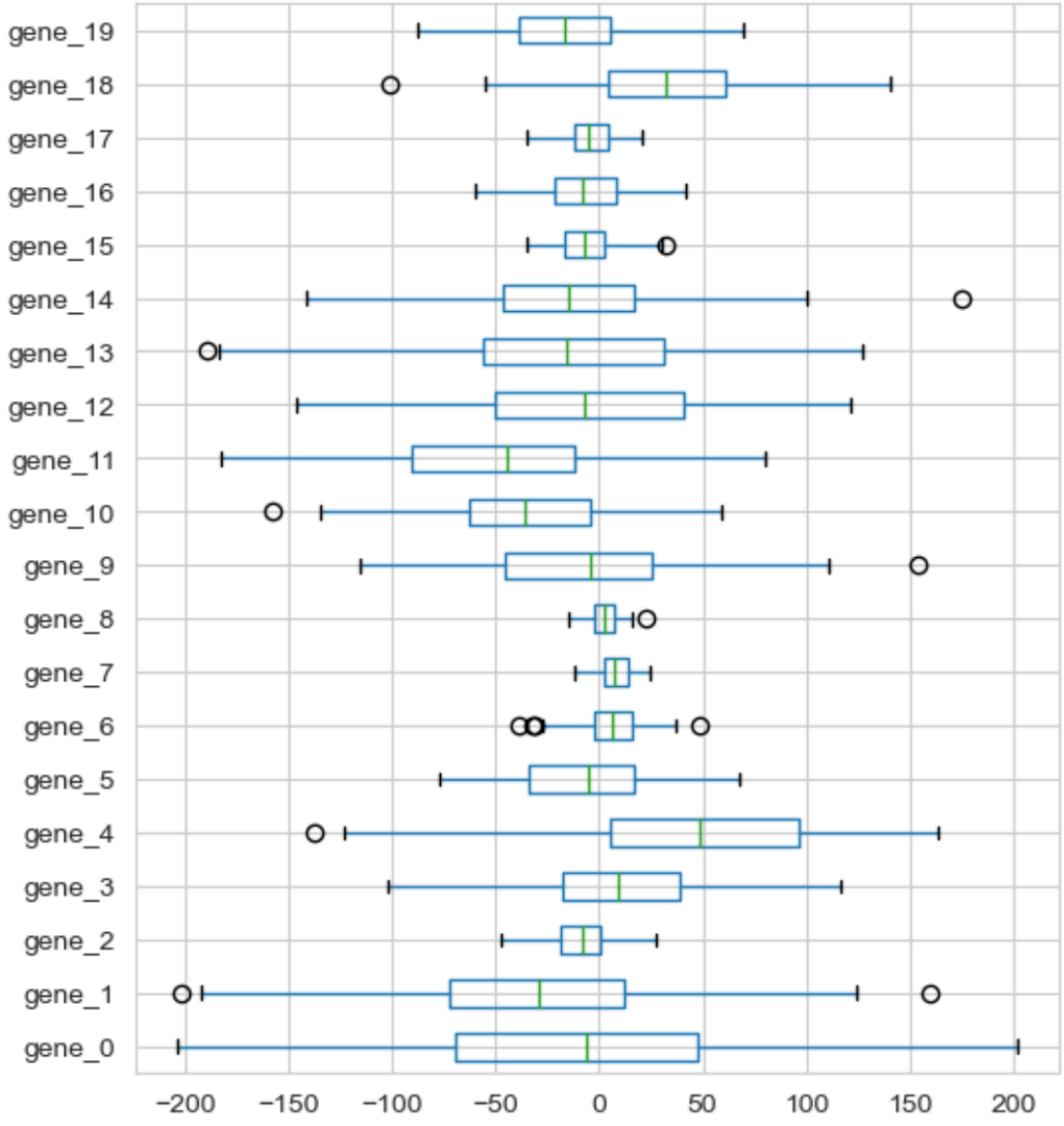
Normalization helps optimization algorithms like gradient descent converge faster by ensuring consistent feature magnitudes, reducing the risk of slow or unstable learning.

Methods include Min-Max Scaling and Standardization, where Min-Max Scaling rescales values within a fixed range, and Standardization (Z-score normalization) ensures features have a mean of 0 and standard deviation of 1

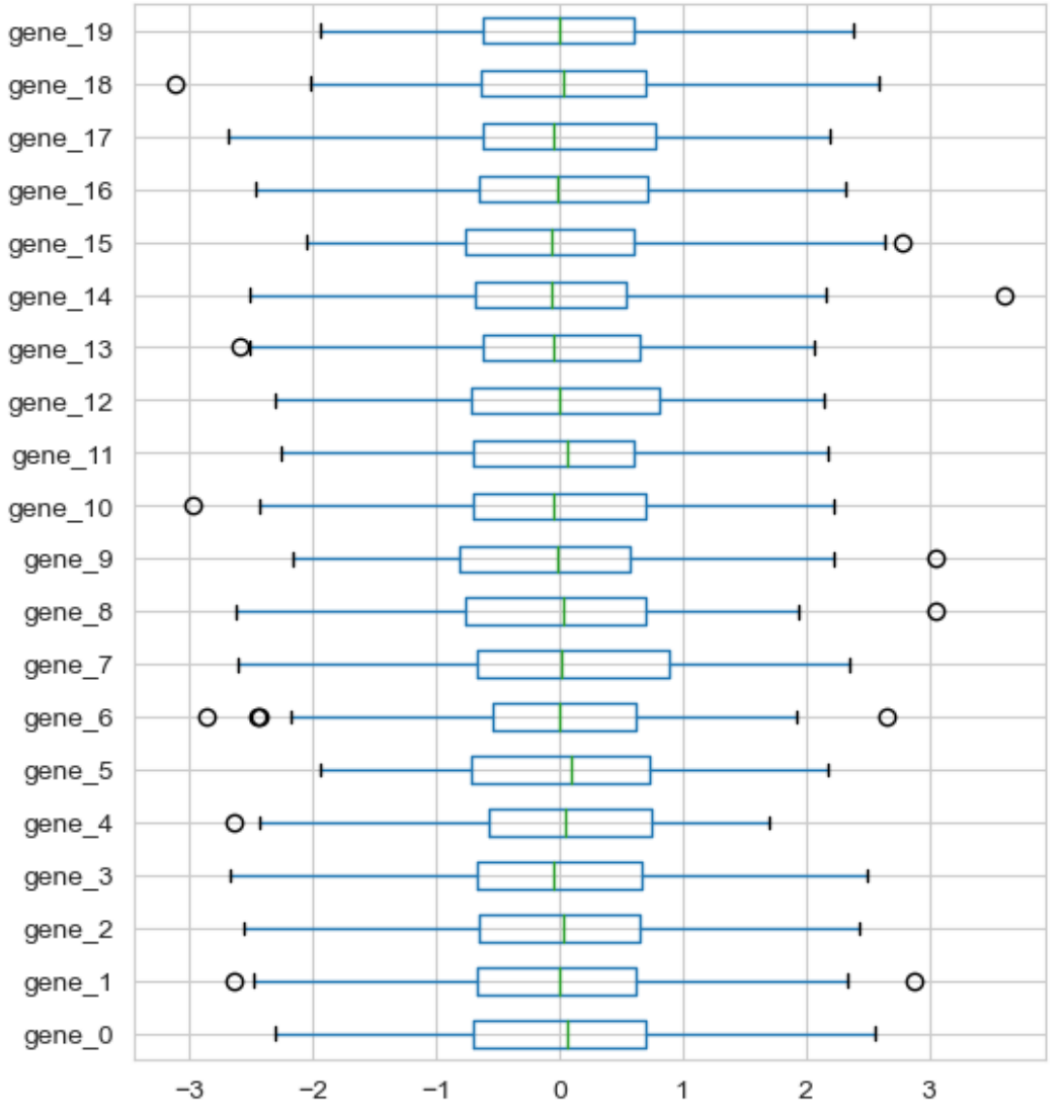
$$x_{norm} = \frac{x - x_{min}}{x_{max} - x_{min}}$$

$$x_{norm} = \frac{x - \mu}{\sigma}$$

Data normalization

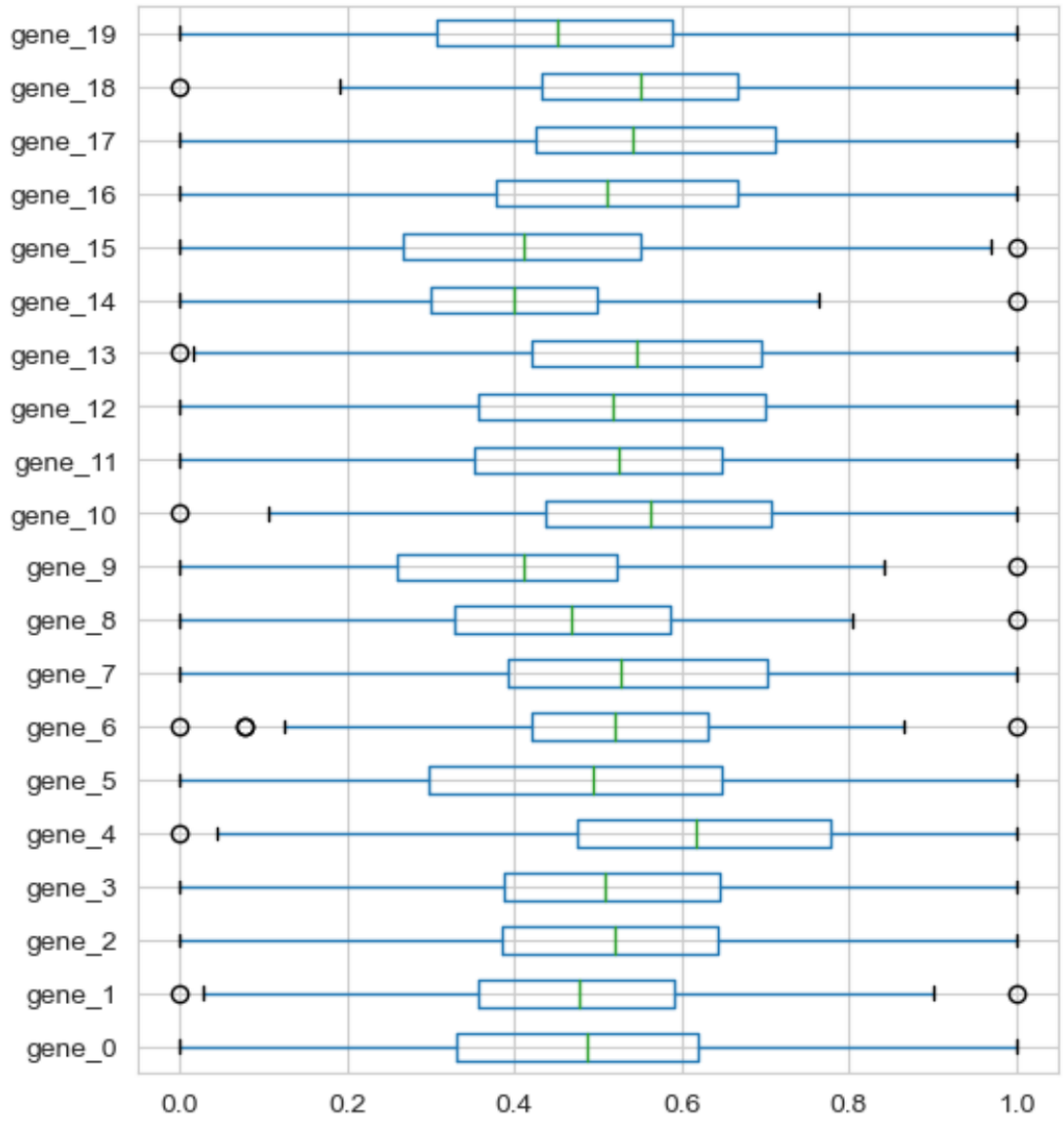


Data normalization: standardization



$$x_{norm} = \frac{x - \mu}{\sigma}$$

Data normalization: min-max scaling



$$x_{norm} = \frac{x - x_{min}}{x_{max} - x_{min}}$$

Data leakage

Data leakage occurs when information from the test set influences the model during training.

If preprocessing (e.g., scaling, imputation) is done **before splitting**, the statistics (e.g., mean, standard deviation, min-max values) are computed on the **entire dataset**.

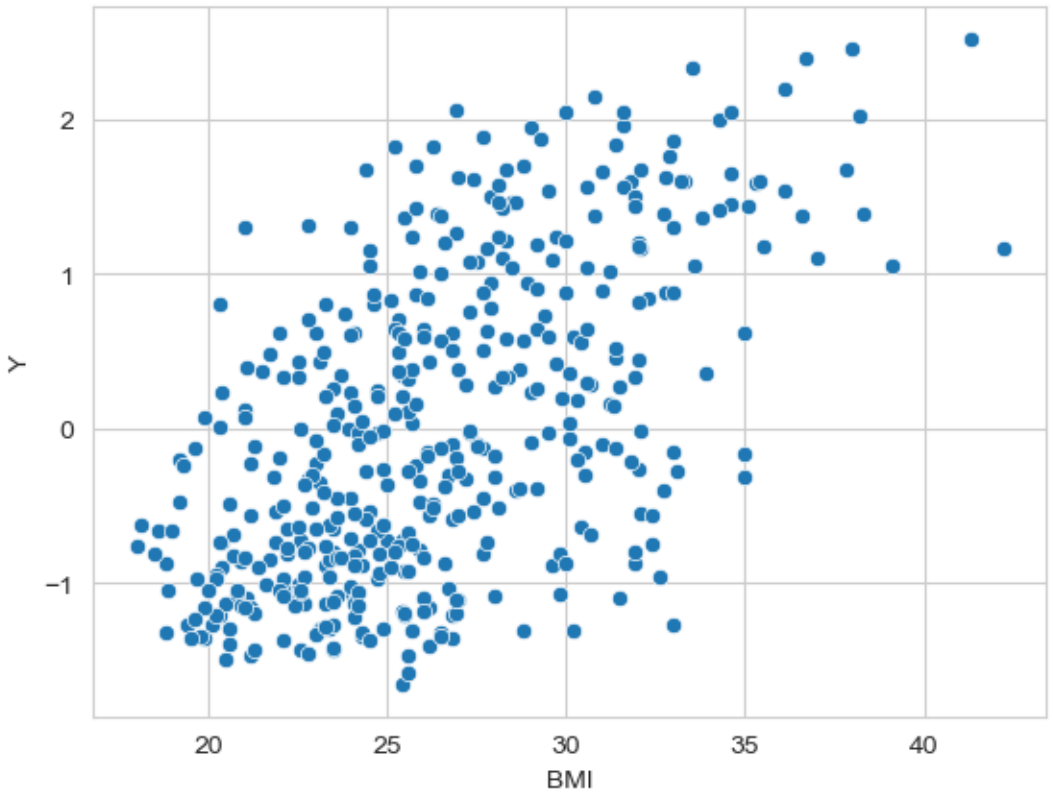
This means the test data indirectly influences the transformations applied to the training data, which **violates the principle that the test set should remain unseen** until final evaluation.

Data preprocessing

GitHub

data_preprocessing.ipynb

Regression



Can BMI explain Y?

Can BMI predict Y?

How does Y vary with BMI?

Regression is a general term for **modeling the relationship** between a **label** (a.k.a. dependent variable or target) and one or more **features** (a.k.a. the attributes, the independent or explanatory variable(s)).

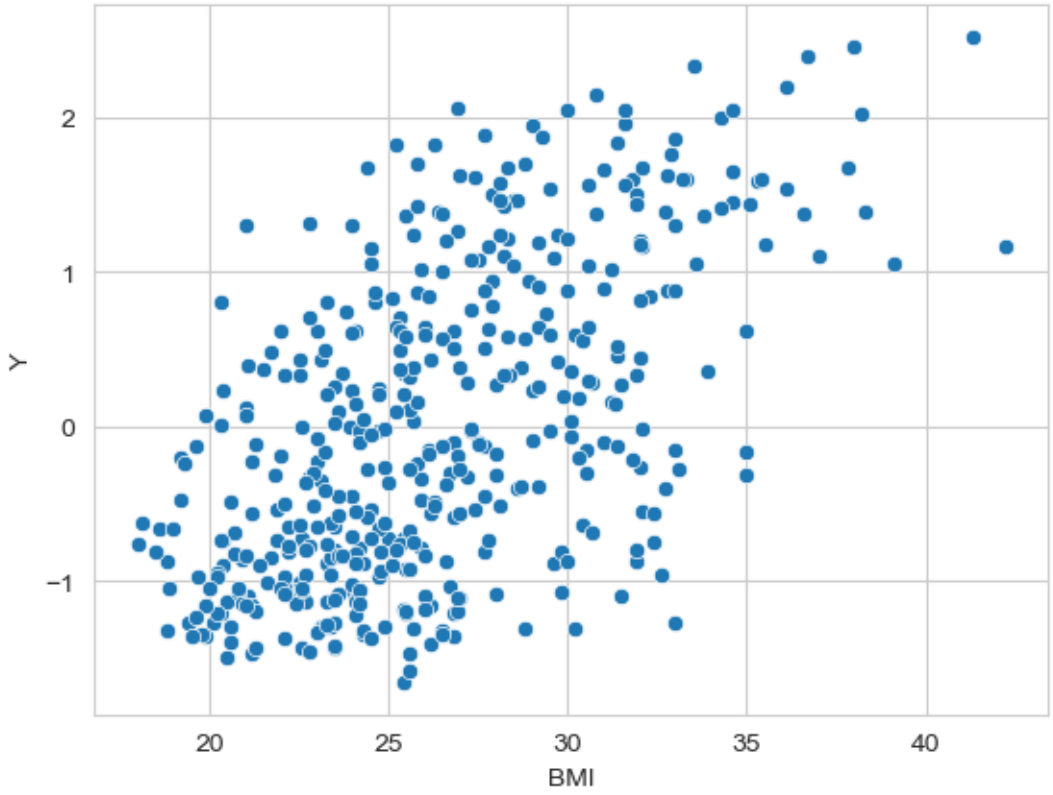
Linear regression

Linear regression models the relationship between input features (X) and a target variable (Y) using a linear equation: $Y=wX+b$, where w represents the **weight** (slope) and b is the **bias** (intercept).

It minimizes the error using the **least squares method**, which finds the best-fit line by reducing the sum of squared differences between actual and predicted values.

Simple Linear Regression involves one independent variable, while Multiple Linear Regression extends this to multiple input features, allowing for more complex predictions.

Linear regression



We need to make

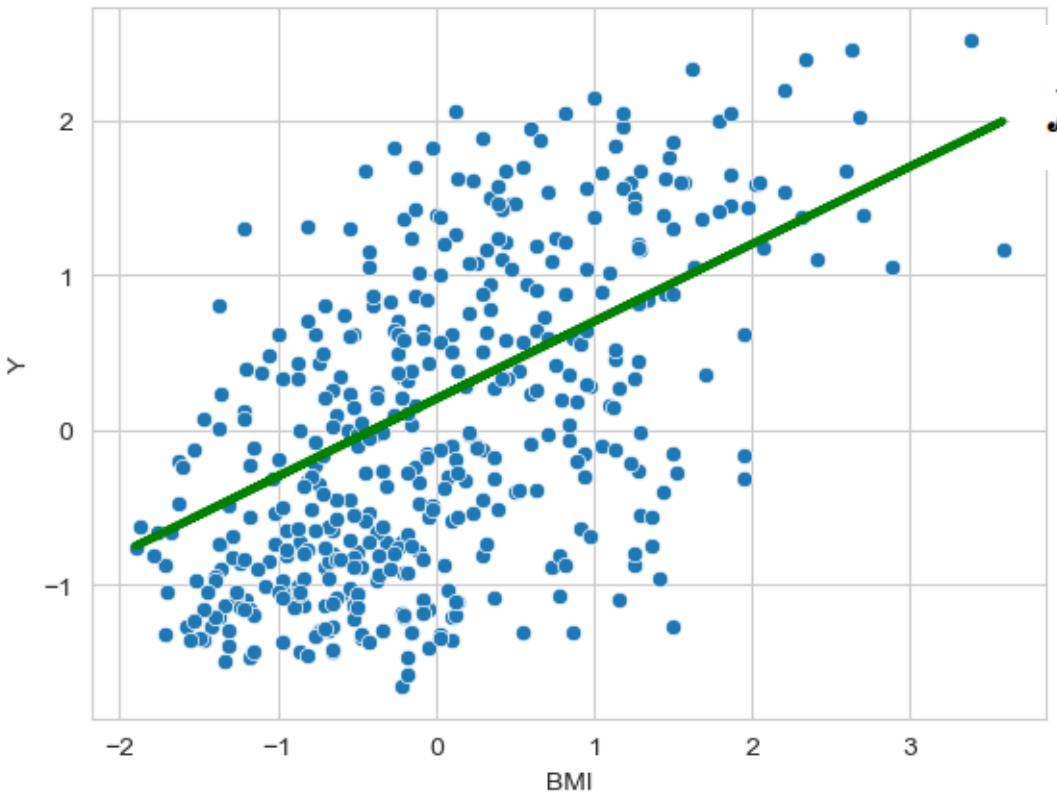
assumptions *linear relationship*

about the

model *linear model*

that generated the data.

Linear regression



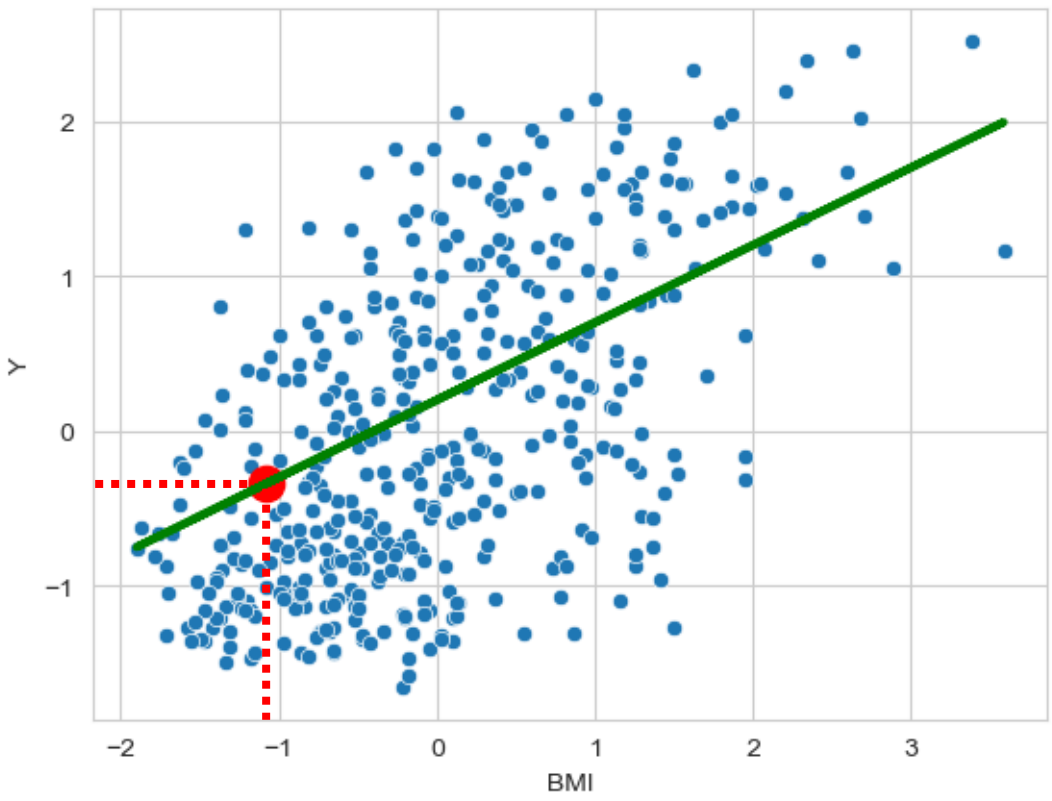
$$f(x) = 0.5 \cdot x + 0.2 = ax + b$$

a and *b* are **model parameters**

a is the **slope** to the line (direction)

b is the **intercept** or bias (position)

Linear regression: prediction



BMI(70kg, 1.8 meter) = 21.6

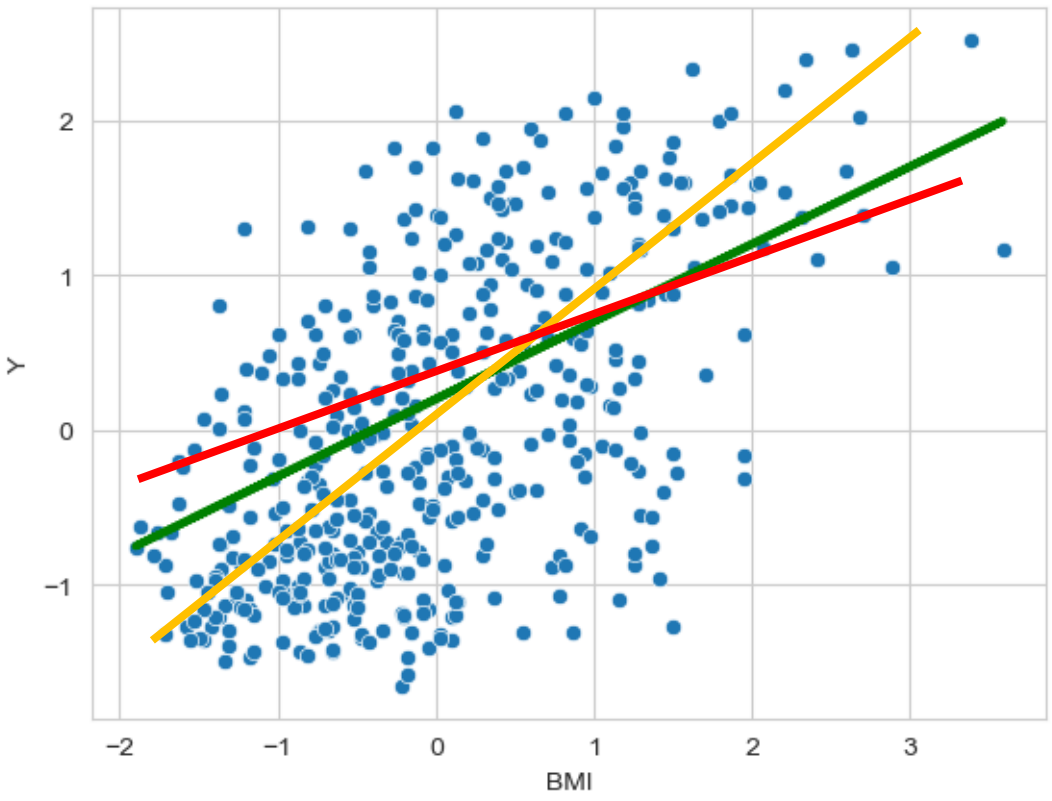
scaled data!

scaled BMI = -1.08

predicted value = -0.34

generalization!!

Linear regression: fitting



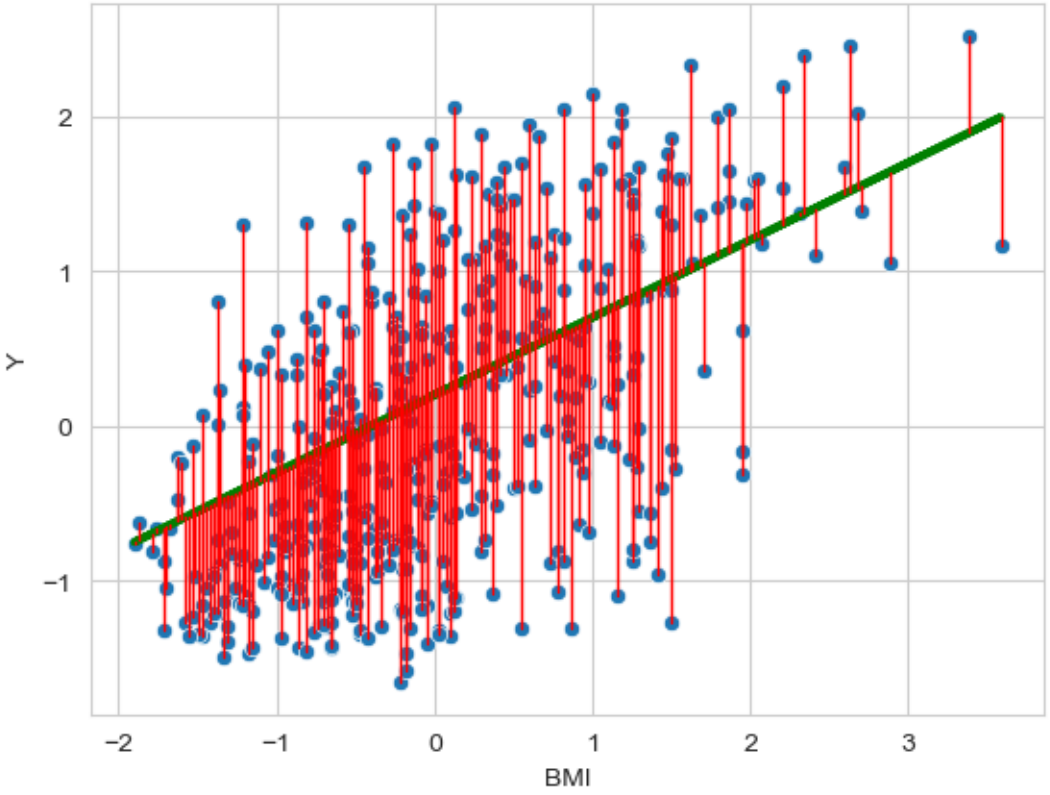
What values for *a* and *b* fit the data best?

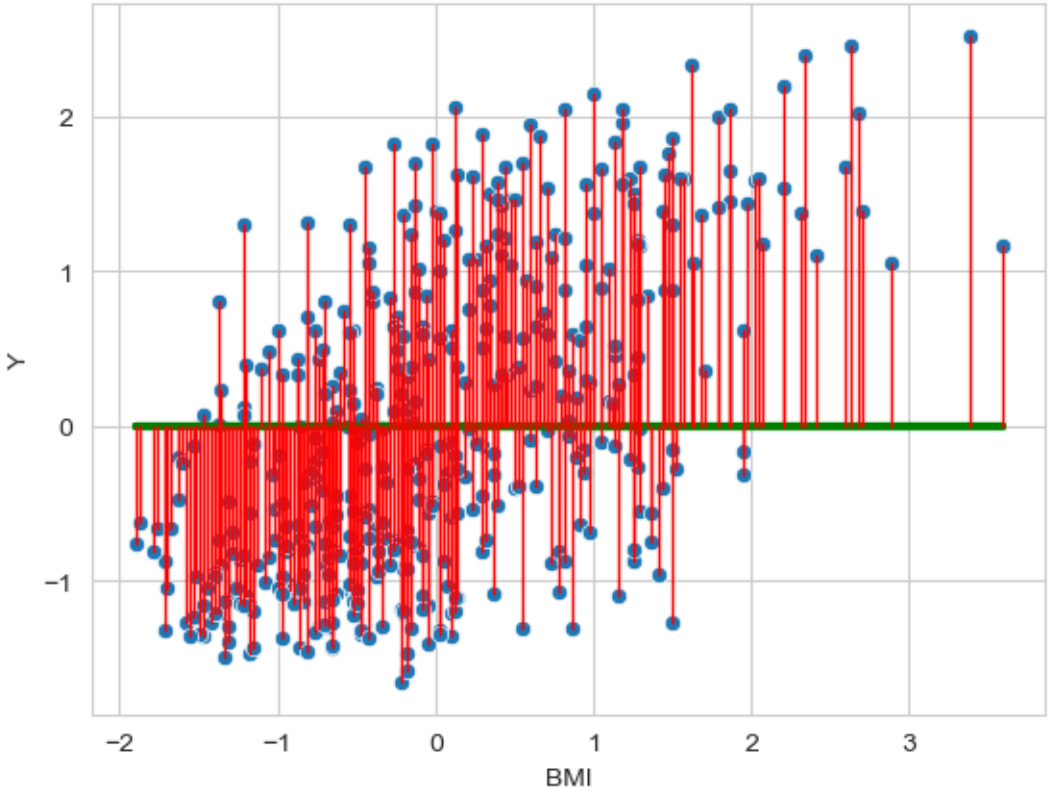
How can we evaluate them?

Linear regression: fitting

$$SS_{error} = \sum_{i=1}^{n} (y^{(i)} - f(x^{(i)}))^2$$

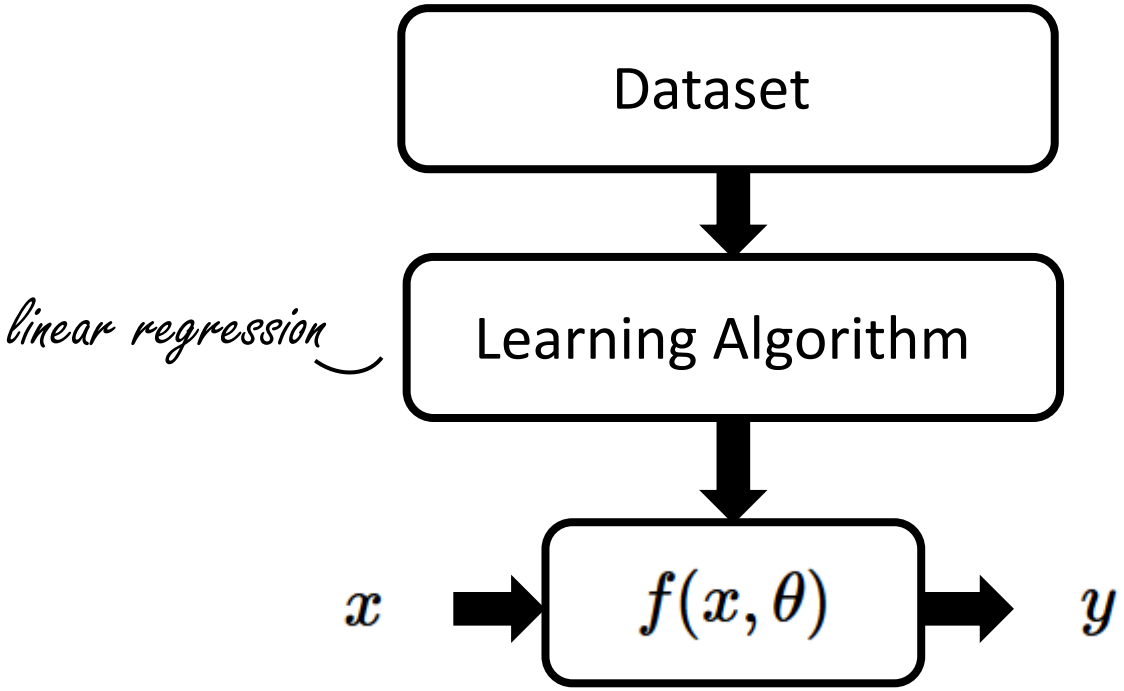
$$SS_{total} = \sum_{i=1}^{n} (y^{(i)} - \bar{y})^2$$

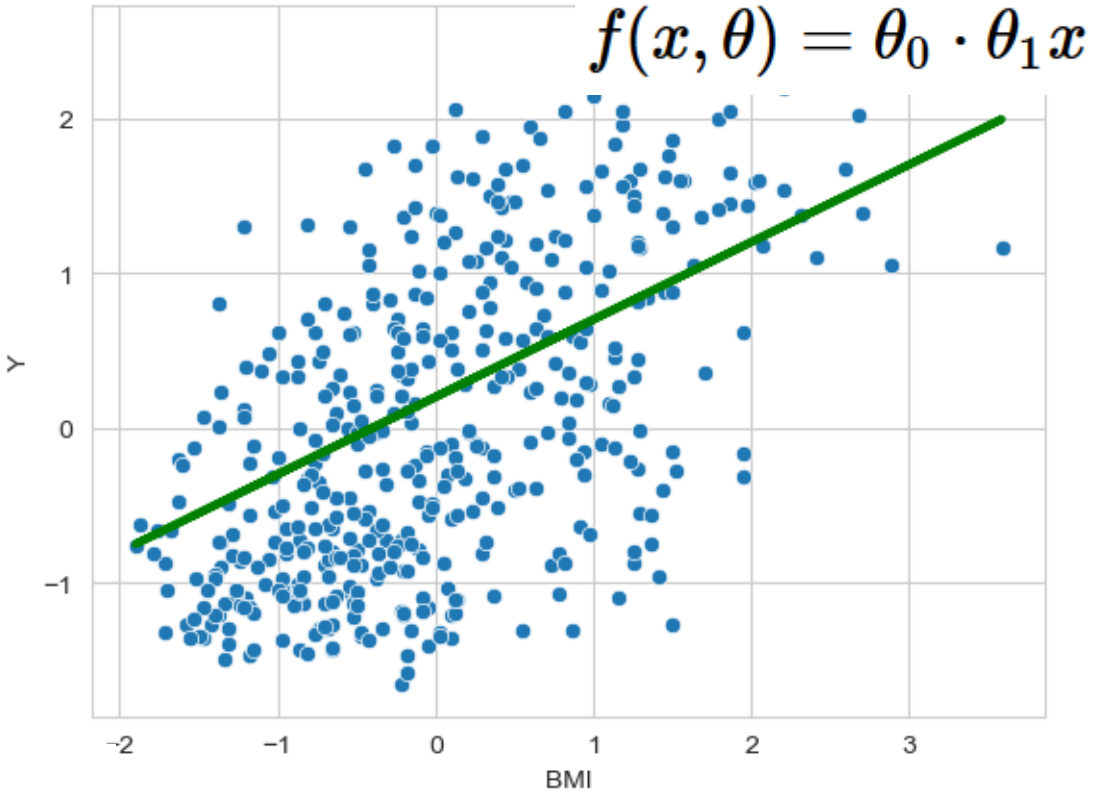




$$R^2 = 1 - \frac{SS_{error}}{SS_{total}} = 0.296$$

Linear regression: formalized



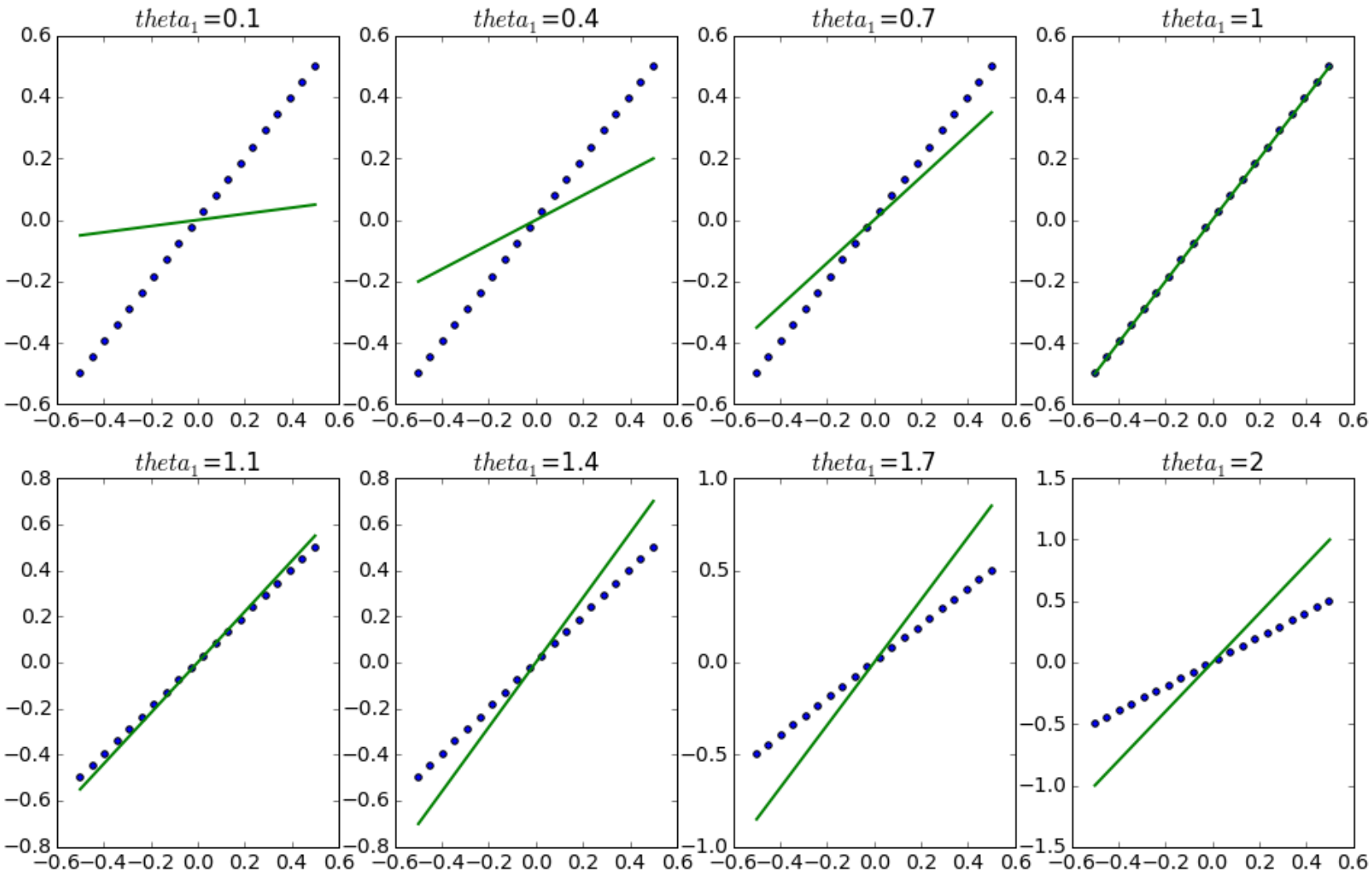


$$f(x,\theta) = \theta_0 x_0 + \theta_1 x_1 + \theta_2 x_2 + \ldots + \theta_m x_m = \theta' x$$

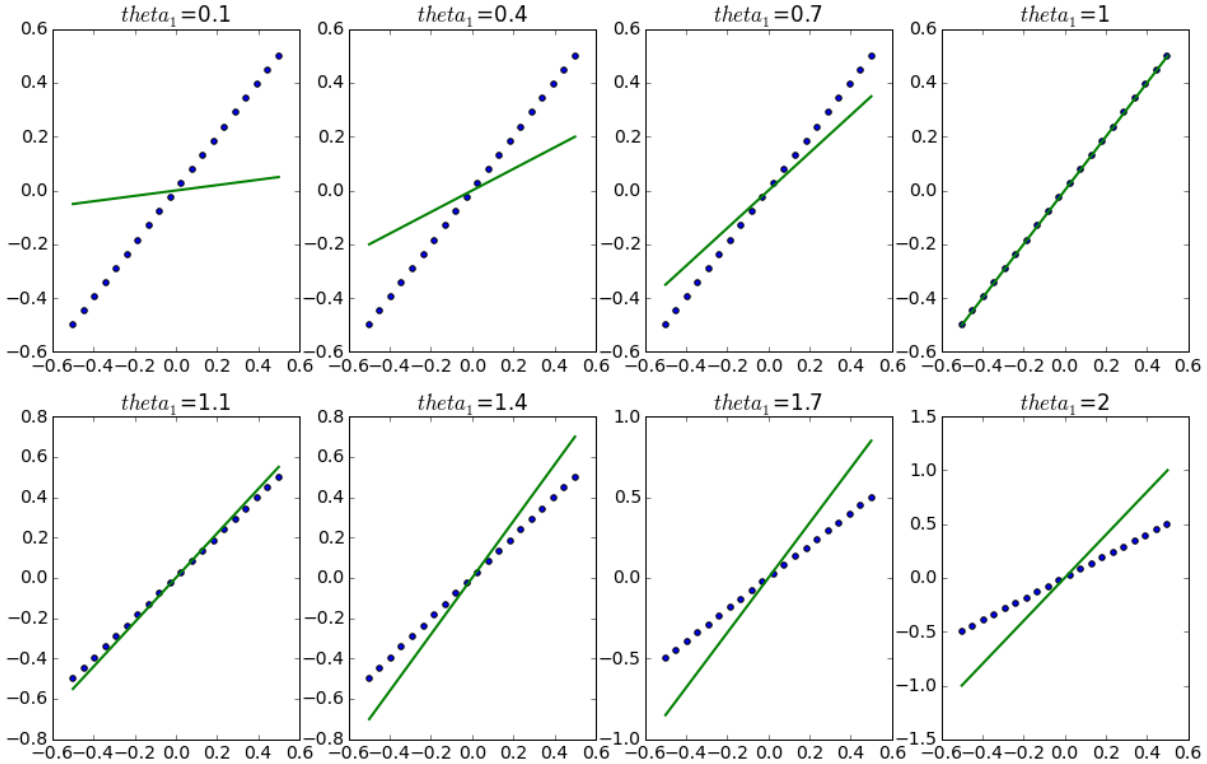
Linear regression: cost (loss function)

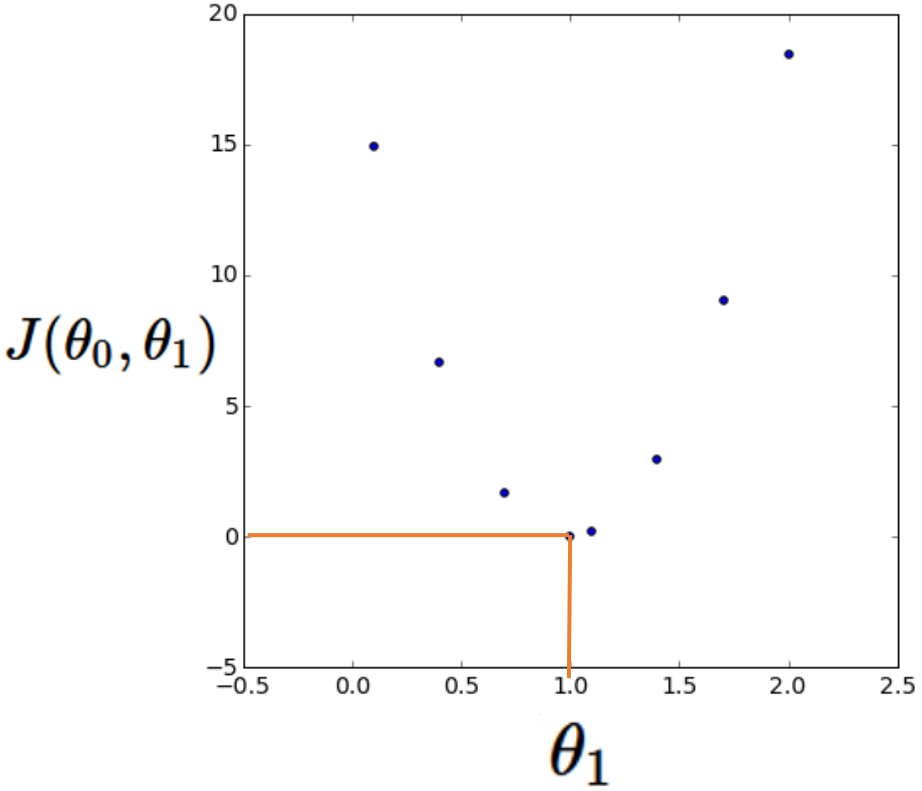
$$f(x) = x$$

$$(\theta_1 = 1, \theta_0 = 0)$$



Linear regression: cost (loss function)





$$J(\theta_0, \theta_1) = \frac{1}{2n} \sum_{i=1}^{n} (f(x^{(i)}, \theta) - y^{(i)})^2$$

Linear regression

Fit a linear model

$$f(x, \theta) = \theta_0 x_0 + \theta_1 x_1 + \theta_2 x_2 + \ldots + \theta_m x_m = \theta' x$$

to the data set such that the cost function

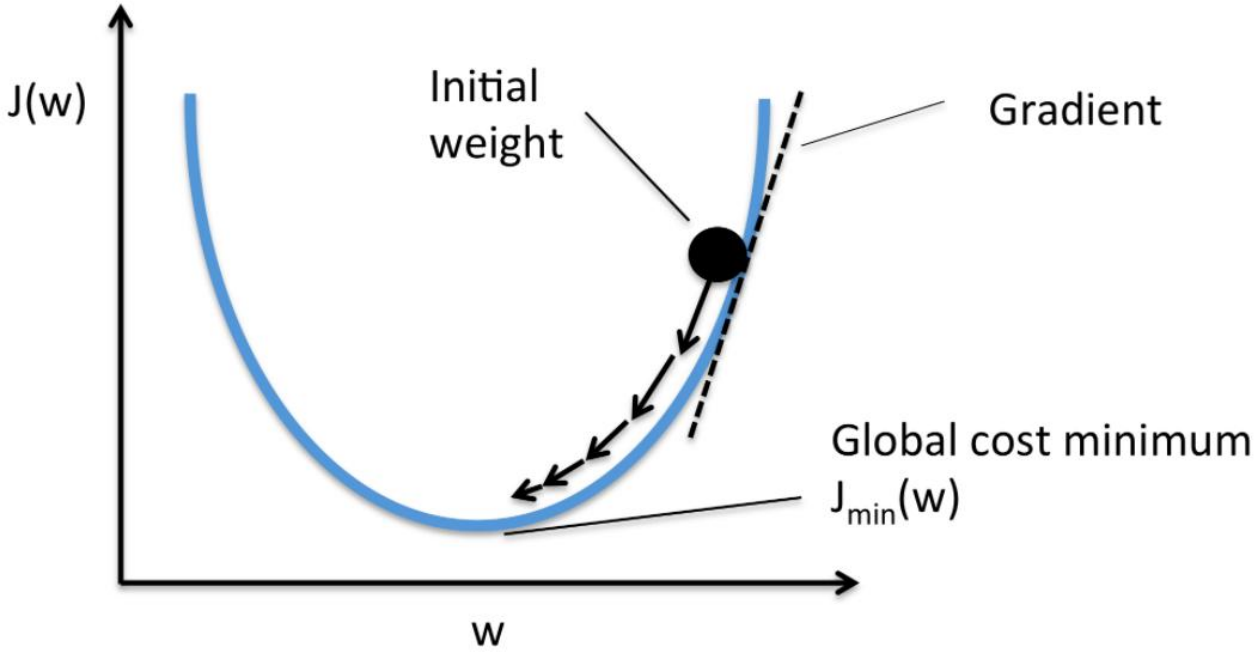
$$J(\theta) = \frac{1}{2n} \sum_{i=1}^{n} (f(x^{(i)}, \theta) - y^{(i)})^2$$

is minimal.

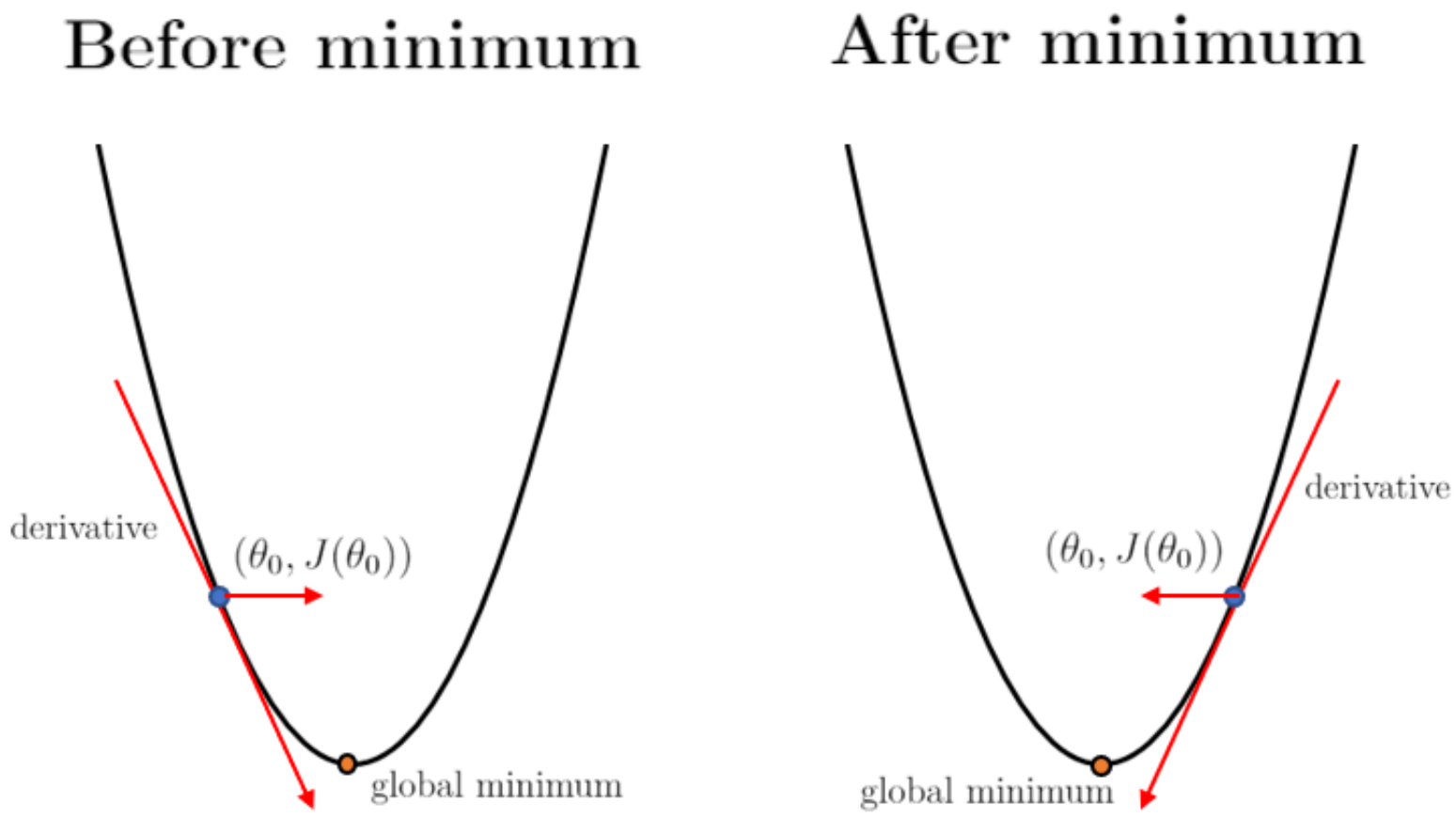
Learning (aka Training or Fitting): Gradient Descent

Gradient Descent is an optimization algorithm used to minimize a loss function by **iteratively** updating model parameters in the direction of the steepest descent (**negative gradient**).

The **learning rate** (α) controls the step size of updates, where too large a value may cause divergence, and too small a value may lead to slow convergence.



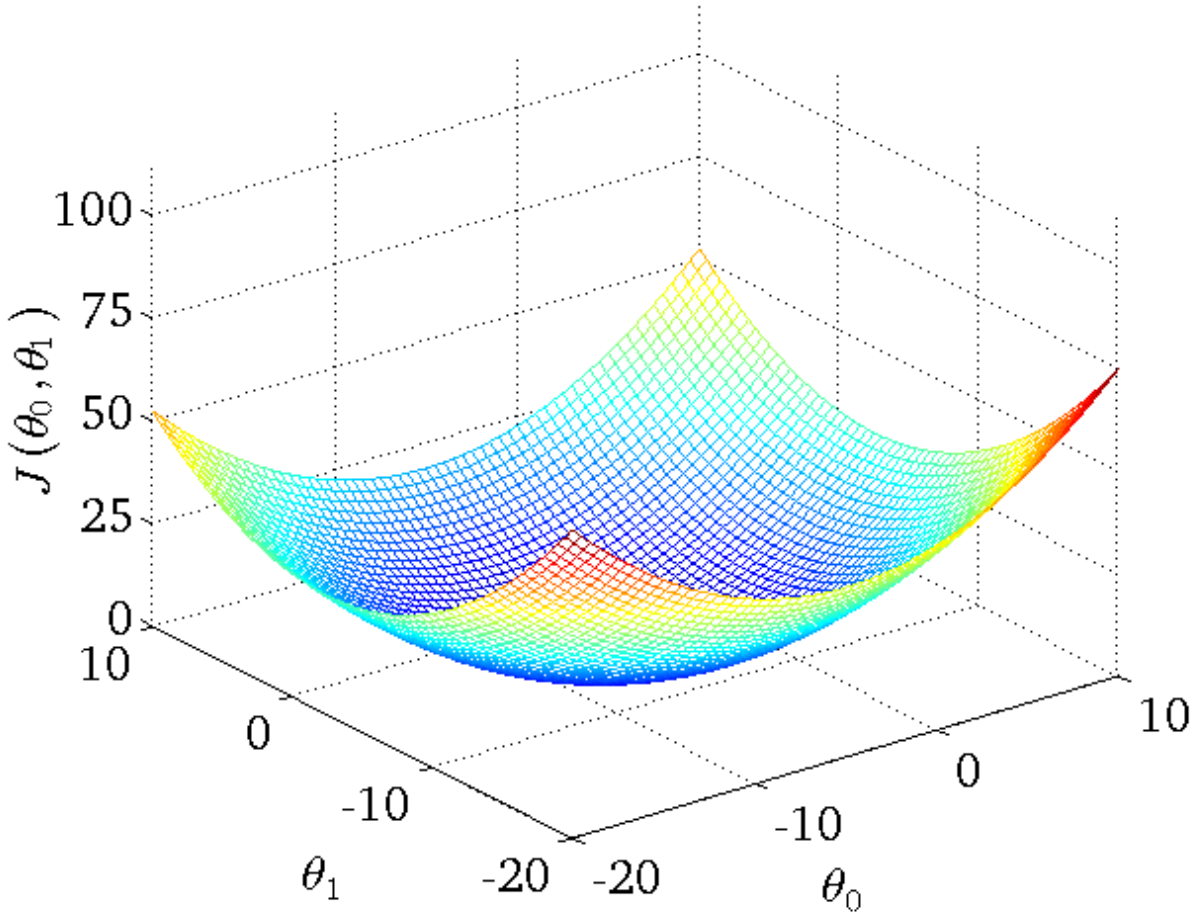
Gradients



Since the derivative is negative,
if we subtract the derivative from θ_0,
it will increase and go closer the minimum.

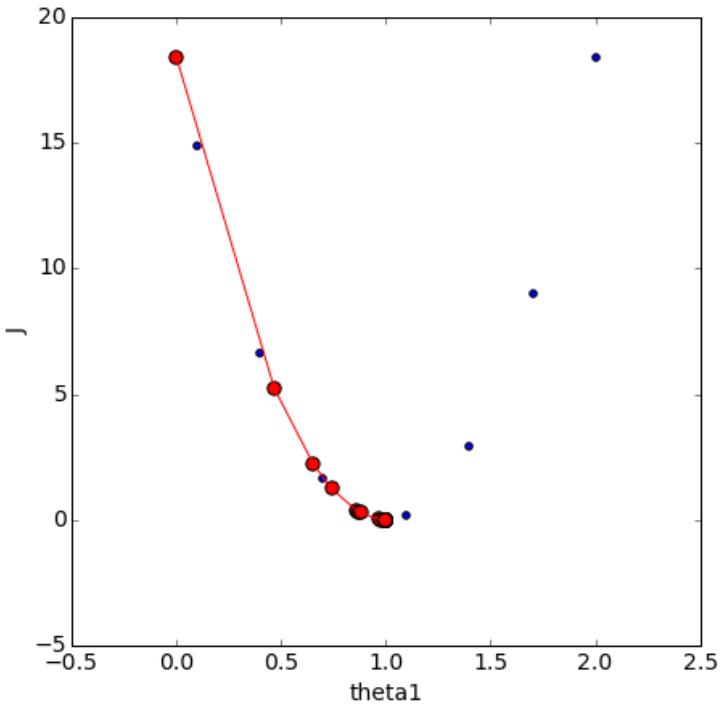
Since the derivative is positive,
if we subtract the derivative from θ_0,
it will decrease and go closer the minimum.

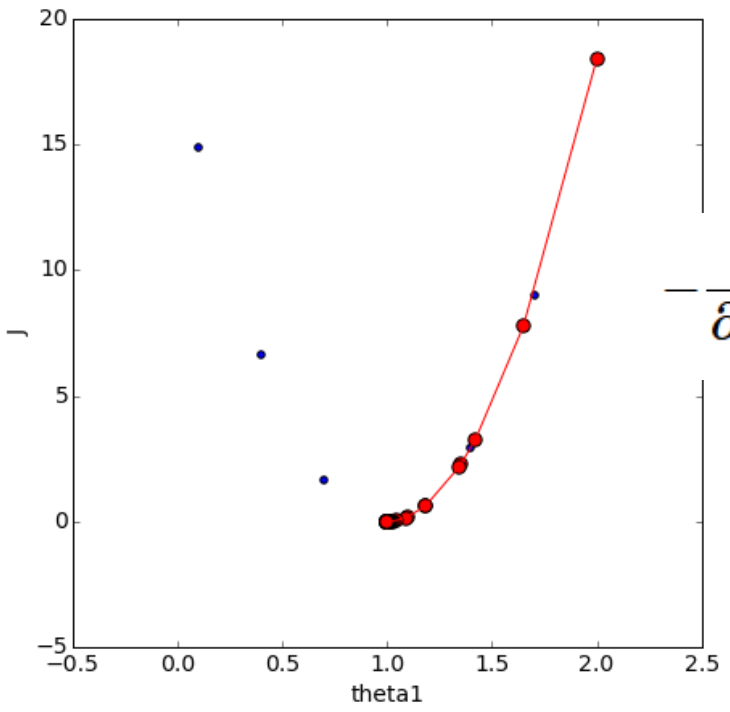
Linear regression: cost (loss function)



Gradient descent

1. start with some random initialization of θ, i.e. a randomly chosen model
2. increment or decrement the value(s) of θ slightly such that $J(\theta)$ is reduced
3. repeat step 2. until we observe convergence





$$-\frac{\partial}{\partial \theta_j} J(\theta)$$

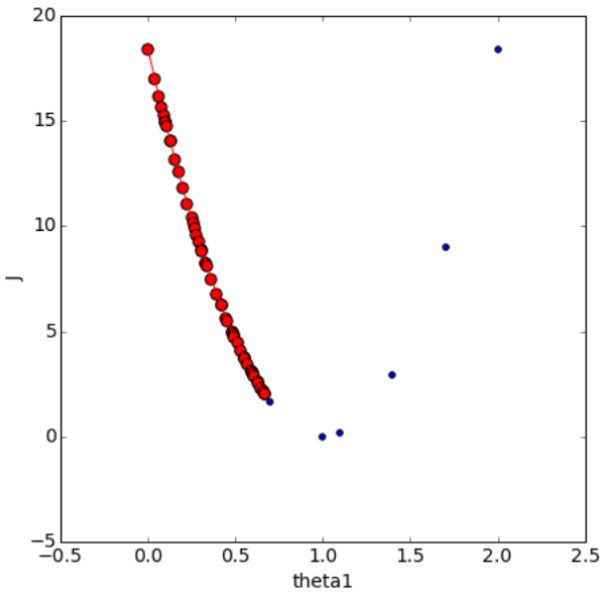
Gradient descent: learning rate

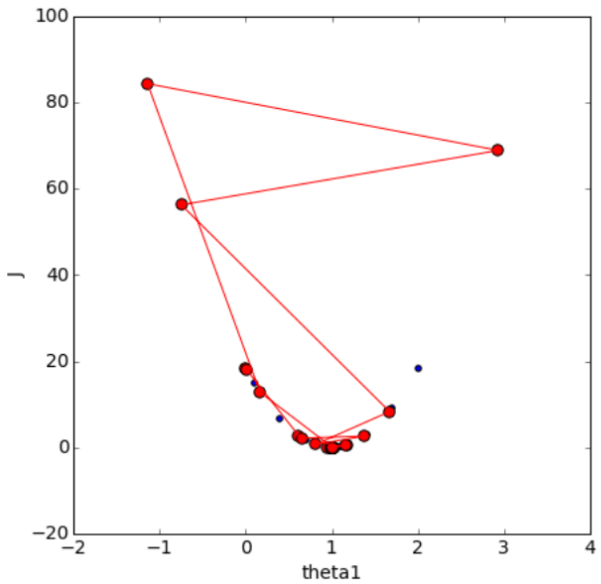
$$\theta_0 := \theta_0 - \alpha \frac{1}{n} \sum_{i=1}^{n} (f(x^{(i)}, \theta) - y^{(i)}) x_0^{(i)}$$

$$\theta_1 := \theta_1 - \alpha \frac{1}{n} \sum_{i=1}^{n} (f(x^{(i)}, \theta) - y^{(i)}) x_1^{(i)}$$

$$\theta_2 := \theta_2 - \alpha \frac{1}{n} \sum_{i=1}^{n} (f(x^{(i)}, \theta) - y^{(i)}) x_2^{(i)}$$

$$\ldots$$





Linear regression

model: $$f(x,\theta) = \theta_0 x_0 + \theta_1 x_1 + \theta_2 x_2 + \ldots + \theta_m x_m$$

cost function: $$J(\theta) = \frac{1}{2n}\sum_{i=1}^{n}(f(x^{(i)},\theta) - y^{(i)})^2$$

Goal: $$\underset{J(\theta)}{\text{minimize}} \quad J(\theta)$$

Learning:
1. start with some θ
2. change θ to reduce $J(\theta)$
3. repeat 2. until convergence

$$\theta_0 := \theta_0 - \alpha\frac{1}{n}\sum_{i=1}^{n}(f(x^{(i)},\theta) - y^{(i)})x_0^{(i)}$$

$$\theta_1 := \theta_1 - \alpha\frac{1}{n}\sum_{i=1}^{n}(f(x^{(i)},\theta) - y^{(i)})x_1^{(i)}$$

$$\theta_2 := \theta_2 - \alpha\frac{1}{n}\sum_{i=1}^{n}(f(x^{(i)},\theta) - y^{(i)})x_2^{(i)}$$

...

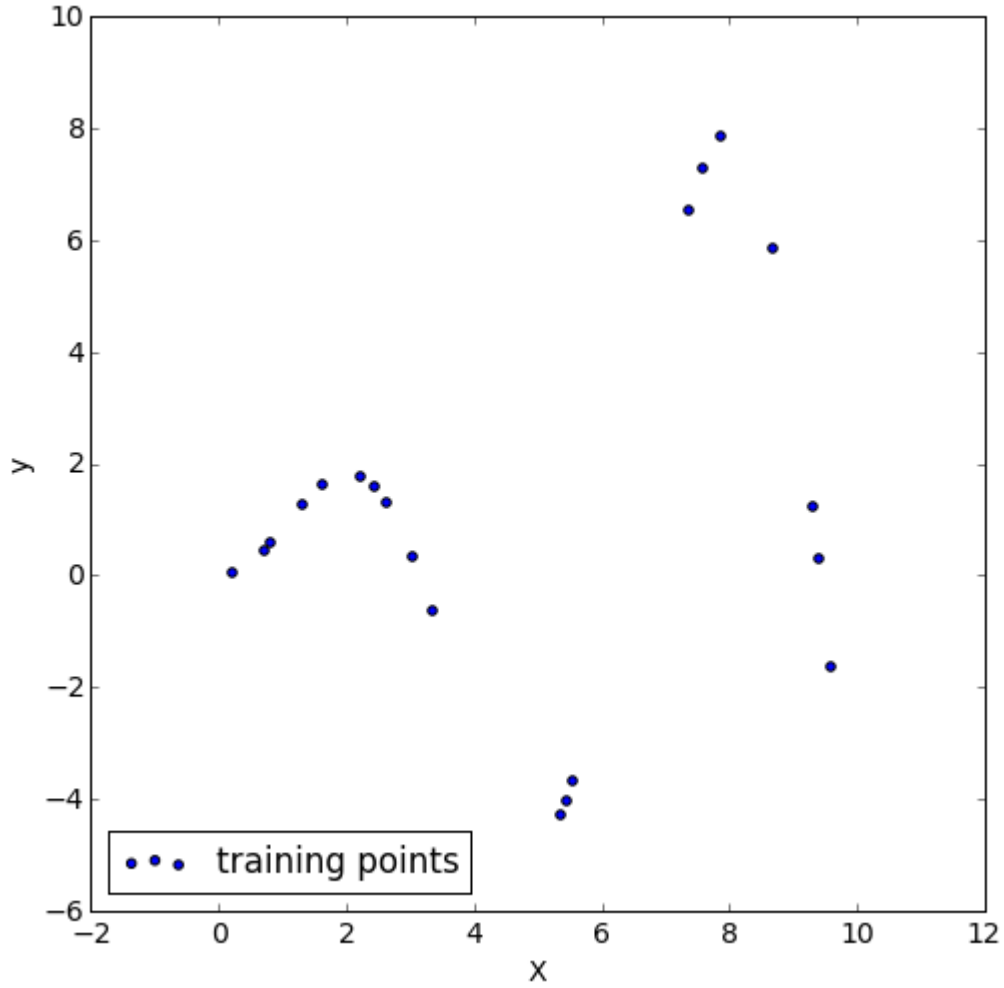
Non-linear regression

Polynomial transformation creates **higher-order features** from the original features, allowing a linear model to learn complex, non-linear relationships in the data.

Linear regression or classification models can then fit non-linear patterns by treating these transformed features as **inputs** while still using a linear equation in the higher-dimensional space.

This effectively changes the **decision boundary in the original feature space** from a straight line to a curve, enabling the model to separate or fit data points that a purely linear model could not handle.

Non-linear regression



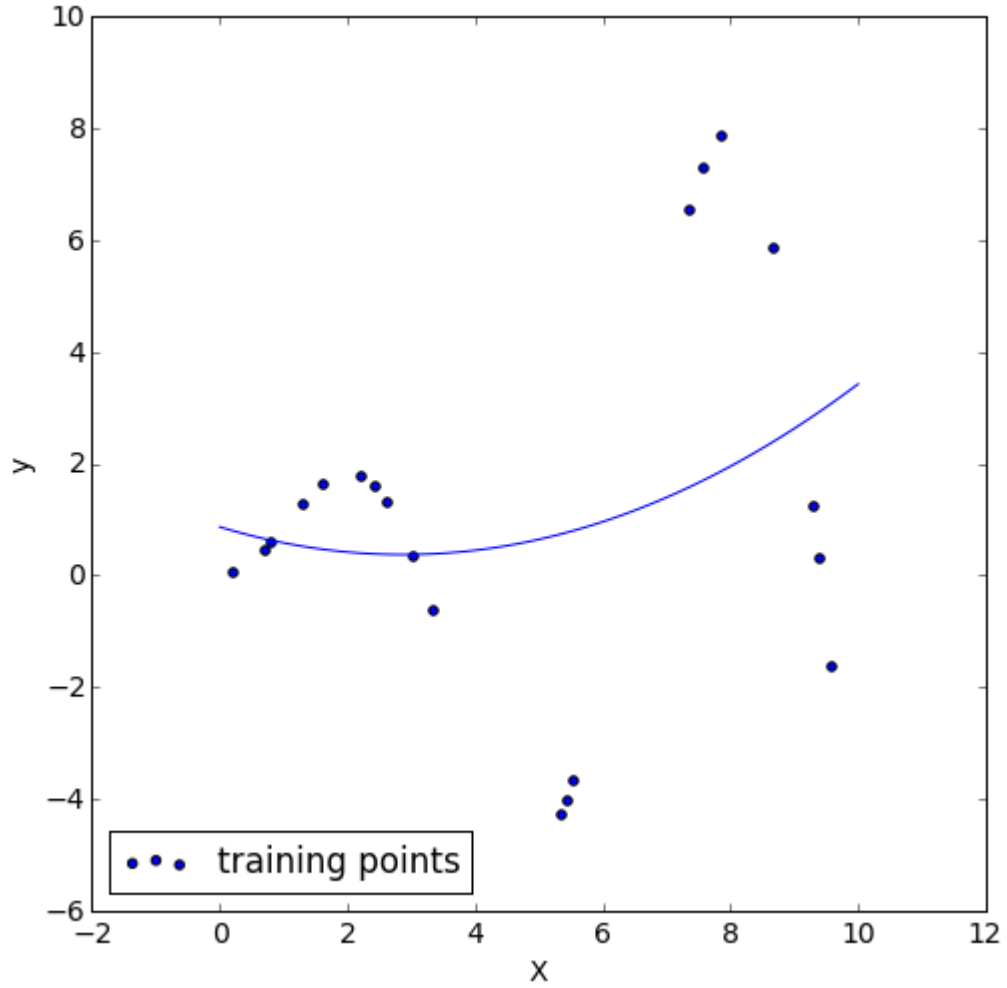
Does *y* vary linearly with *X*?

Can we fit a non-linear model? Yes

Or we could add polynomial transformations of the features.

$$f(x, \theta) = \theta_0 x_0 + \theta_1 x_1 + \theta_2 x_1^2$$

Non-linear regression

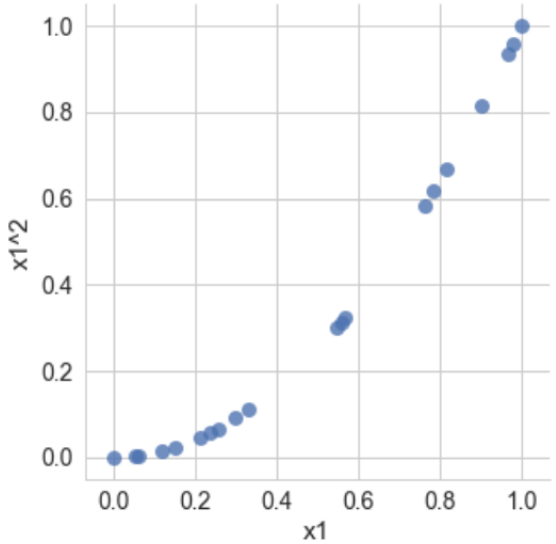


Does *y* vary linearly with *X*?

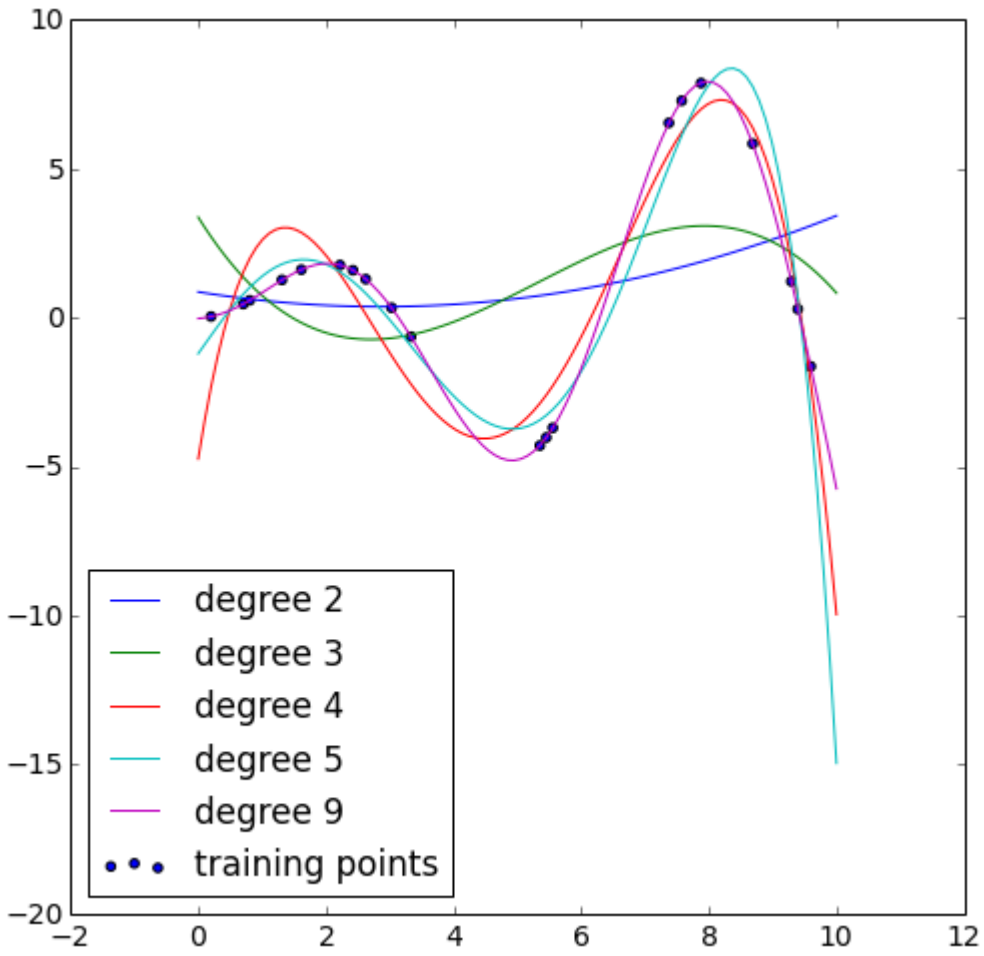
Can we fit a non-linear model? Yes

Or we could add polynomial transformations of the features.

$$f(x, \theta) = \theta_0 x_0 + \theta_1 x_1 + \theta_2 x_1^2$$

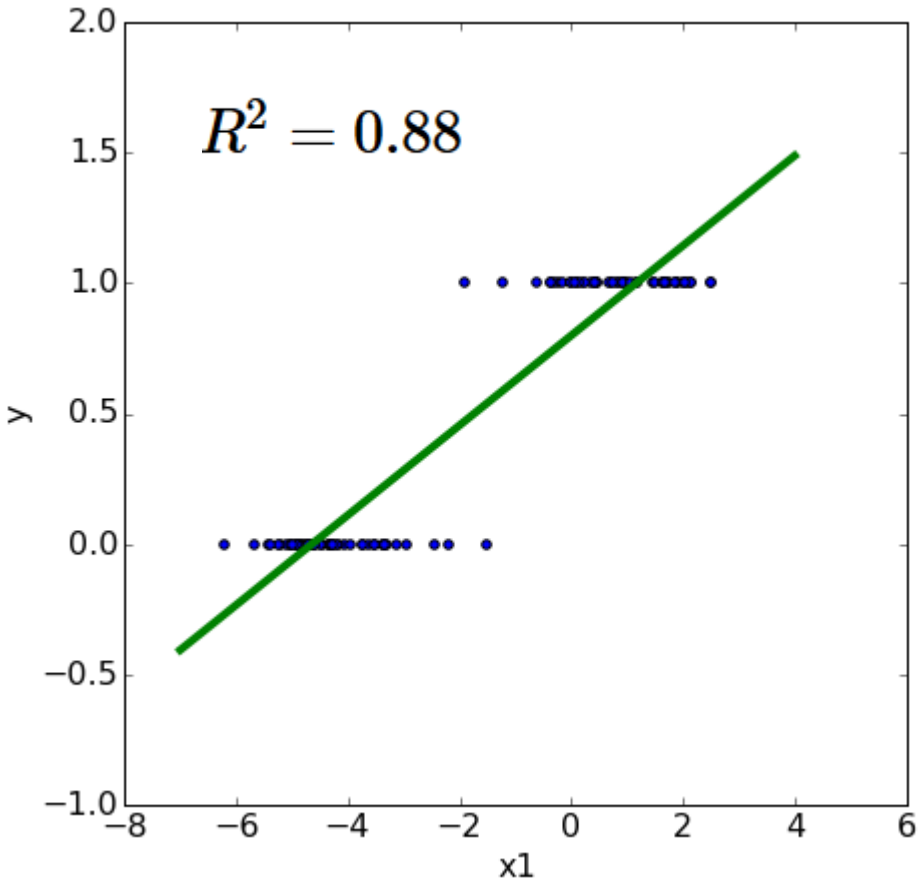


Non-linear regression



$$f(x,\theta) = \theta_0 x_0 + \theta_1 x_1 + \theta_2 x_1^2 + \theta_3 x_1^3 + \theta_4 x_1^4 + \theta_5 x_1^5 + \theta_6 x_1^6 + \theta_7 x_1^7 + \theta_8 x_1^8 + \theta_9 x_1^9$$

Classification



We need to make

assumptions *linear relationship*

about the

model *linear model*

that generated the data.

Logistic regression

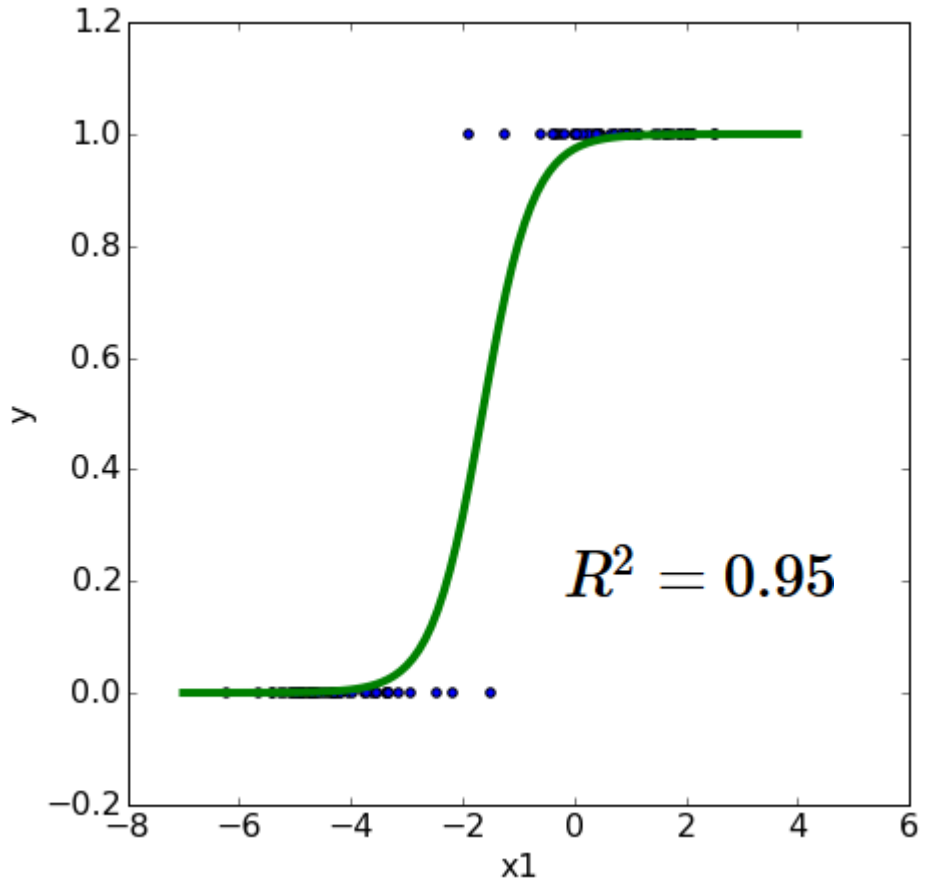
Logistic regression is a classification algorithm that predicts probabilities for binary or multi-class outcomes using the **sigmoid** function to map outputs between 0 and 1.

It models the relationship between input features and the **probability** of a class.

The decision boundary is defined by a threshold (typically 0.5), where predictions above the threshold are classified as one class and below as the other.

Optimization is performed using gradient descent to minimize the log loss (**cross-entropy loss**), ensuring the best separation of classes.

Logistic regression



We need to make

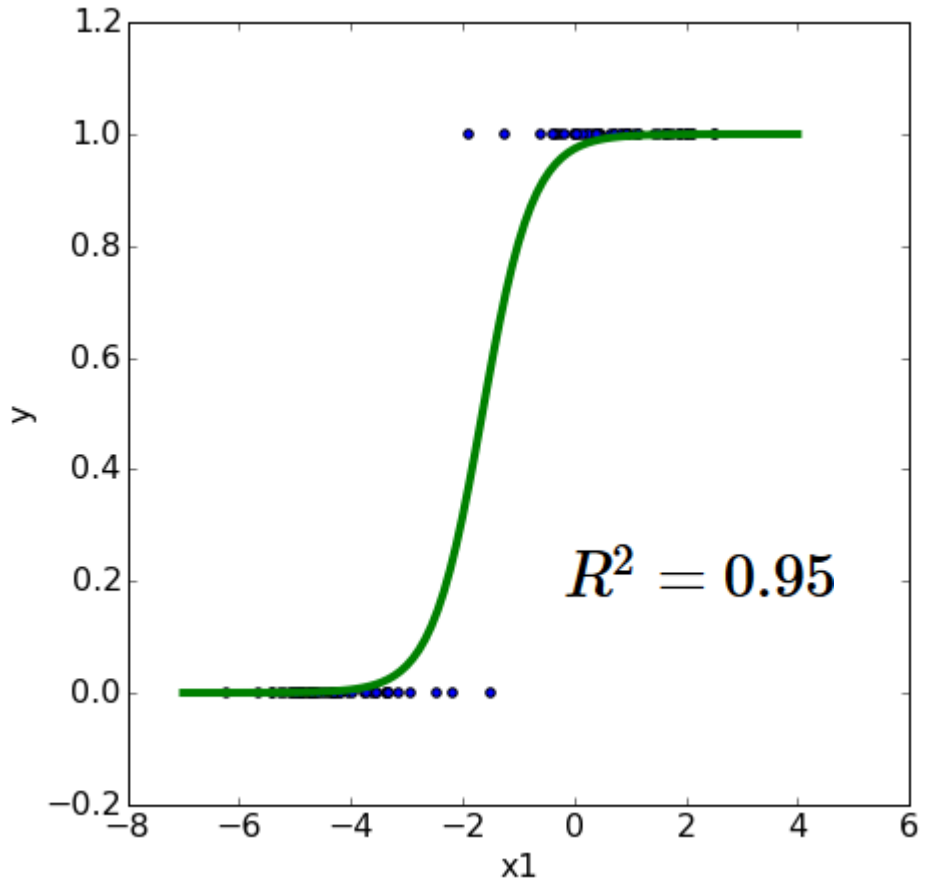
assumptions *linearly separable*

about the

model *logistic model*

that generated the data.

Logistic regression



$$f(x, \theta) = g(\theta_0 + \theta_1 x_1)$$

$$g(z) = \frac{1}{1 + e^{-z}}$$

Logistic regression: cost (loss function)

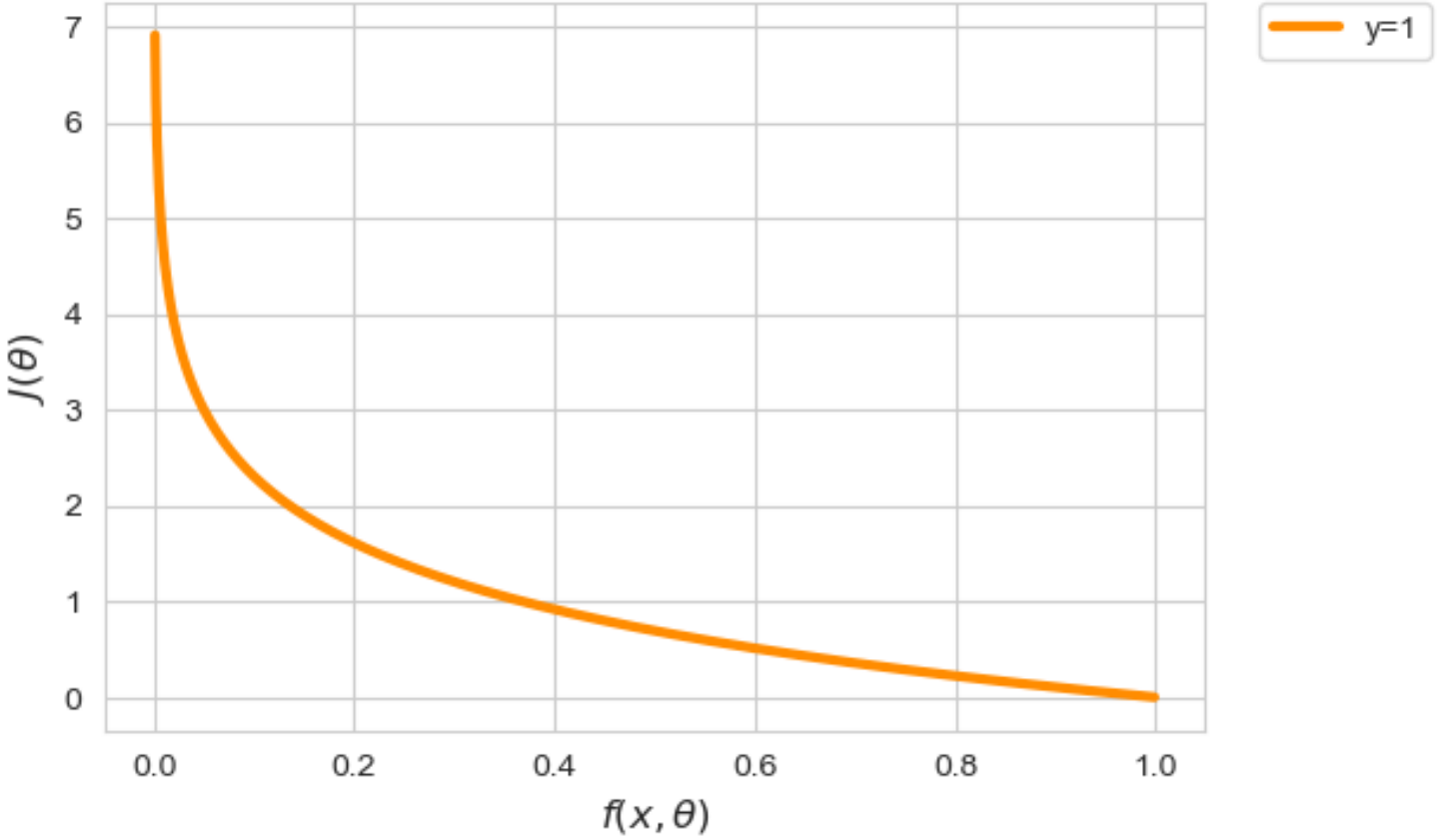
$$J(\theta) = -[\frac{1}{n}\sum_{i=1}^{n} y^{(i)} log(f(x^{(i)}, \theta)) + (1 - y^{(i)}) log(1 - f(x^{(i)}, \theta))]$$

We know that $y^{(i)}$ is either 0 or 1. If $y^{(i)} = 1$ then the cost function $J(\theta)$ is incremented by $-log(f(x^{(i)}, \theta))$.

Similarly, if $y^{(i)} = 0$ then the cost function $J(\theta)$ is incremented by $-log(1 - f(x^{(i)}, \theta))$.

Logistic regression: cost (loss function)

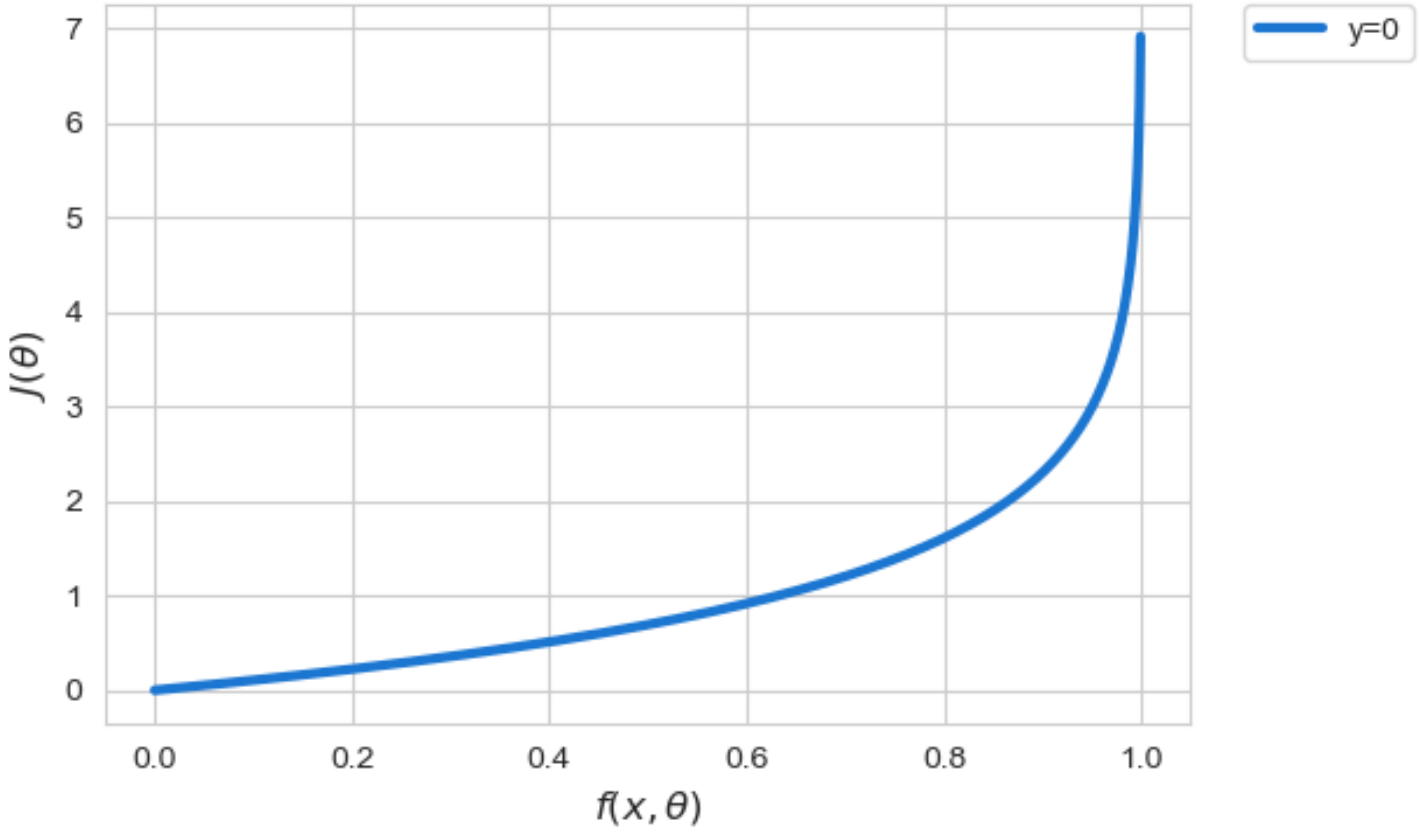
We know that $y^{(i)}$ is either 0 or 1. If $y^{(i)} = 1$ then the cost function $J(\theta)$ is incremented by $-log(f(x^{(i)}, \theta))$.



Logistic regression: cost (loss function)

Similarly, if $y^{(i)} = 0$ then the cost function $J(\theta)$ is incremented by

$$-log(1 - f(x^{(i)}, \theta)).$$



Logistic regression

Fit a logistic model

$$f(x,\theta) = g(\theta_0 x_0 + \theta_1 x_1 + \theta_2 x_2 + \ldots + \theta_m x_m) = g(\theta' x)$$

to the data set such that the cost function

$$J(\theta) = -[\frac{1}{n}\sum_{i=1}^{n} y^{(i)} log(f(x^{(i)},\theta)) + (1 - y^{(i)}) log(1 - f(x^{(i)},\theta))]$$

is minimal using gradient descent

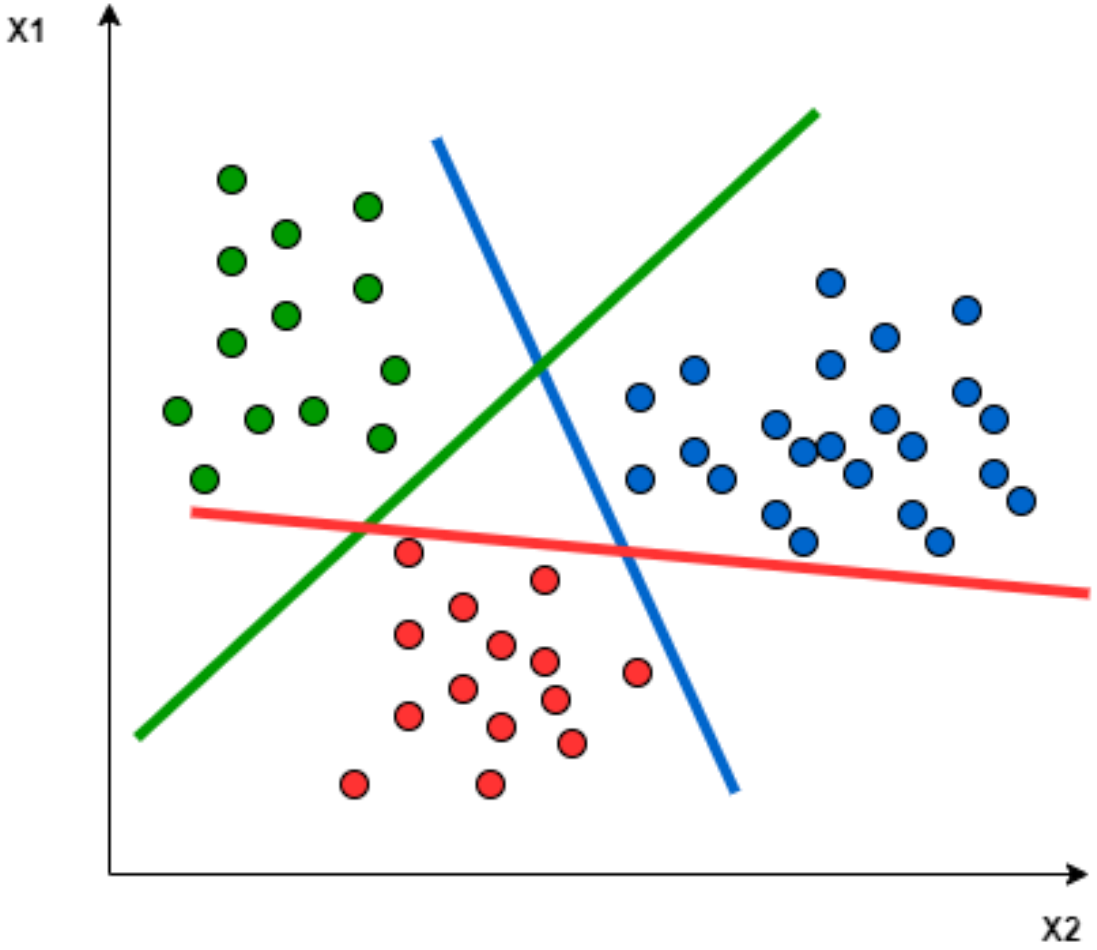
$$\theta_j := \theta_j - \alpha \frac{1}{n}\sum_{i=1}^{n}(f(x^{(i)},\theta) - y^{(i)})x_j^{(i)}$$

Multiclass classification: one-against-all

Fit a binary classifier for each class.

Each binary classifier computes a probability prediction.

The class with the highest probability is predicted.



Logistic regression

GitHub

breast_cancer_classification.ipynb

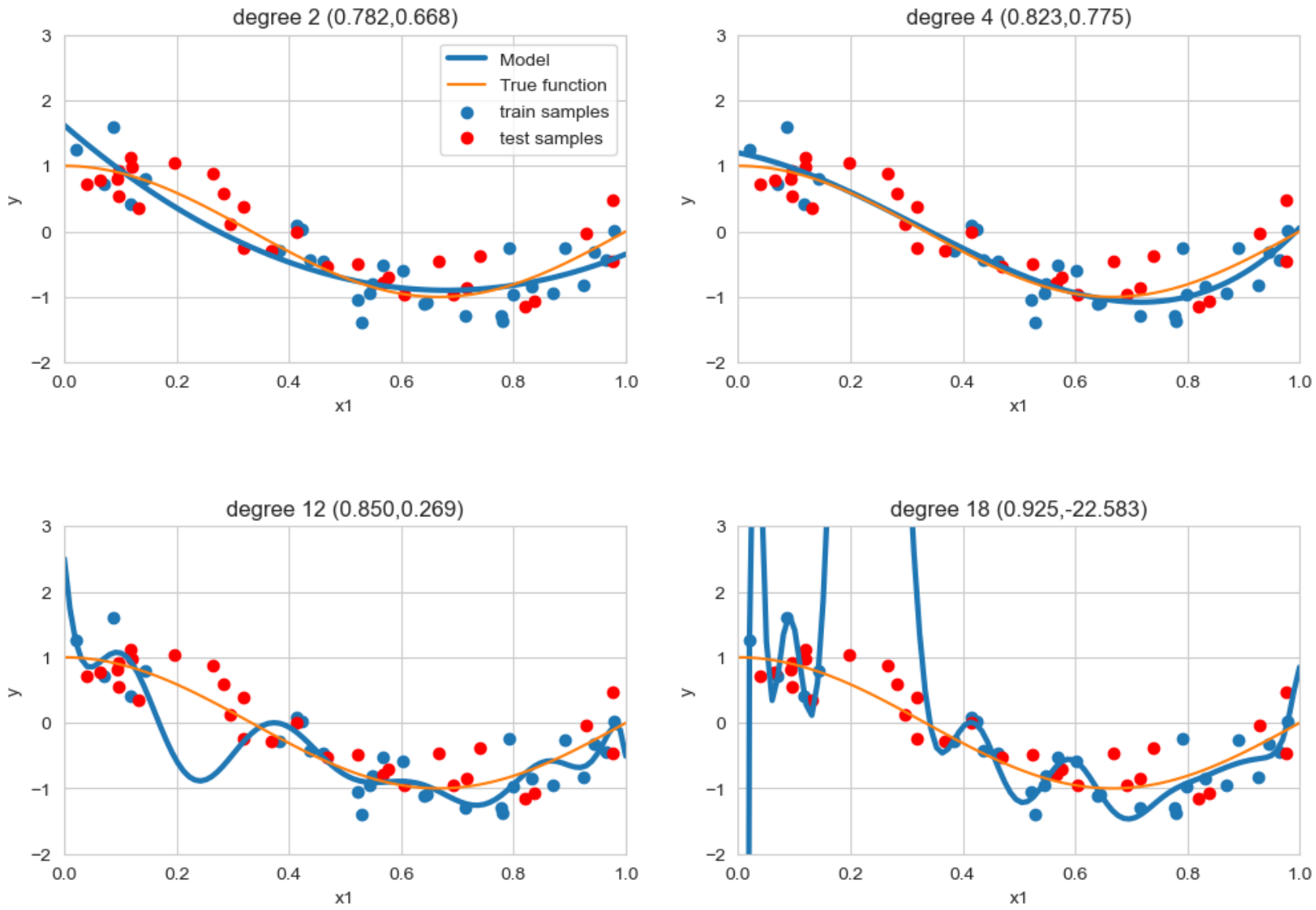
Logistic regression

```python
# Initialize the logistic regression model
model = LogisticRegression()

# Train the model on the scaled training data
model.fit(X_train_scaled, y_train)
```

Logistic regression

```python
# Make predictions on the test set

# Class predictions (0 or 1)
y_pred = model.predict(X_test_scaled)

# Probability of being in class 1 (benign)
y_pred_prob = model.predict_proba(X_test_scaled)[:, 1]

# Calculate accuracy
accuracy = accuracy_score(y_test, y_pred)
print(f"Accuracy: {accuracy:.4f}")

# Classification report
print("\nClassification Report:")
print(classification_report(y_test, y_pred, target_names=breast_cancer.target_names))
```

Model regularization



label noise

feature noise

feature relevance

relatively small train sets

don't try to fit the data perfectly

don't try to use all features

notice how R^2 on the train set does increase with *d*

Model regularization

Model regularization prevents overfitting by adding a **penalty term** to the loss function, discouraging overly complex models that memorize **noise** instead of learning patterns.

L1 regularization (Lasso) adds the absolute values of weights, promoting sparsity by forcing some weights to become exactly zero, effectively selecting important features.

L2 regularization (Ridge) adds the squared values of weights, preventing large weight values and ensuring smooth generalization without eliminating features.

Elastic Net combines L1 and L2 penalties, balancing feature selection and weight shrinkage, making it useful for high-dimensional datasets with correlated features.

Regularized linear regression

$$f(x,\theta) = \theta_0 x_0 + \theta_1 x_1 + \theta_2 x_2 + \ldots + \theta_m x_m$$

cost function

$$J(\theta) = \frac{1}{2n}\sum_{i=1}^{n}(f(x^{(i)},\theta) - y^{(i)})^2$$

$$J(\theta) = \frac{1}{2n}\sum_{i=1}^{n}(f(x^{(i)},\theta) - y^{(i)})^2 + \lambda\sum_{j=1}^{m}\theta_j^2$$

regularized cost function

$$\theta_0 := \theta_0 - \alpha\frac{1}{n}\sum_{i=1}^{n}(f(x^{(i)},\theta) - y^{(i)})x_0^{(i)}$$

$$\theta_j := \theta_j - \alpha\frac{1}{n}\sum_{i=1}^{n}(f(x^{(i)},\theta) - y^{(i)})x_j^{(i)} - \frac{\lambda}{n}\theta_j$$

Regularized logistic regression

$$f(x,\theta) = g(\theta_0 x_0 + \theta_1 x_1 + \theta_2 x_2 + \ldots + \theta_m x_m)$$

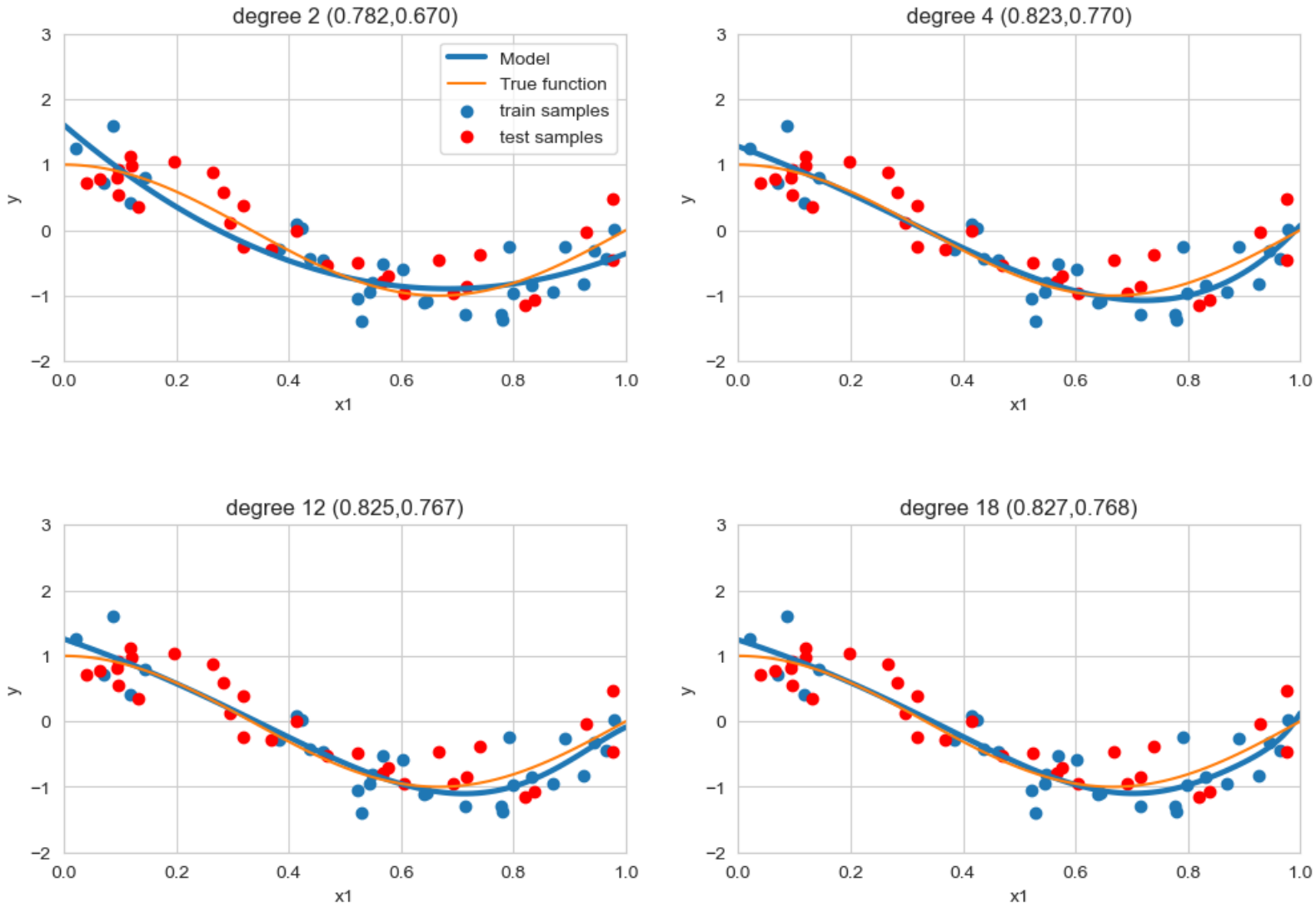
cost function

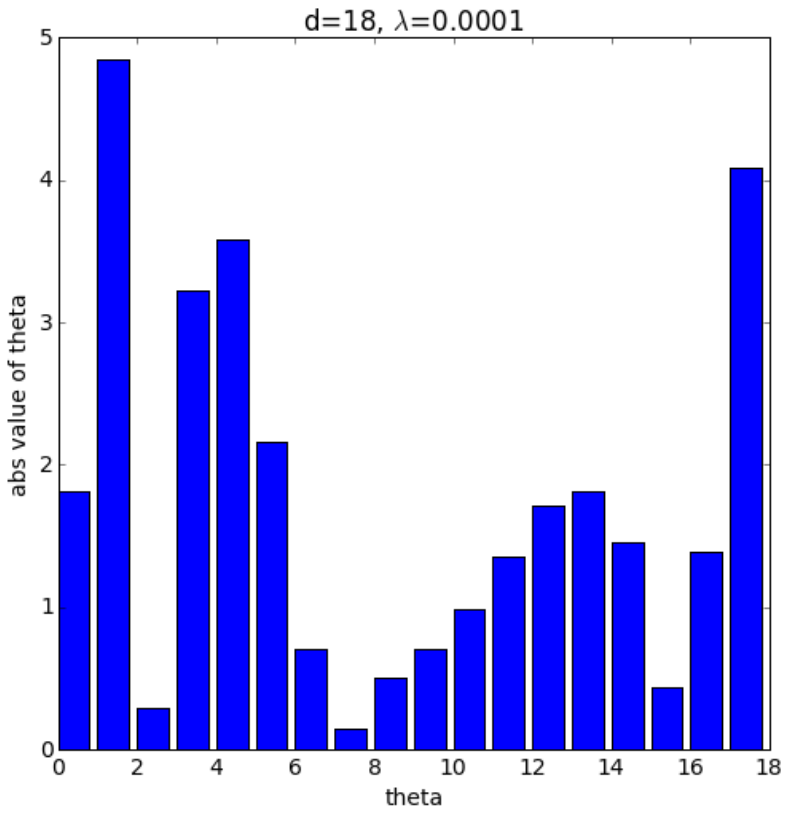
$$J(\theta) = -[\frac{1}{n}\sum_{i=1}^{n} y^{(i)} log(f(x^{(i)},\theta)) + (1-y^{(i)}) log(1-f(x^{(i)},\theta))]$$

$$J(\theta) = -[\frac{1}{n}\sum_{i=1}^{n} y^{(i)} log(f(x^{(i)},\theta)) + (1-y^{(i)}) log(1-f(x^{(i)},\theta))] + \frac{\lambda}{2m}\sum_{j=1}^{m}\theta_j^2$$

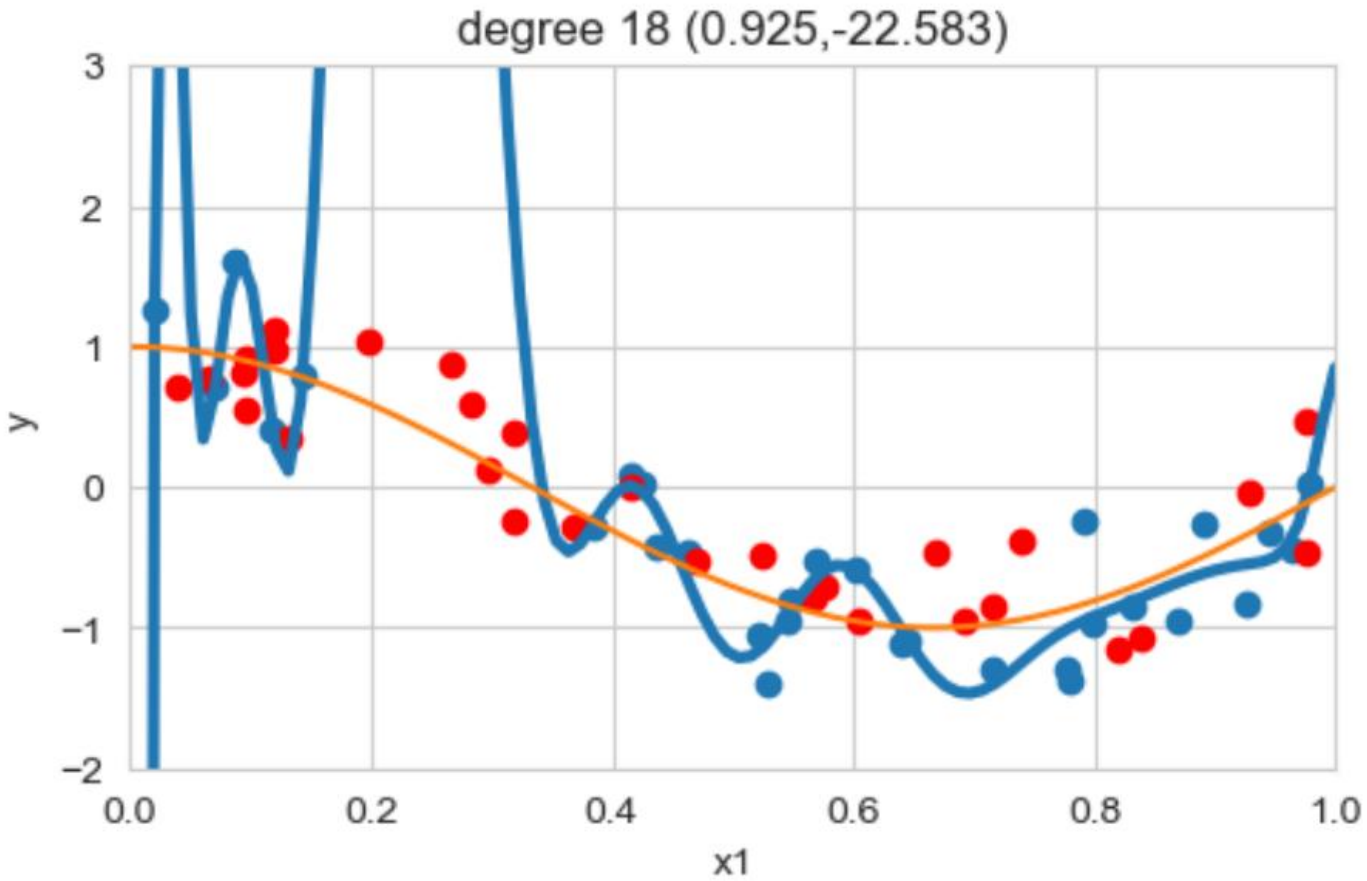
regularized cost function

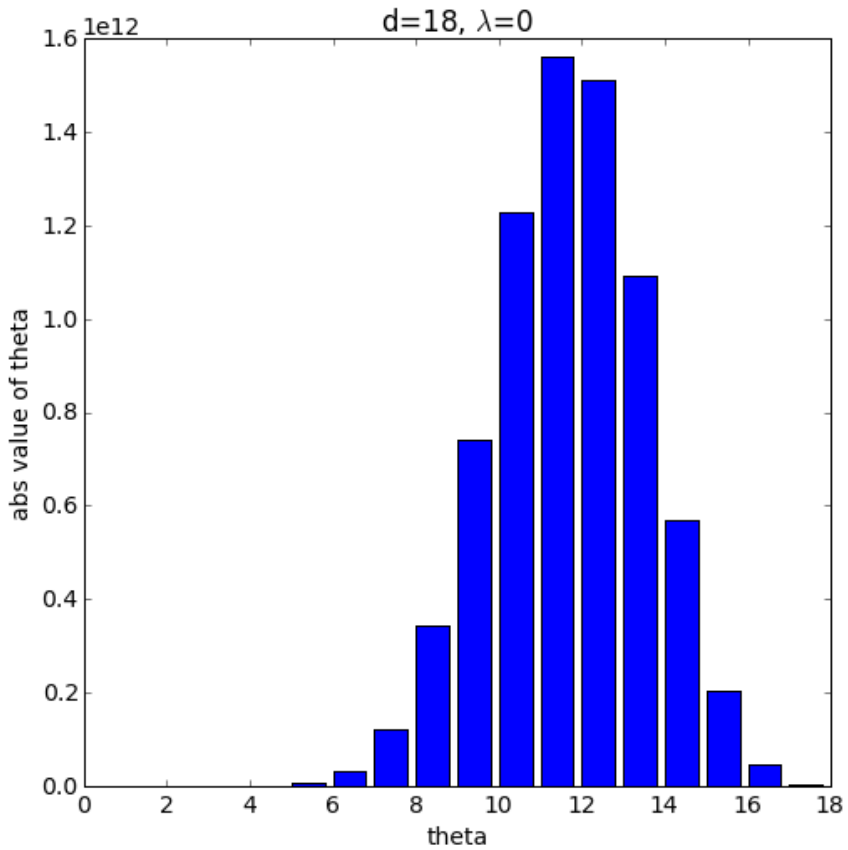
Regularized linear regression





Regularized linear regression





Support vector machine

Support Vector Machines (SVMs) are powerful classification algorithms that find the optimal hyperplane to separate data points into different classes with the **maximum margin**.

SVM uses support vectors, which are the closest data points to the decision boundary, to define and position the hyperplane, making it robust to outliers.

It can handle non-linearly separable data using the kernel trick, which transforms input features into higher-dimensional spaces (e.g., polynomial, radial basis function (RBF)) to make data linearly separable.

SVM is effective for both classification and regression (SVR) and works well with small to medium-sized datasets but can be computationally expensive for large datasets.

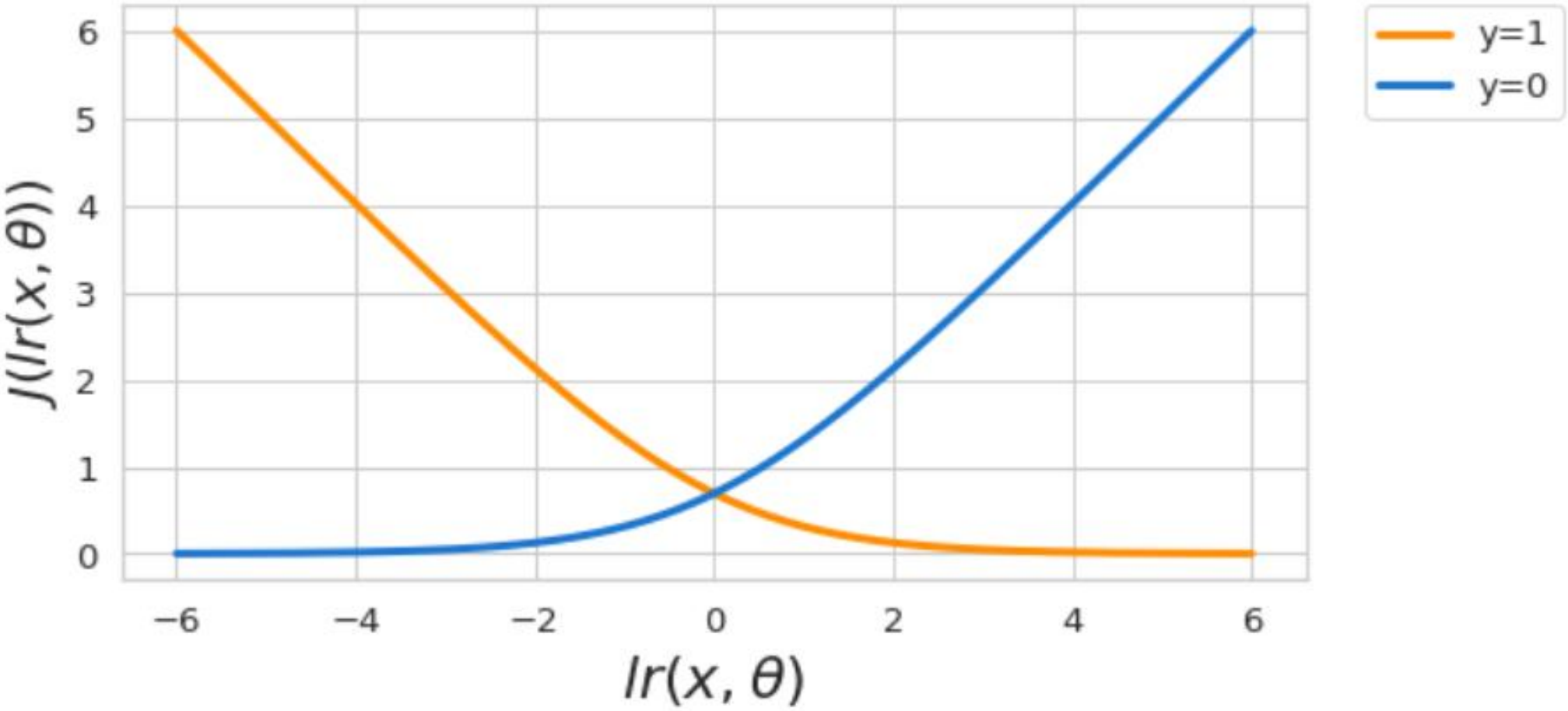
Logistic regression: cost (loss function)

$$J(\theta) = -[\frac{1}{n}\sum_{i=1}^{n} y^{(i)} log(f(x^{(i)}, \theta)) + (1 - y^{(i)}) log(1 - f(x^{(i)}, \theta))]$$

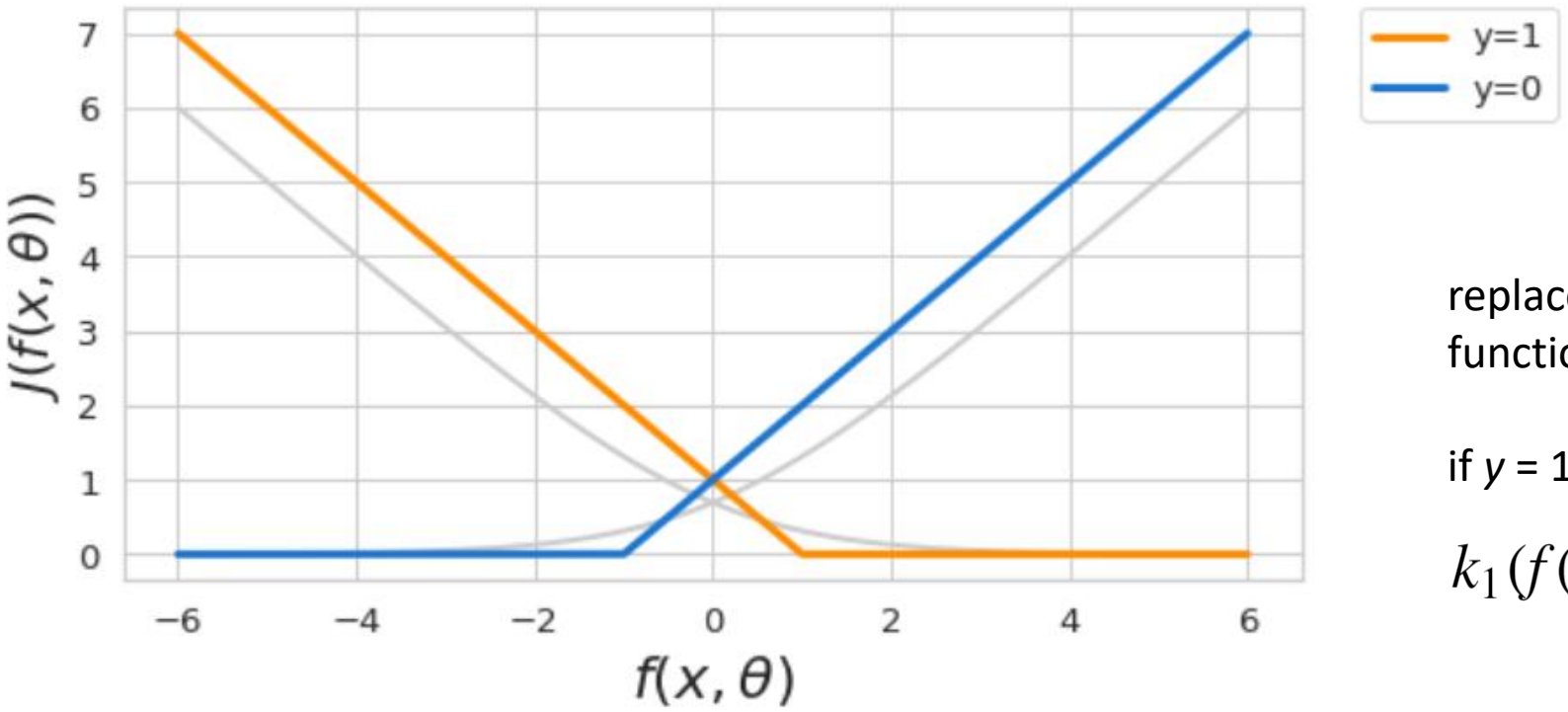
We know that $y^{(i)}$ is either 0 or 1. If $y^{(i)} = 1$ then the cost function $J(\theta)$ is incremented by $-log(f(x^{(i)}, \theta))$.

Similarly, if $y^{(i)} = 0$ then the cost function $J(\theta)$ is incremented by $-log(1 - f(x^{(i)}, \theta))$.

Logistic regression: cost (loss function)



Support vector machine: cost (loss function)

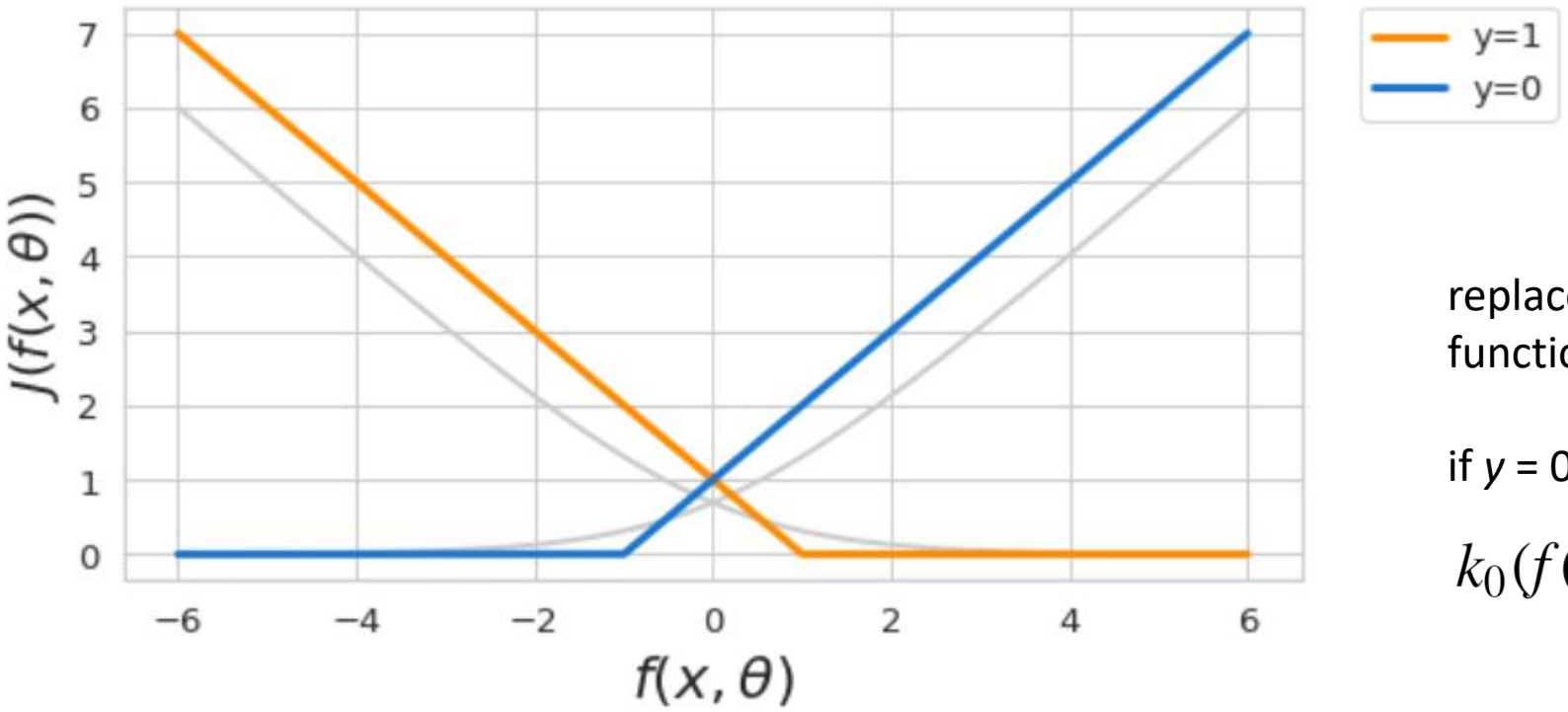


replace cost function by piecewise linear function

if y = 1 then the contribution to the cost is

$$k_1(f(x,\theta)) = max(0, 1 - f(x,\theta))$$

Support vector machine: cost (loss function)



replace cost function by piecewise linear function

if y = 0 then the contribution to the cost is

$$k_0(f(x,\theta)) = max(0, 1 + f(x,\theta))$$

Support vector machine: cost (loss function)

Fit a linear model

$$f(x, \theta) = \theta' x$$

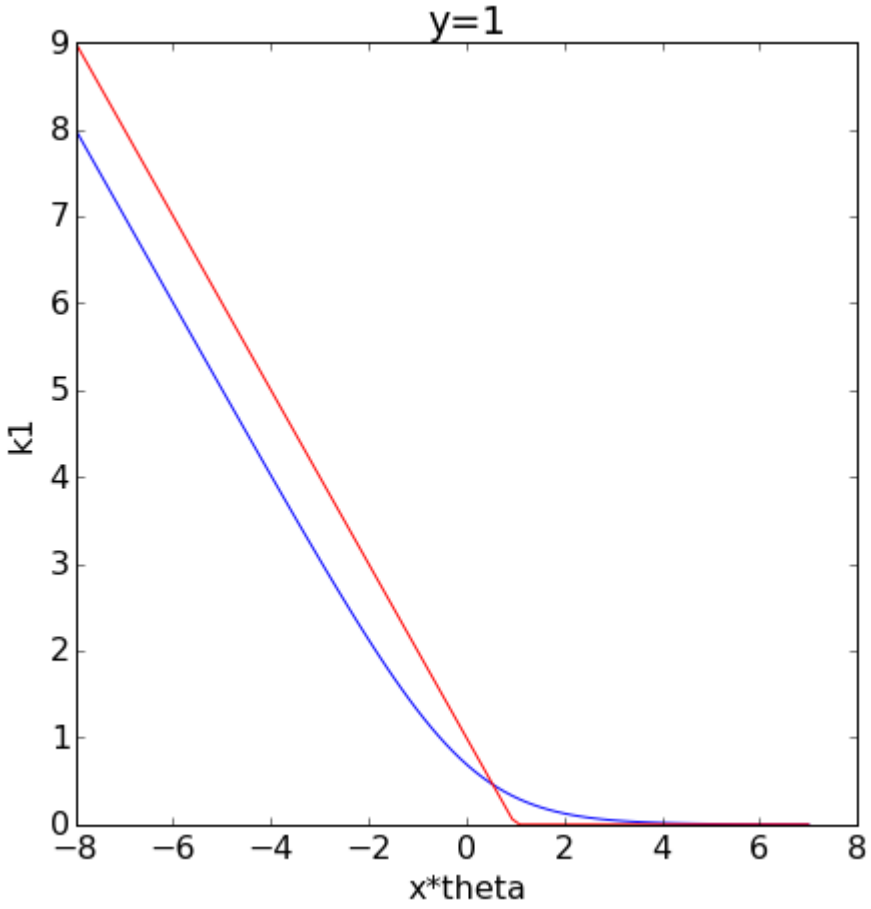
such that

$$J(\theta) = C \left[\sum_{i=1}^{n} y^{(i)} k_1(\theta' x^{(i)}) + (1 - y^{(i)}) k_0(\theta' x^{(i)}) \right] + \frac{1}{2} \sum_{j=1}^{m} \theta_j^2$$

with $k_1(\theta' x) = max(0, 1 - \theta' x)$ and $k_0(\theta' x) = max(0, 1 + \theta' x)$

is minimized.

Support vector machine: cost (loss function)

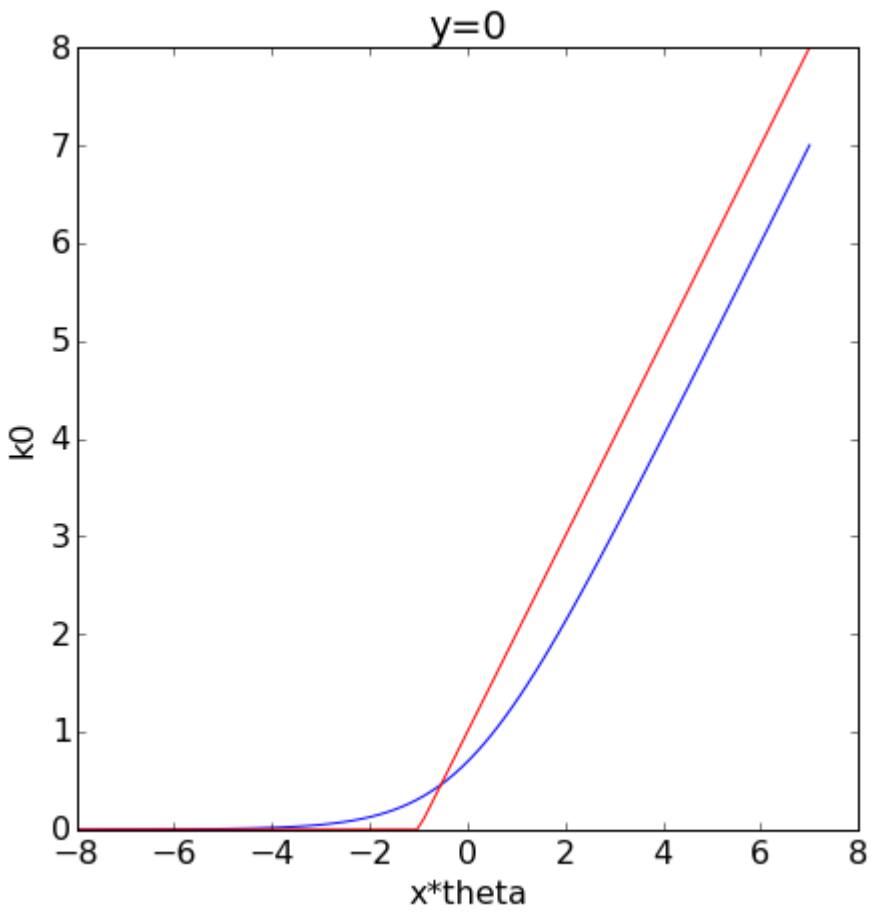


In this case the contribution to the cost needs to be small when the model predicts high values (>0) and large when the model predicts low values (<0).

For SVMs we see that the contribution to cost decreases linearly and becomes zero when

$$\theta' x \geq 1$$

Support vector machine: cost (loss function)



In this case the contribution to the cost needs to be large when the model predicts high values (>0) and small when the model predicts low values (<0).

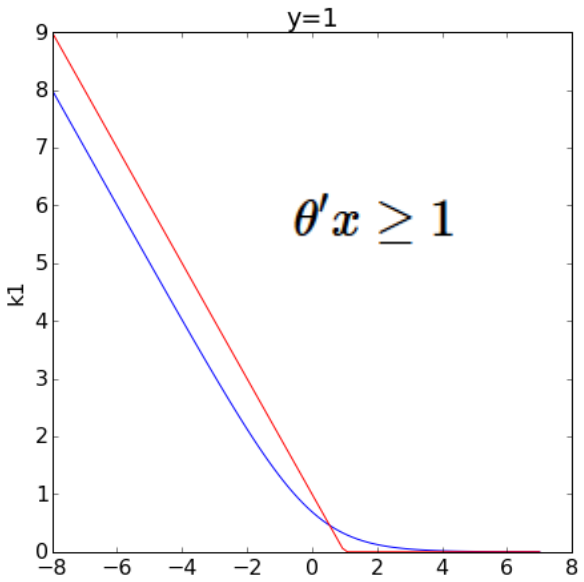
For SVMs we see that the contribution to the cost is zero for

$$\theta' x \leq -1$$

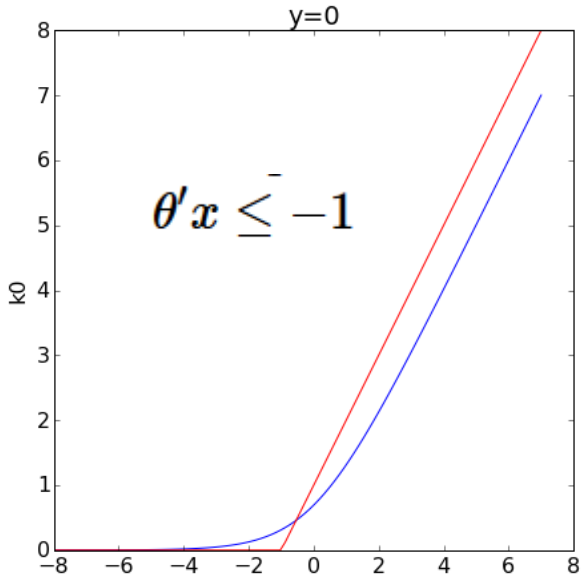
and then increases linearly.

Support vector machine: cost (loss function)

$$J(\theta) = C\left[\sum_{i=1}^{n} y^{(i)} k_1(\theta' x^{(i)}) + (1 - y^{(i)}) k_0(\theta' x^{(i)})\right] + \frac{1}{2}\sum_{j=1}^{m} \theta_j^2$$



$f(x, \theta)$



$f(x, \theta)$

Consider a train set with two classes that are perfectly linearly separable.

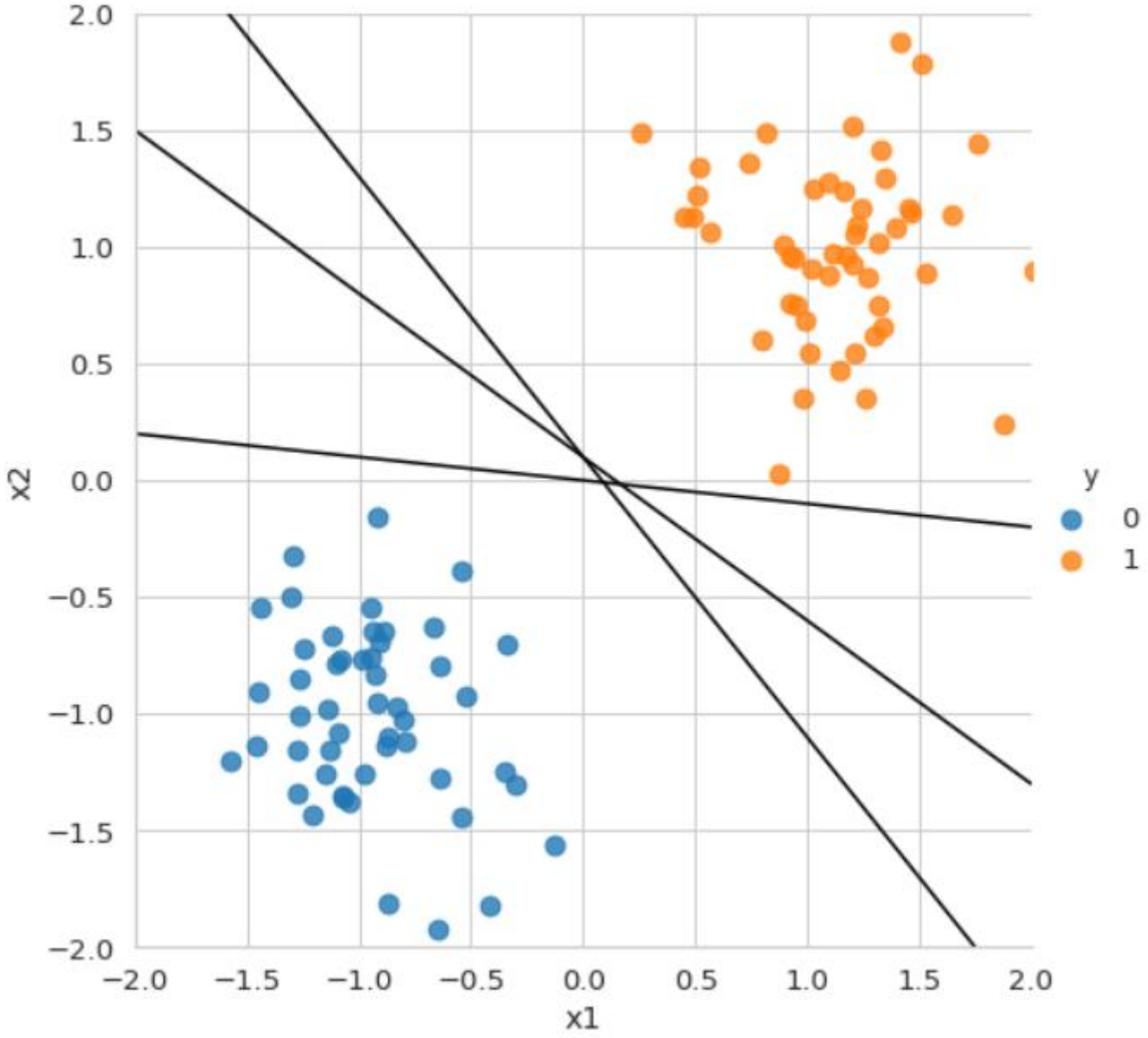
The piecewise cost function can be made zero.

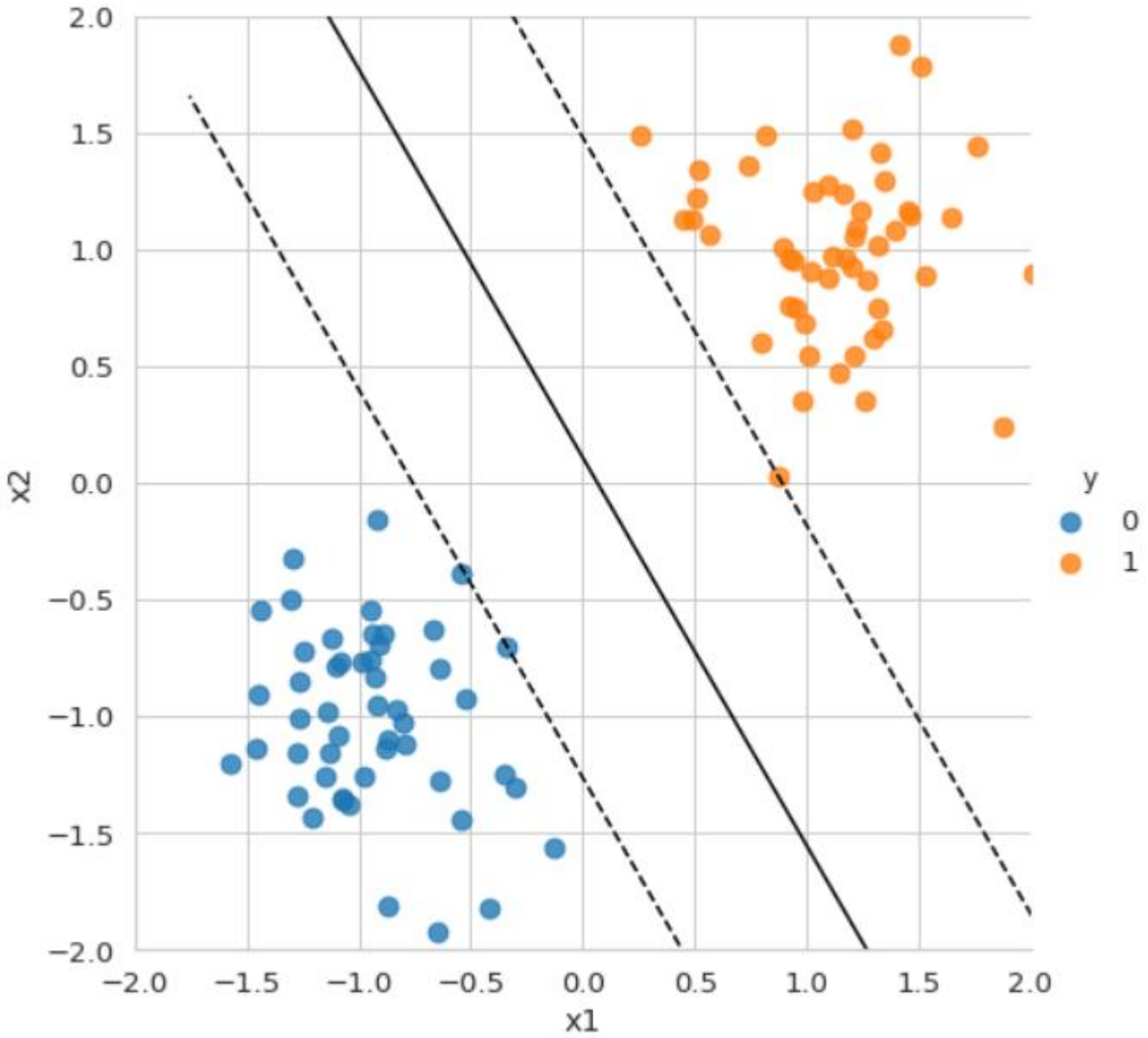
The SVM objective can be written as

$$\min_{\theta} \frac{1}{2}\sum_{j=1}^{m} \theta_j^2$$

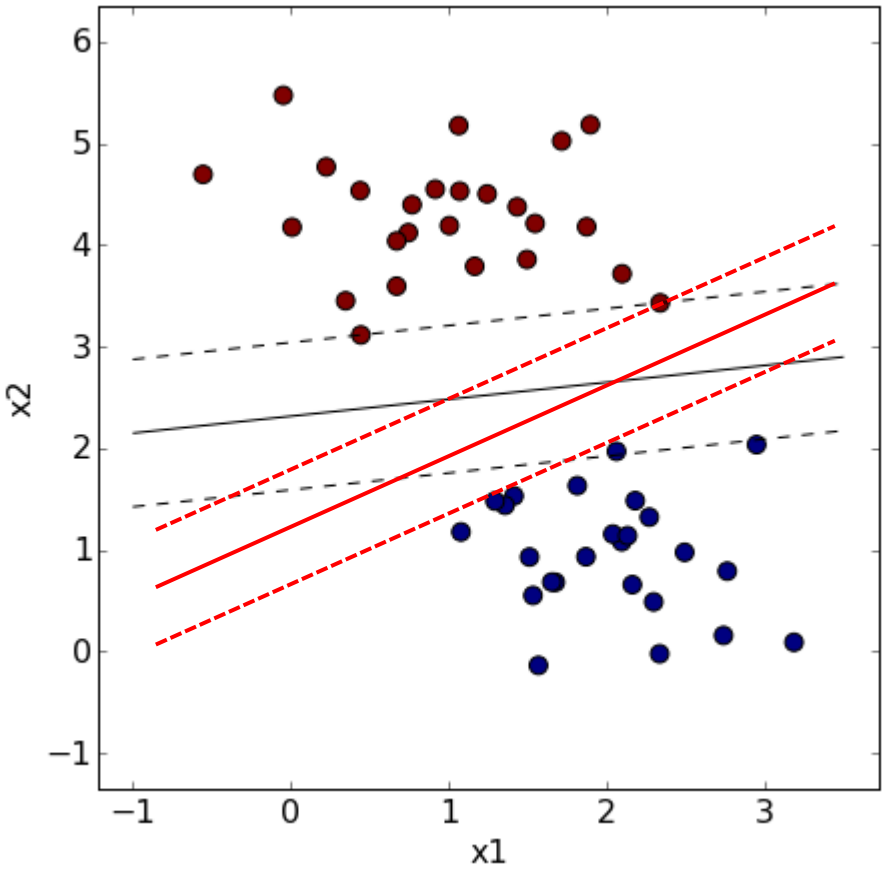
$$\text{subject to } \theta' x^{(i)} \geq 1 \text{ if } y^{(i)} = 1$$

$$\theta' x^{(i)} \leq -1 \text{ if } y^{(i)} = 0$$

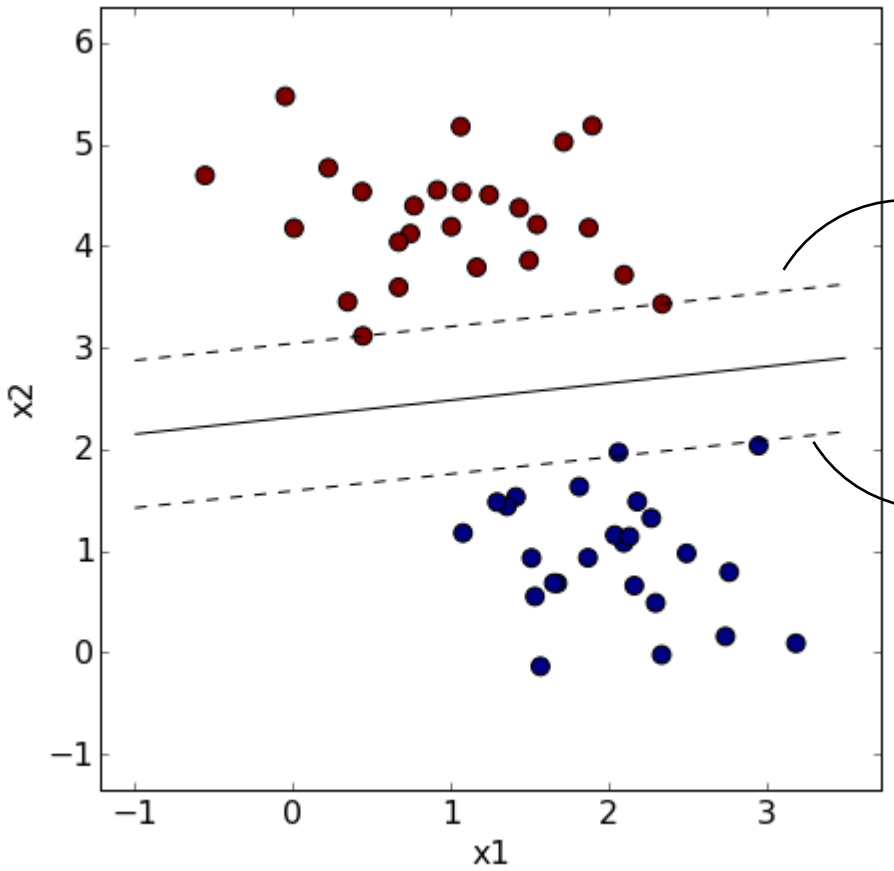

2.0
1.5
1.0
0.5
0.0
−0.5
−1.0
−1.5
−2.0
x2
−2.0
−1.5
−1.0
−0.5
0.0
0.5
1.0
1.5
2.0
x1
y
0
1


2.0
1.5
1.0
0.5
0.0
−0.5
−1.0
−1.5
−2.0
x2
−2.0
−1.5
−1.0
−0.5
0.0
0.5
1.0
1.5
2.0
x1
y
0
1

Support vector machine: large margin classifier



Support vector machine: large margin classifier



$$\theta' x^{(i)} = 1$$

$$\theta' x^{(i)} = -1$$

red points statisfy $\theta' x^{(i)} \geq 1$

blue points satisfy $\theta' x^{(i)} \leq -1$

distance bewteen two dashed lines is

$$\frac{2}{\sqrt{\sum_{j=1}^{m} \theta_j^2}}$$

SVM objective was

$$\min_{\theta} \frac{1}{2} \sum_{j=1}^{m} \theta_j^2$$

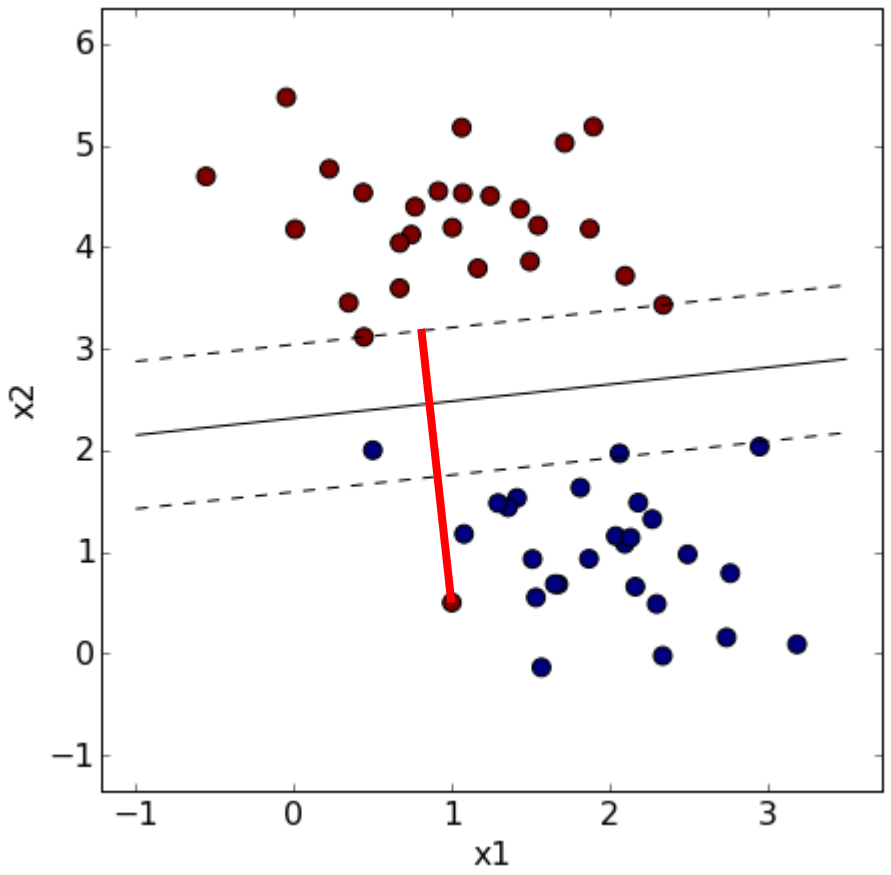
Support vector machine

Soft margin SVM allows some misclassification by introducing **slack variables** (ξ) to handle overlapping classes, making the model more flexible when data is not perfectly separable.

It balances margin maximization and classification error using a **regularization parameter** (C), where a small C allows more misclassifications (wider margin), and a large C forces stricter separation (narrower margin).

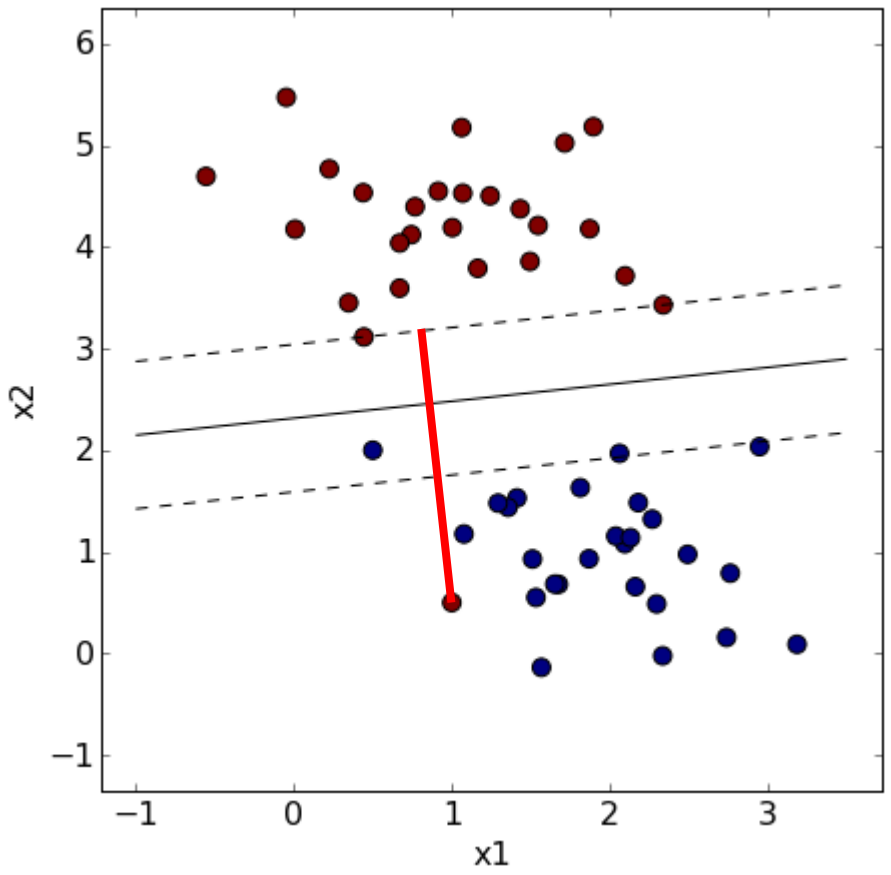
Soft margin SVM improves generalization by preventing overfitting in noisy or complex datasets while still maintaining a robust decision boundary.

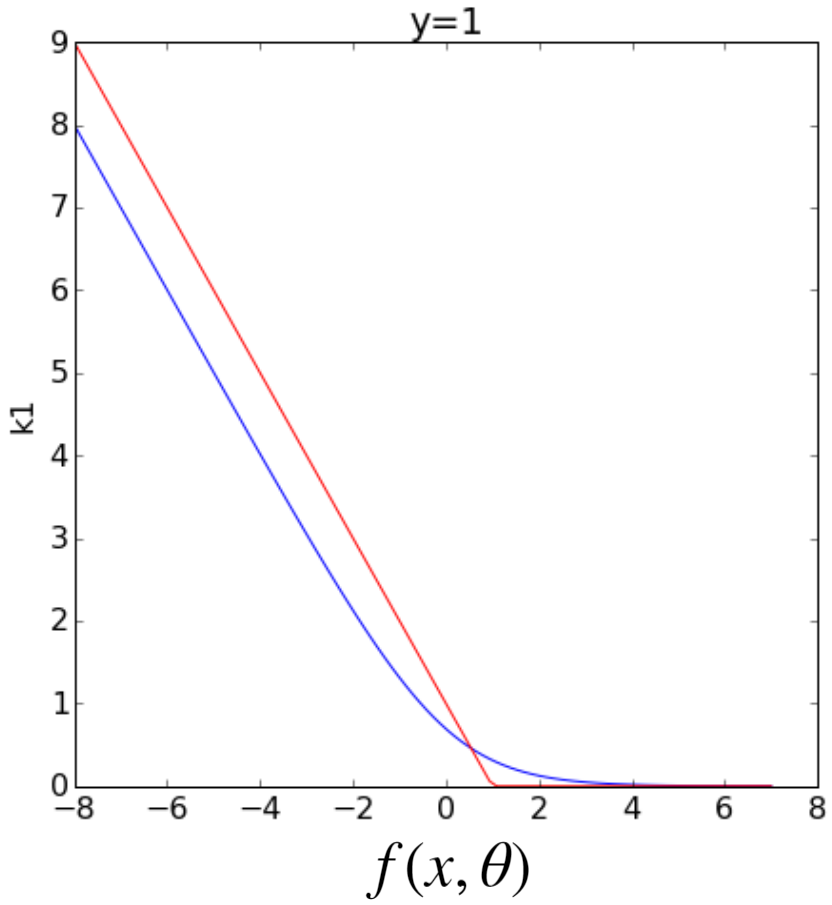
Support vector machine: soft margin



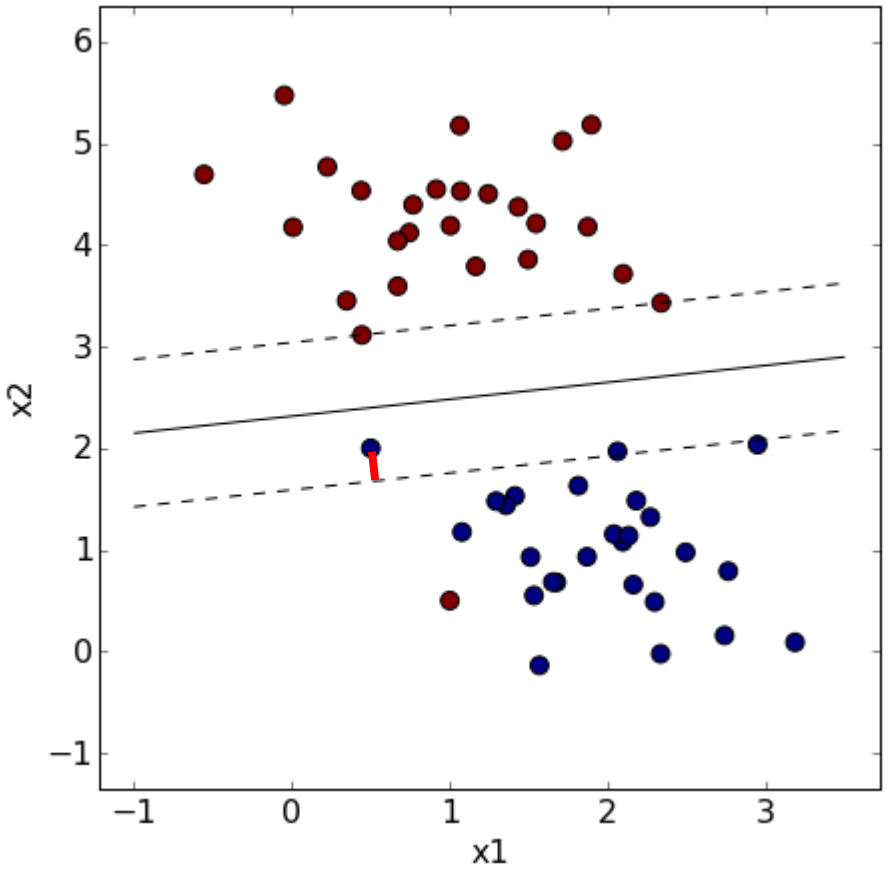
$$\theta' x^{(i)} \geq 1 \text{ if } y^{(i)} = 1$$
$$\theta' x^{(i)} \leq -1 \text{ if } y^{(i)} = 0$$

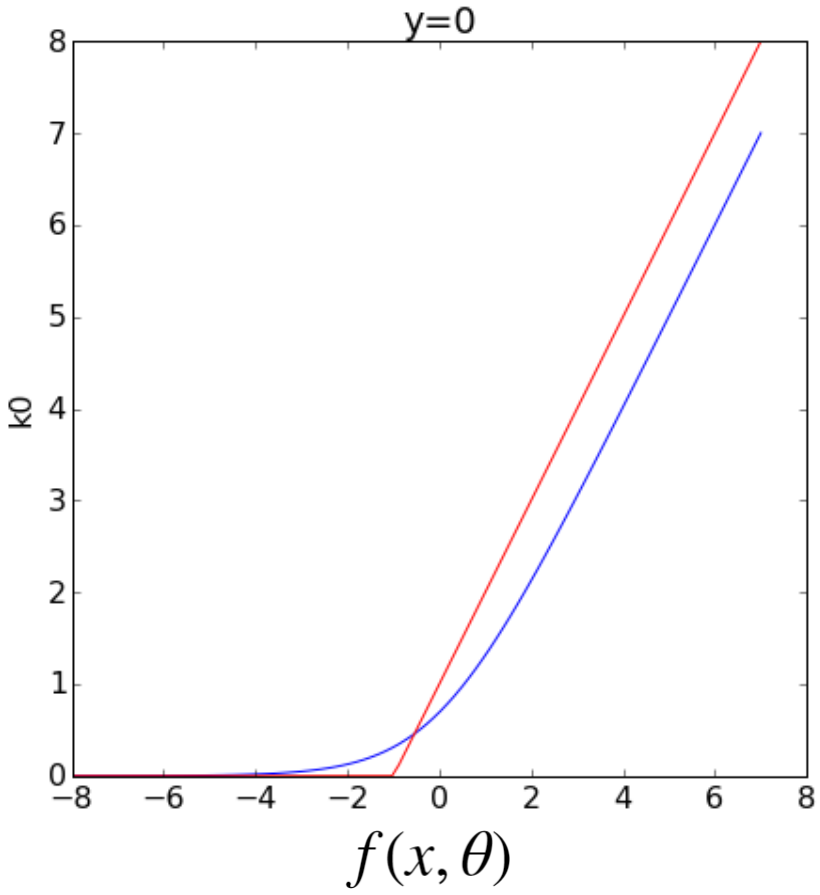
Support vector machine: soft margin





Support vector machine: soft margin



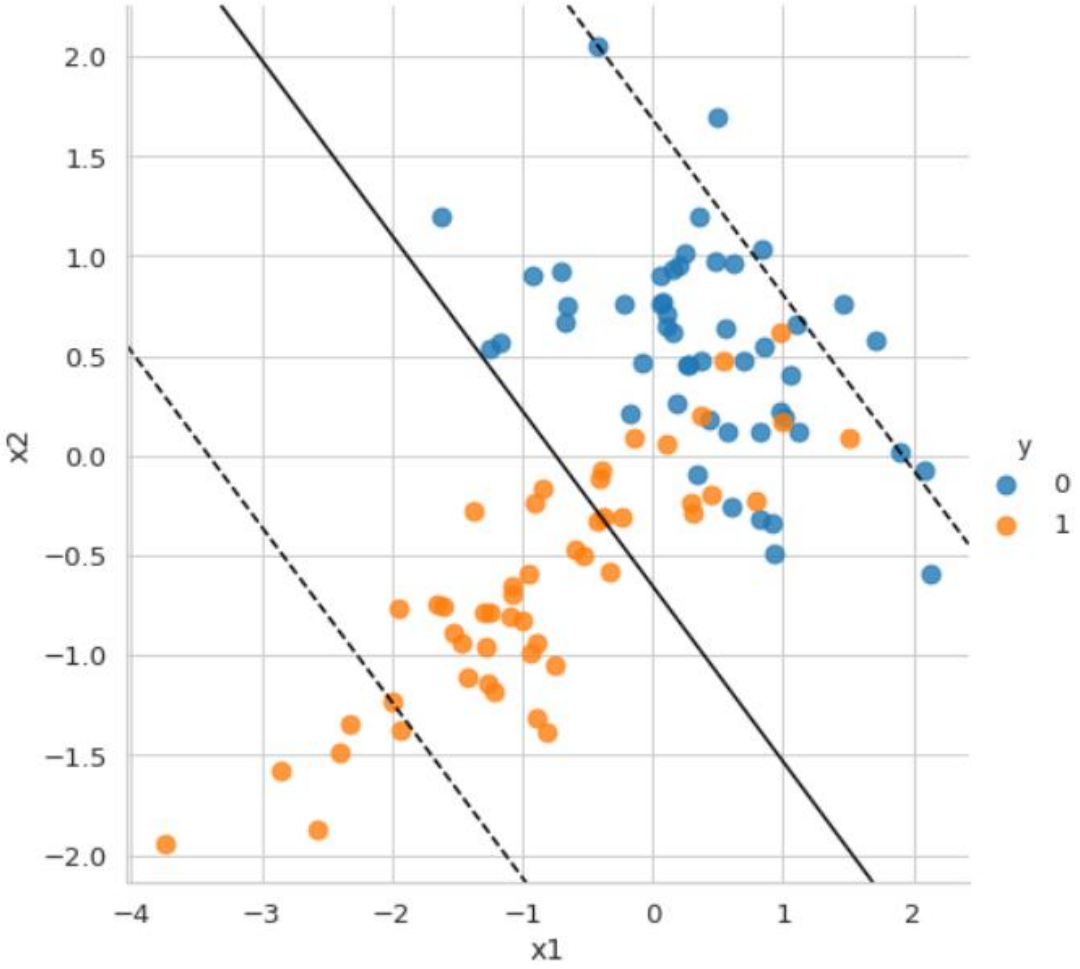


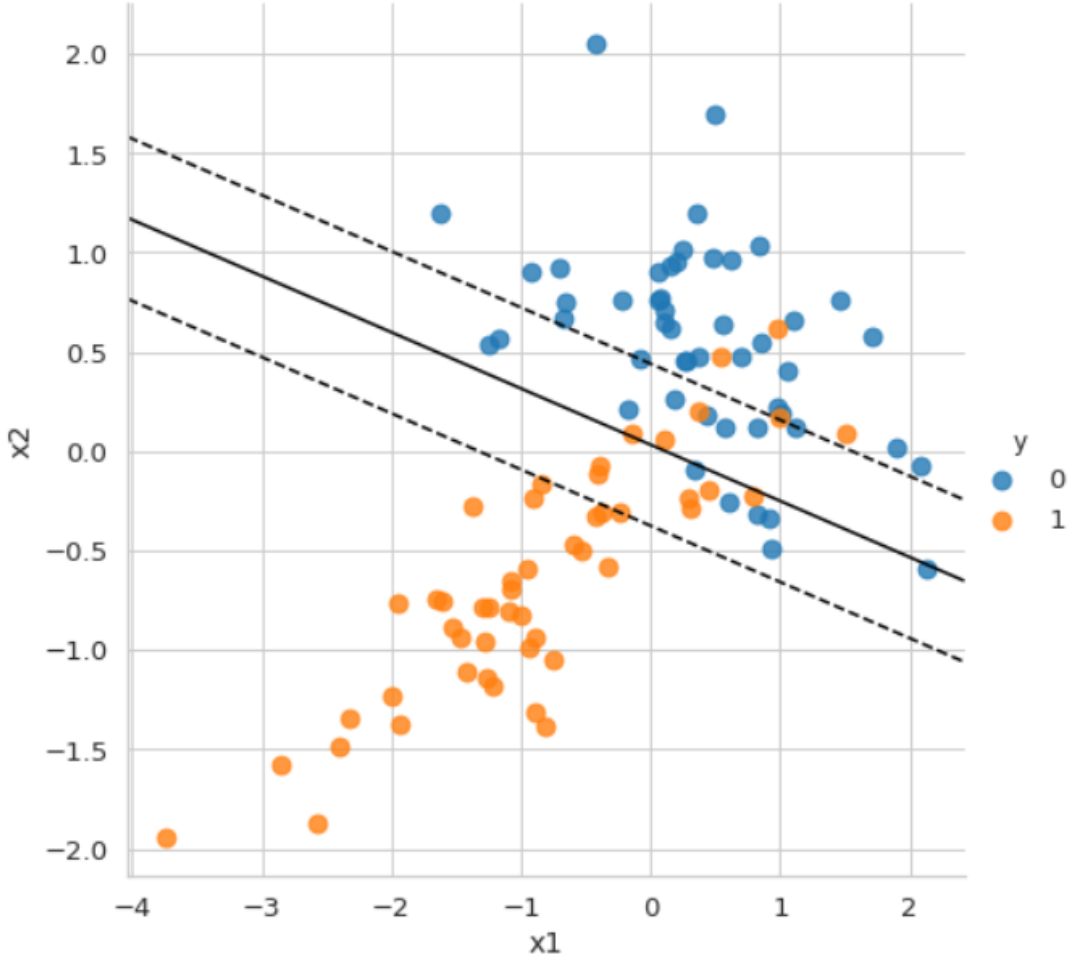
Support vector machine: soft margin

$$\min_{\theta,\xi} \quad \frac{1}{2}\sum_{j=1}^{m}\theta_j^2 + C\sum_{i=1}^{n}\xi_i$$

$$\text{subject to} \quad \begin{cases} \theta' x^{(i)} \geq 1 - \xi_i & \text{if } y^{(i)} = 1 \\ \theta' x^{(i)} \leq -1 + \xi_i & \text{if } y^{(i)} = 0 \\ \xi_i \geq 0 & \forall i \end{cases}$$

Support vector machine: regularization hyperparameter





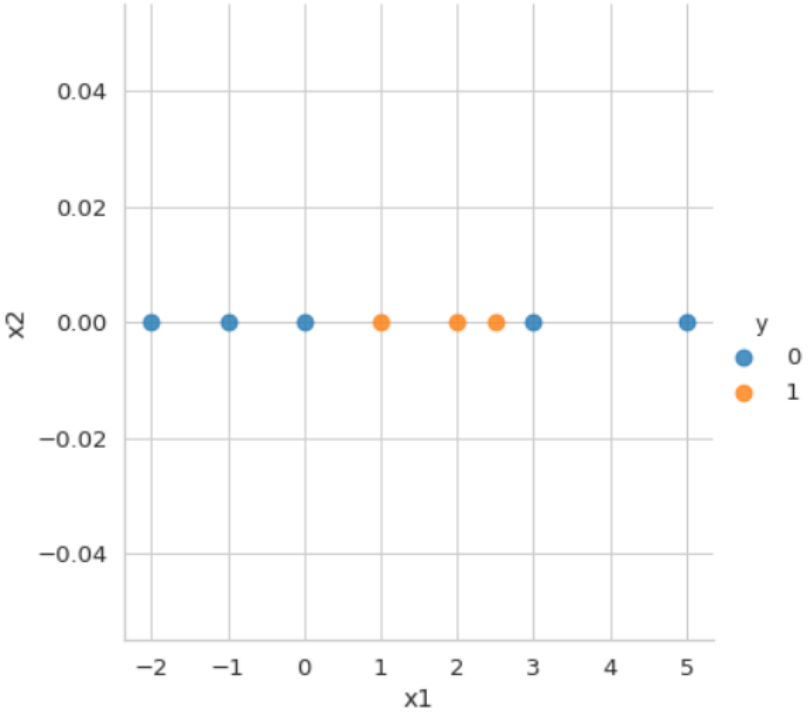
Support vector machine: kernels

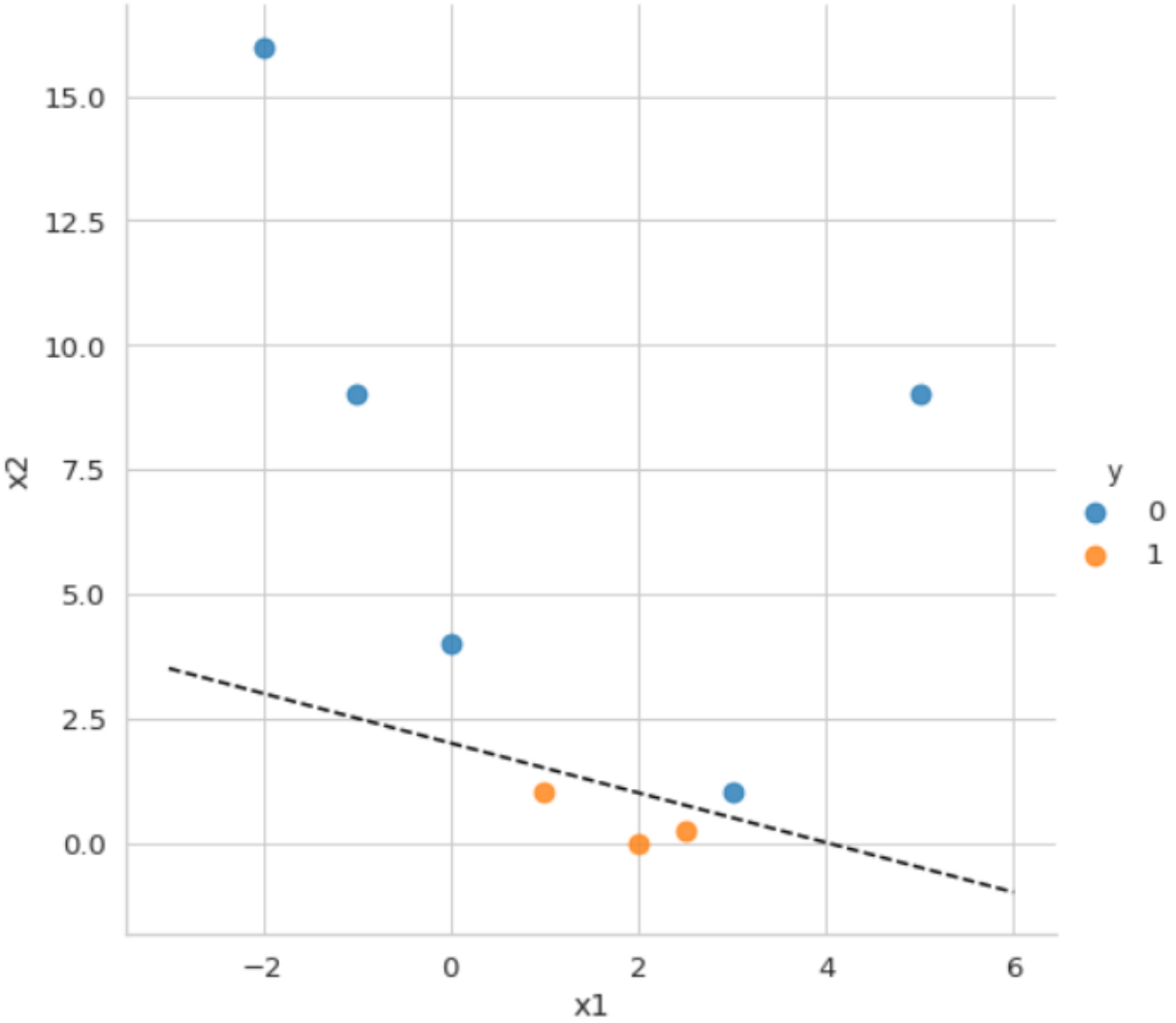
The kernel trick allows SVM to handle non-linearly separable data by **transforming** input features into a higher-dimensional space where a linear decision boundary can be found.

Common kernel functions include Polynomial, Radial Basis Function (RBF), and Sigmoid, each mapping data to a **different feature space** to capture complex relationships.

It **avoids explicitly computing** high-dimensional transformations, making computations efficient while still benefiting from the power of non-linear feature mappings.

Support vector machine: kernels





Support vector machine: kernels

SVMs can also be formulated as a linear function of the samples (dual form) instead of the features as

$$f(x, \theta) = \sum_{i=1}^{n} \theta_i (x \cdot x^{(i)}) + \theta_0$$

where a model parameter is associated with a datapoint instead of a feature.

If we apply a polynomial transformation $\phi(x)$ we obtain

$$f(x, \theta) = \sum_{i=1}^{n} \theta_i \big(\phi(x) \cdot \phi(x^{(i)})\big) + \theta_0$$

Support vector machine: kernels

For **degree 2**, you generate:

- All original features: $x_1, x_2, \ldots, x_{100}$ → 100
- All squared terms: $x_1^2, x_2^2, \ldots, x_{100}^2$ → 100
- All pairwise interactions: $x_1x_2, x_1x_3, \ldots, x_{99}x_{100}$

The number of **unique pairs** (without repetition) is:

$$\binom{100}{2} = \frac{100 \times 99}{2} = 4,950$$

At **degree 3**, you start getting terms like:

- $x_1x_2x_3$, $x_4^2x_5$, etc.

The number of 3-way combinations from 100 features is:

$$\binom{100+3-1}{3} = \binom{102}{3} = 171,700$$

Degree 4? **Over 4 million** features.

Support vector machine: kernels

Since the model only uses the dot product of the transformed feature vectors

$$f(x, \theta) = \sum_{i=1}^{n} \theta_i \big(\phi(x) \cdot \phi(x^{(i)})\big) + \theta_0$$

we can replace that dot product by a kernel function

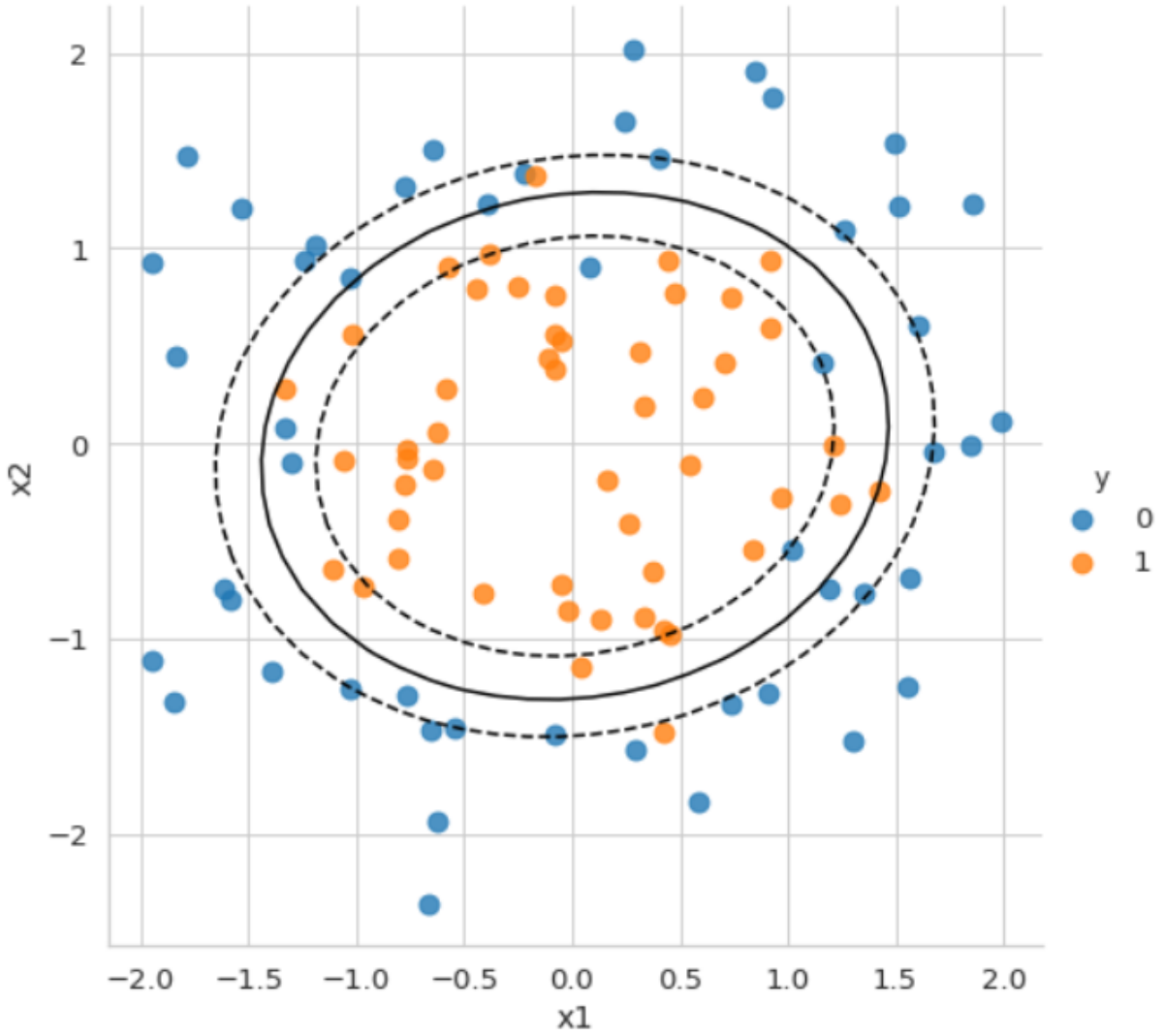
$$K(x, x^{(i)}) = \phi(x) \cdot \phi(x^{(i)})$$

such that the model becomes

$$f(x, \theta) = \sum_{i=1}^{n} \theta_i K(x, x^{(i)}) + \theta_0$$

The data points $x^{(i)}$ for which $\theta_i > 0$ are called the support vectors.

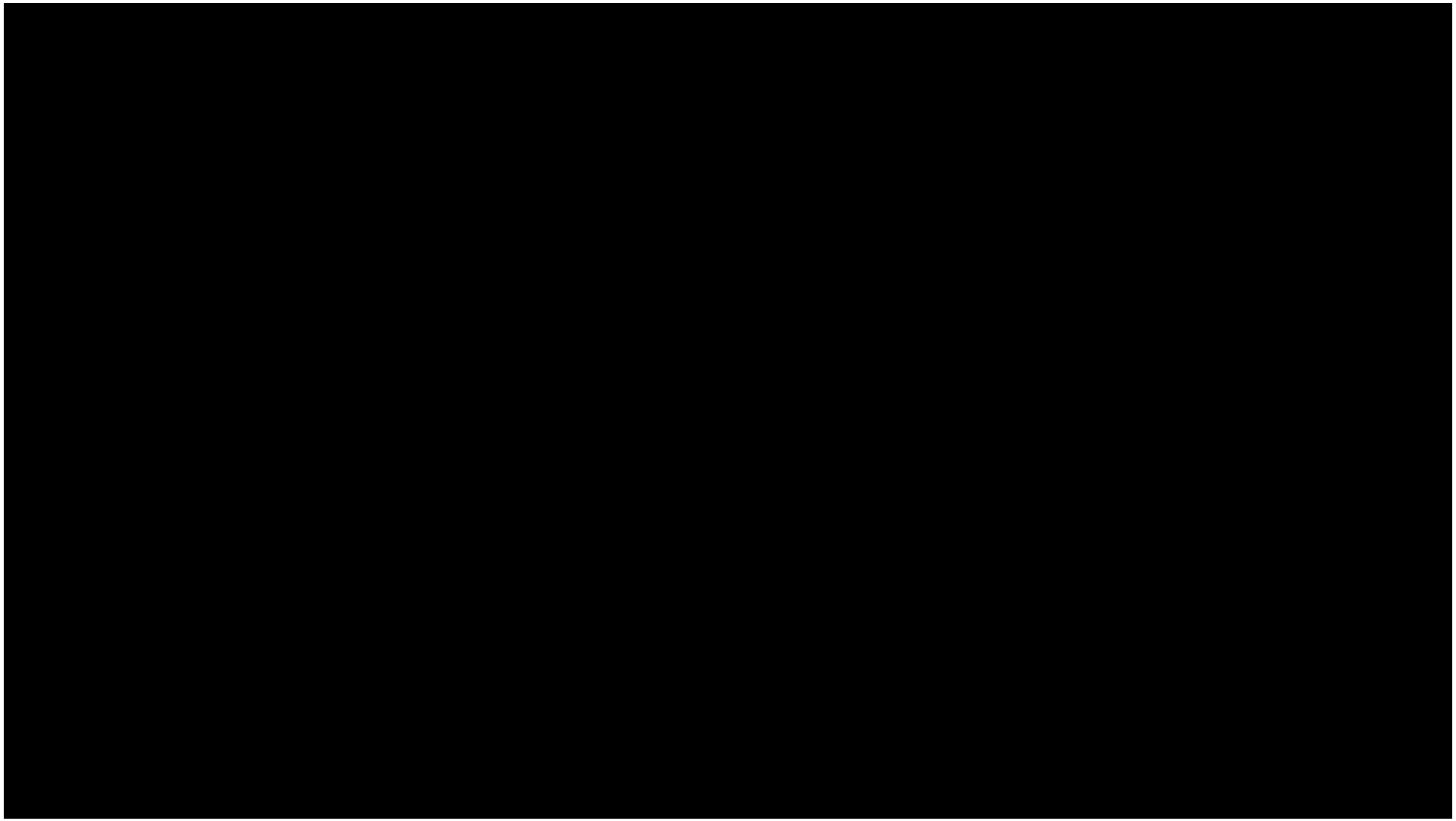
Support vector machine: polynomial kernel



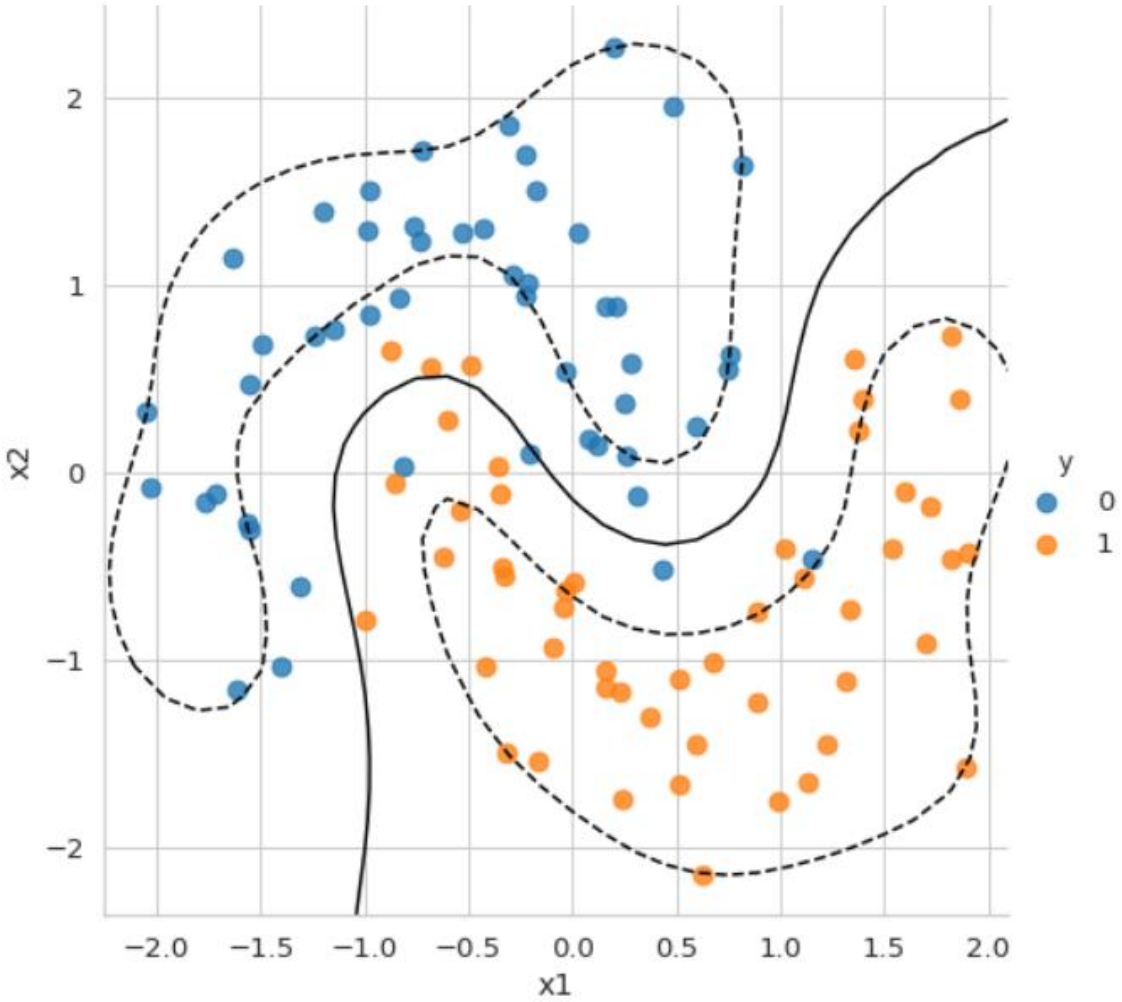
$$f(x, \theta) = \sum_{i=1}^{n} \theta_i K(x, x^{(i)}) + \theta_0$$

$$K(x^{(i)}, x^{(j)}) = (x^{(i)} \cdot x^{(j)} + c)^d$$

Support vector machine: polynomial kernel



Support vector machine: RBF kernel



$$f(x, \theta) = \sum_{i=1}^{n} \theta_i K(x, x^{(i)}) + \theta_0$$

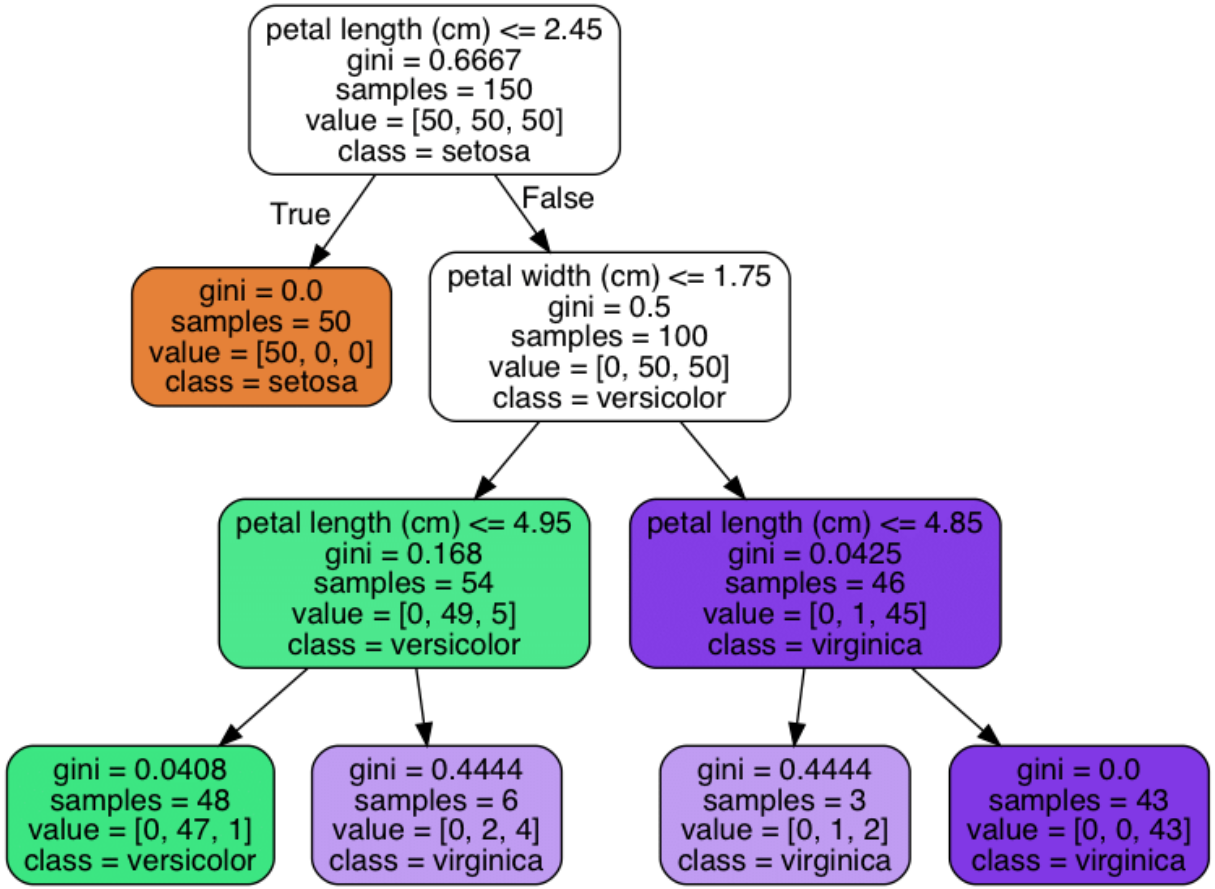
$$K(x^{(i)}, x^{(j)}) = \exp\left[-\frac{||x^{(i)} - x^{(j)}||^2}{2\sigma^2}\right]$$

Ensemble methods

Decision trees serve as the foundation for **ensemble methods** like Random Forest and Gradient Boosting, which combine multiple trees to improve accuracy and robustness.

Ensemble learning reduces **variance and bias**, as bagging (e.g., Random Forest) decreases overfitting, while boosting (e.g., AdaBoost, XGBoost) enhances weak learners by correcting misclassified instances iteratively.

Decision tree



advantages:

ease of interpretation
handles continuous and discrete features
invariant to monotone transformation of features
variable selection automated
low bias (deep trees)

disadvantages:

high variance
overfitting

Bias and variance

Bias refers to the error introduced by approximating a complex problem with a simple model, leading to underfitting when the model fails to capture the underlying patterns in the data.

Variance measures the model's sensitivity to small fluctuations in the training data, causing overfitting when the model learns noise instead of general patterns.

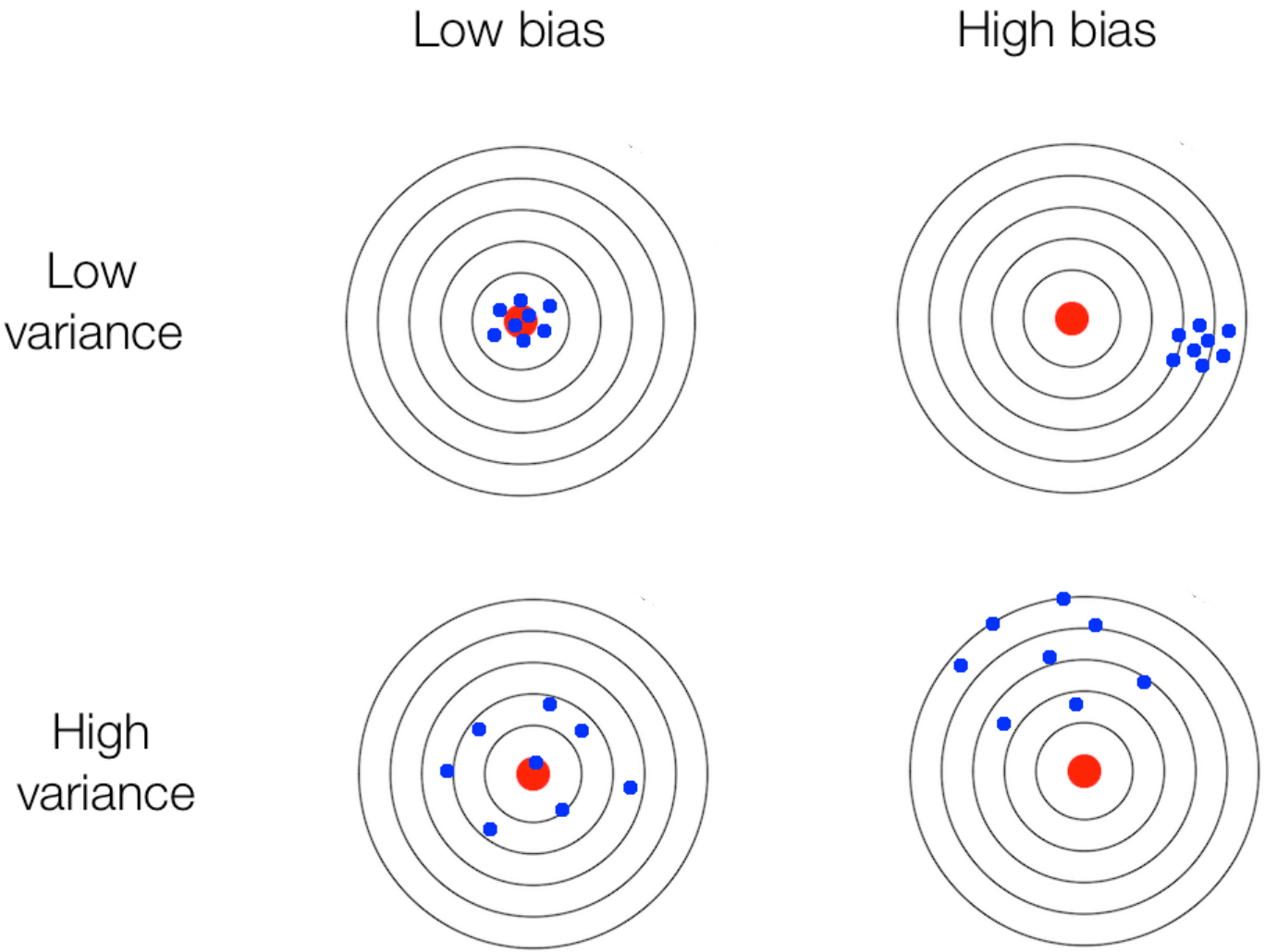
The **bias-variance tradeoff** is a fundamental concept in machine learning, where reducing bias increases variance and vice versa, requiring a balance for optimal model performance.

Bias and variance

Error due to bias

Error due to variance

Irreducible error

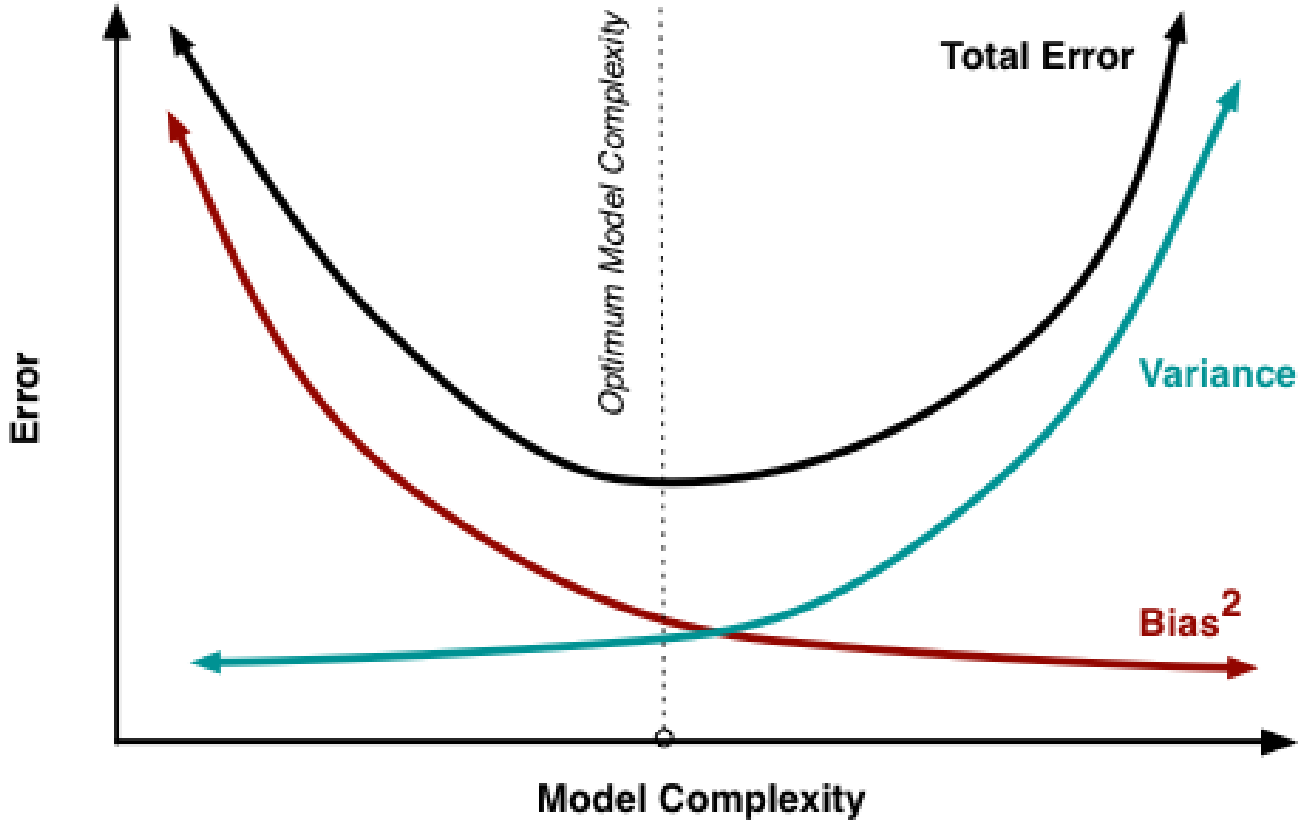


Bias and variance

Error due to bias

Error due to variance

Irreducible error



Bagging and boosting

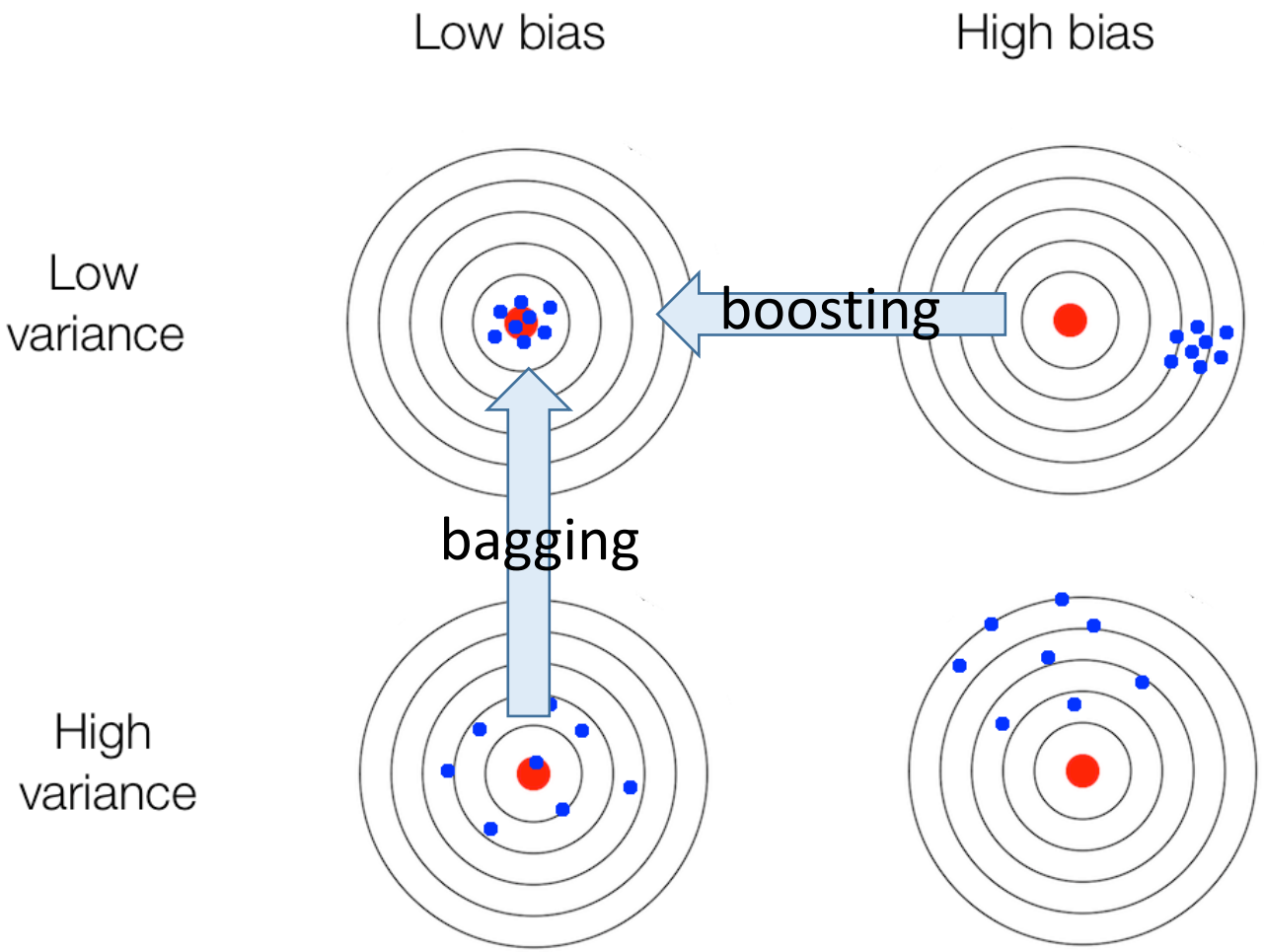
Bagging (Bootstrap Aggregating) reduces variance by training multiple models on different random subsets of the data, then averaging their predictions (for regression) or using majority voting (for classification), as seen in Random Forest.

Boosting sequentially trains models where each model corrects the errors of the previous one, improving weak learners incrementally, as used in AdaBoost, Gradient Boosting, and XGBoost.

Bagging is effective for high-variance models (e.g., decision trees) by stabilizing predictions, while boosting is useful for reducing bias by focusing on difficult-to-classify instances.

Both methods enhance predictive accuracy in ensemble learning, with bagging excelling in variance reduction and boosting providing strong performance on complex datasets.

Bagging and boosting



Bagging

Simulate the notion of different train set samples:

1. sample data points from train set to create new train set
2. fit low bias high variance model on new train set
3. repeat steps (1) and (2) *T* times
4. average the predictions of the *T* models

$$\hat{f}(x, \theta) = \frac{1}{T} \sum_{t=1}^{T} f_t(x, \theta)$$

Bagging: Random Forest

Train set contains *n* data points with *m* features.

Construct *T* low bias high variance decision trees by following these steps:

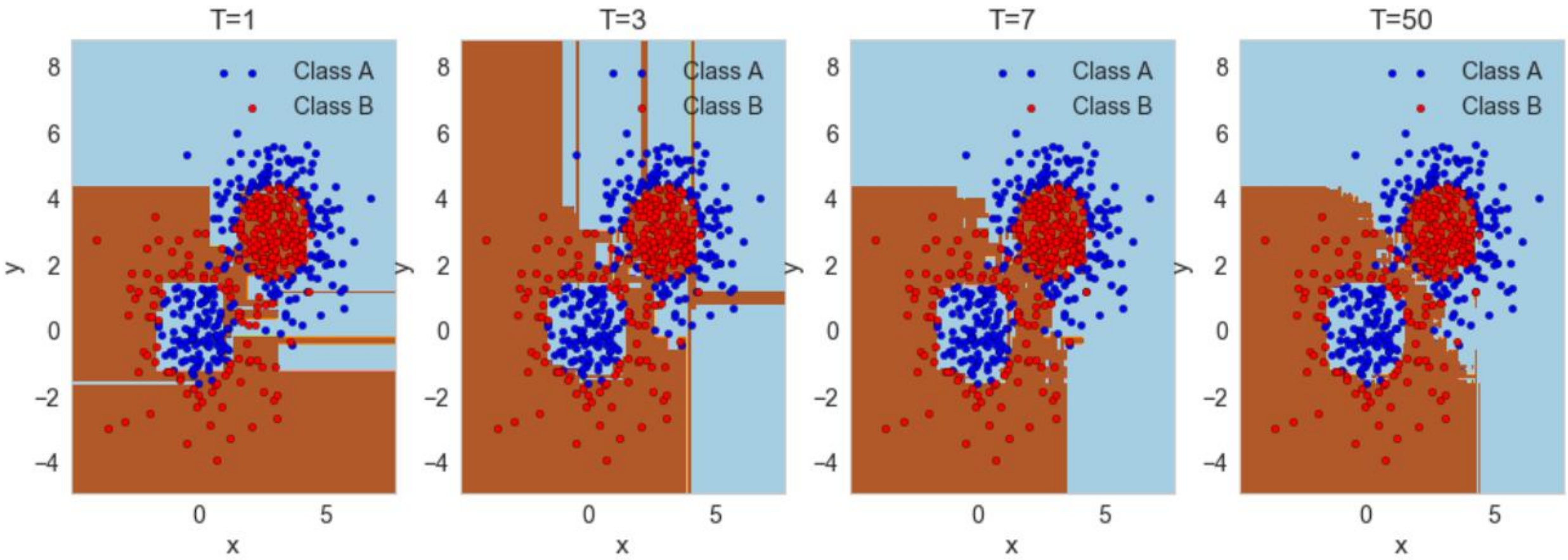
Sample ***n*** data points at random **with replacement** from the train set.

At each node, select $h \ll m$ features at random and compute the best split using only these *h* features.

Each tree is grown to the largest extent possible. There is no pruning or early stopping.

Step 3 ensures that the bagged models are low bias by learning deep complex decision trees.

Bagging: Random Forest



Boosting

reduce the bias of a high bias low variance model

turning an ensemble of weak learners into a strong learner

the meta-model is additive, i.e. Adaboost:

$$\hat{f}(x, \theta) = \sum_{t=1}^{T} \alpha_t f_t(x, \theta)$$

Boosting: Adaboost

$$\boxed{y_i \in \{-1, +1\}}$$

Initialize the weights $D_1(i) = 1/n, i = 1, 2, \ldots, n$.

For $t = 1$ to T:

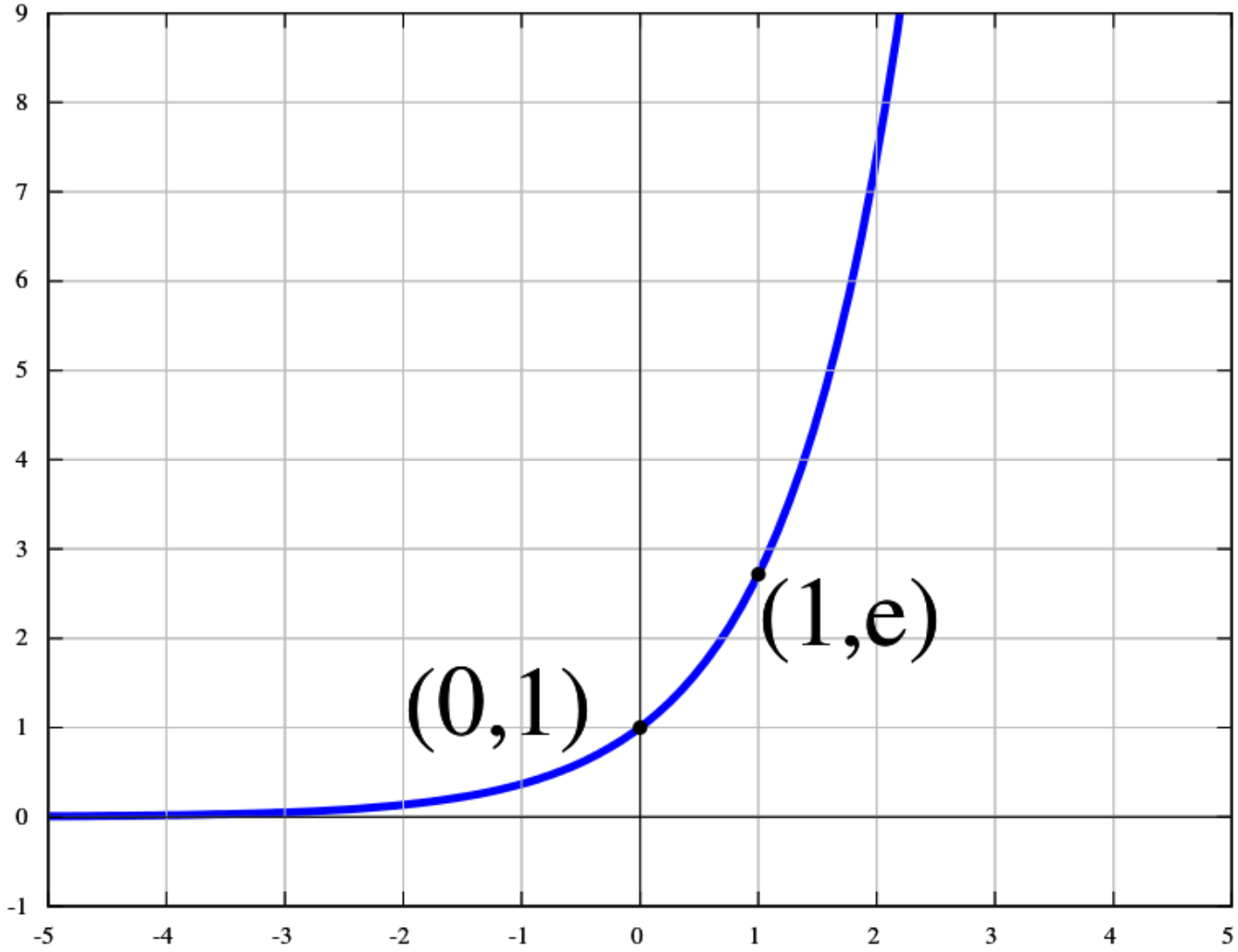
1. Fit a weak classifier $f_t(x, \theta)$ to the trainset data using weights $D_1(i)$.
2. Set $\alpha_t = \frac{1}{2} ln(\frac{1-error}{error})$.
3. Update weights:

$$D_{t+1}(i) = \frac{D_t(i) exp(-\alpha_t y_i f_t(x_i, \theta))}{Z_t},$$

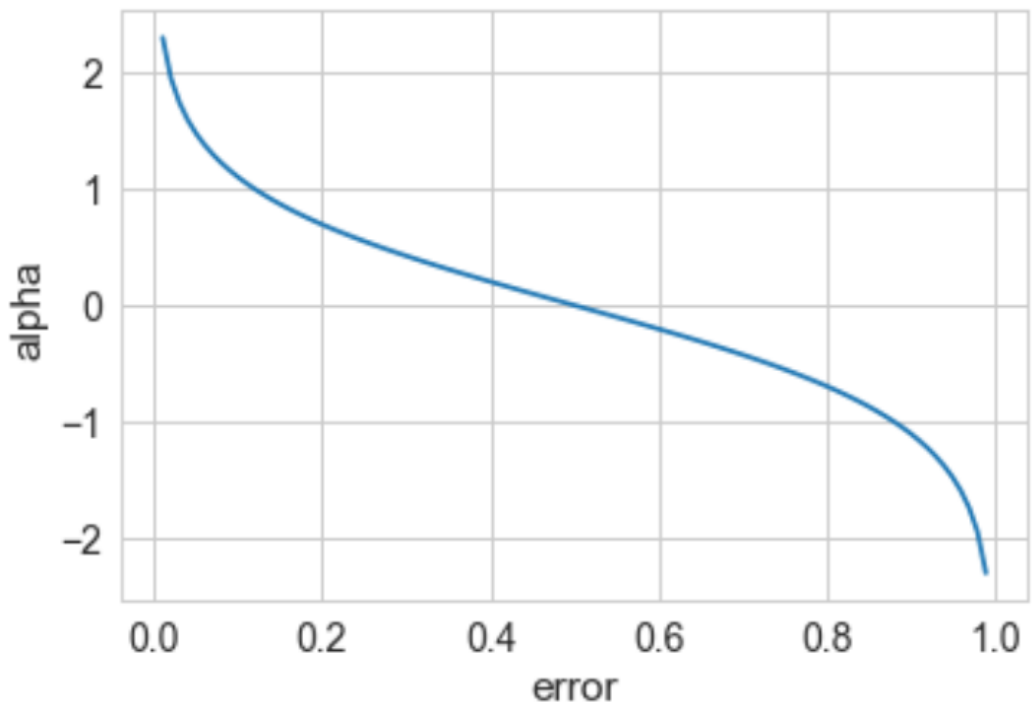
where Z_t is a normalizing factor that makes sure that $\sum D_{t+1}(i) = 1$.

$$\hat{f}(x, \theta) = \sum_{t=1}^{T} \alpha_t f_t(x, \theta)$$

Boosting: Adaboost



Boosting: Adaboost



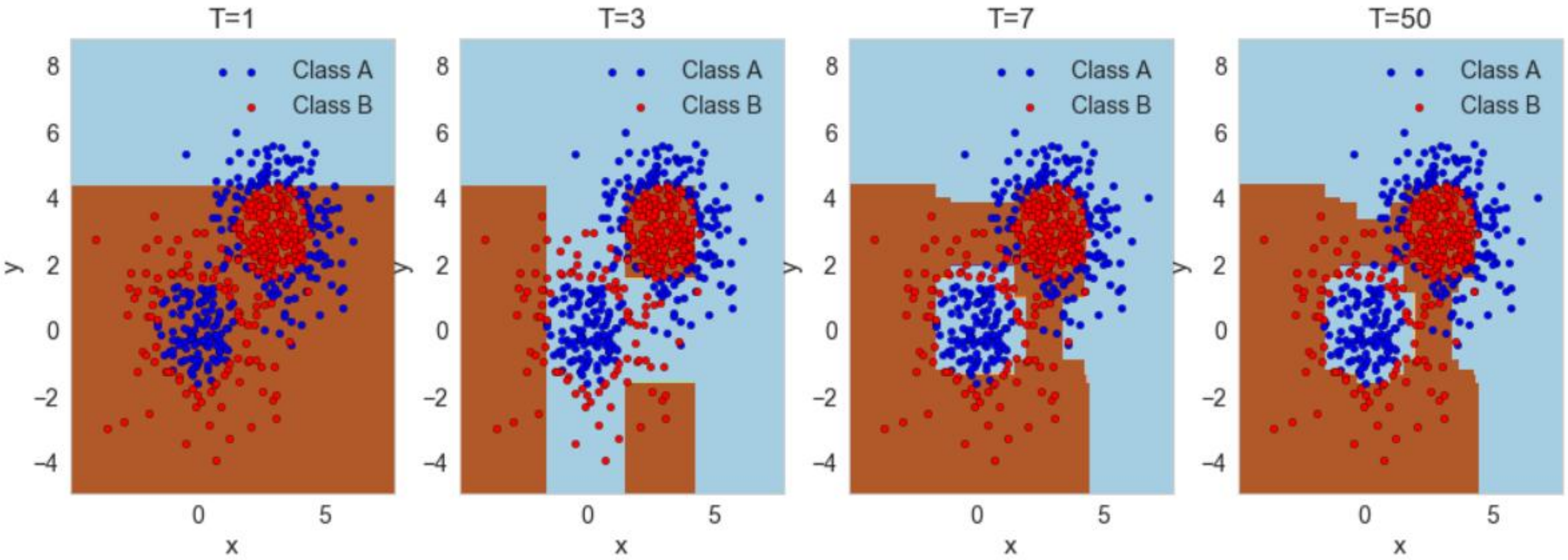
The weight of a weak model in the boosted meta-model increases exponentially as the error approaches 0. Better models are given exponentially more weight.

The weight is zero if the error rate is 0.5. A model with 50% accuracy is no better than random guessing, so it is ignored.

The weight decreases exponentially as the error approaches 1. A negative weight is given to classifiers with worse than 50% accuracy. “Whatever that classifier says, do the opposite!”.

$$\hat{f}(x,\theta) = \sum_{t=1}^{T} \alpha_t f_t(x,\theta)$$

Boosting: Adaboost

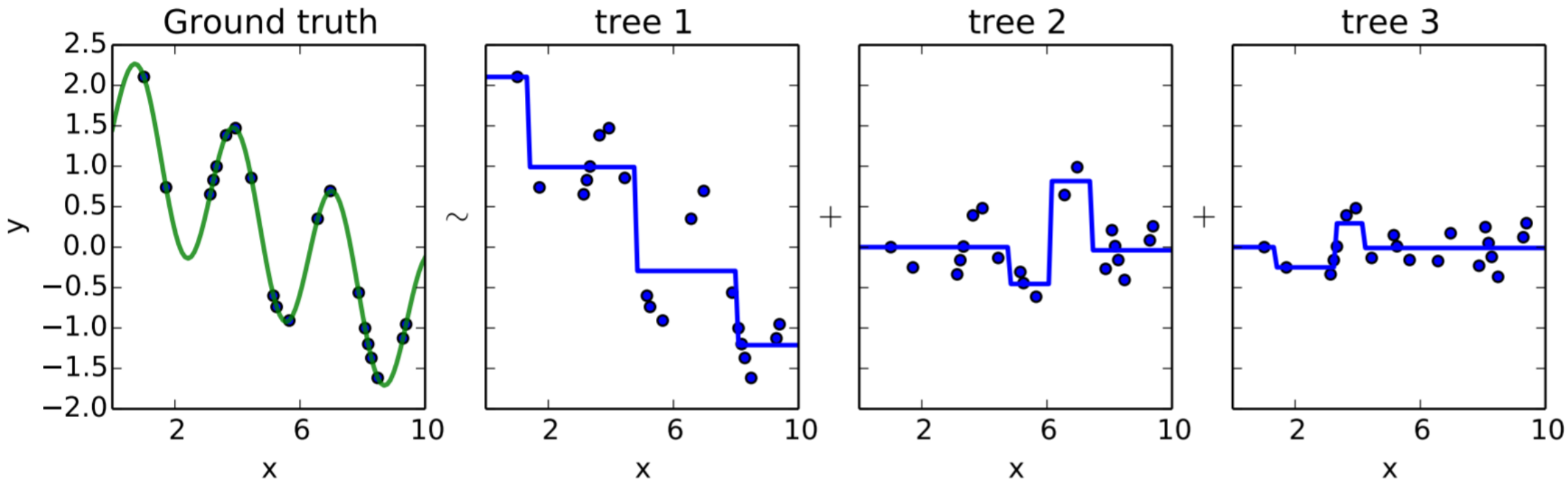


Boosting: Gradient boosting

$$\hat{f}(x, \theta) = \sum_{t=1}^{T} f_t(x, \theta)$$

1. Fit a model $f_1(x, \theta) = y$
2. Fit a model to the **residuals** $h_1(x) = y - f_1(x, \theta)$
3. Create a new model $f_2(x, \theta) = f_1(x, \theta) + h_1(x)$

$$f_0(x, \theta) = \frac{1}{n} \sum_{i=1}^{n} y_i$$

$$f_{t+1}(x, \theta) = f_t(x, \theta) + h_t(x)$$

Boosting: Gradient boosting