Machine Learning for Engineering Problem Solving

Supplementary Slide Decks

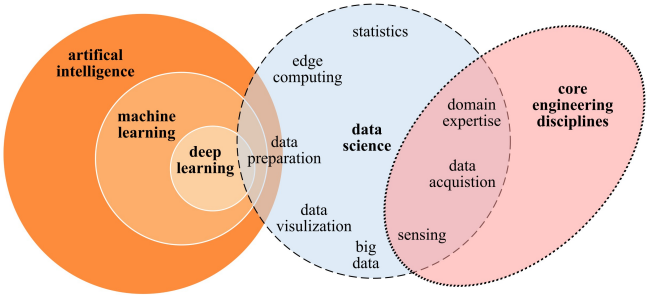
Austin R.J. Downey

Contents

Chapter 1: Basic Concepts in Machine Learning

Austin R.J. Downey

AI, ML, and Data Science



Overlap between the fields of Artificial Intelligence (AI), Data science (DS), and the core engineering disciplines of Civil, Mechanical, Electrical, and Chemical Engineering.

What is Machine Learning?

- Learns patterns from data
- Makes predictions or decisions

- Not robots
- Not rule-based programming

Example: Spam filter

Early AI: Expert Systems

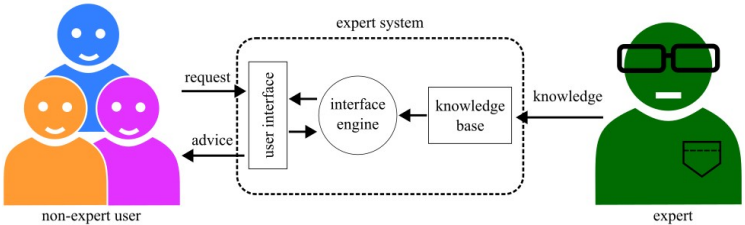


Diagram of an "expert system" in AI, which is a computer program that simulates the decision-making ability of a human expert by using a knowledge base and inference rules.

Examples of Artificial Intelligence

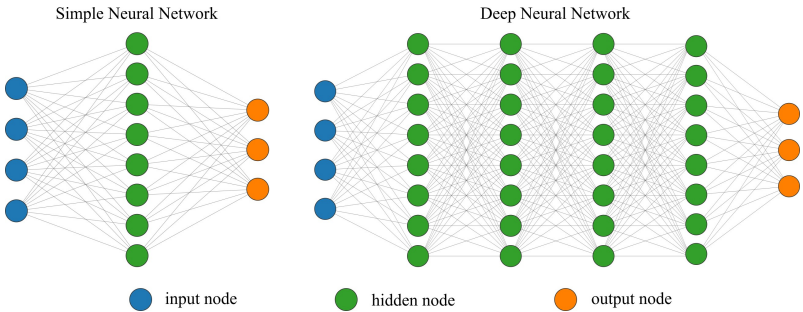
- **Expert systems**: Simulate decision-making using rules and knowledge bases
- **Chatbots**: Simulate conversation with human users

AI in Practice

A Symbolics Lisp Machine, a specialized hardware platform designed to run expert systems which are a version of AI focusing on answering questions to challenging problems[1].

[1] Michael L. Umbricht and Carl R. Friend (Retro-Computing Society of RI), CC BY-SA 3.0 <https://creativecommons.org/licenses/by-sa/3.0>, via Wikimedia Commons

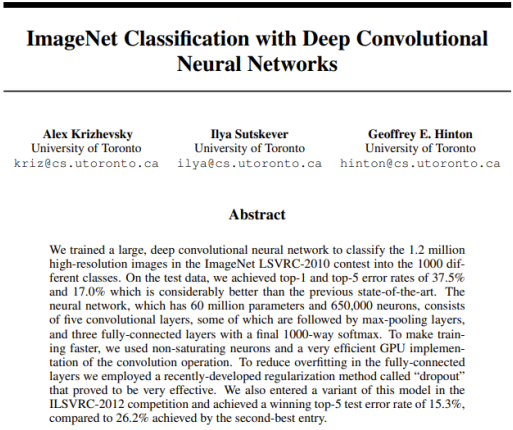
Deep Learning vs Neural Networks



Simple neural network vs deep learning

The ImageNet Breakthrough (2012)

- AlexNet: 60M parameters, 650k neurons
- Enabled by GPUs + dropout
- Top-5 error: 15.3



ImageNet Classification with Deep Convolutional Neural Networks

Alex Krizhevsky
University of Toronto
kriz@cs.utoronto.ca

Ilya Sutskever
University of Toronto
ilya@cs.utoronto.ca

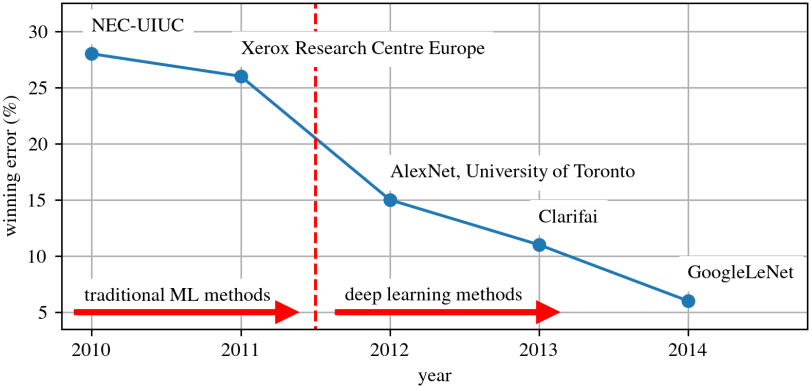
Geoffrey E. Hinton
University of Toronto
hinton@cs.utoronto.ca

Abstract

We trained a large, deep convolutional neural network to classify the 1.2 million high-resolution images in the ImageNet LSVRC-2010 contest into the 1000 different classes. On the test data, we achieved top-1 and top-5 error rates of 37.5% and 17.0% which is considerably better than the previous state-of-the-art. The neural network, which has 60 million parameters and 650,000 neurons, consists of five convolutional layers, some of which are followed by max-pooling layers, and three fully-connected layers with a final 1000-way softmax. To make training faster, we used non-saturating neurons and a very efficient GPU implementation of the convolution operation. To reduce overfitting in the fully-connected layers we employed a recently-developed regularization method called "dropout" that proved to be very effective. We also entered a variant of this model in the ILSVRC-2012 competition and achieved a winning top-5 test error rate of 15.3%, compared to 26.2% achieved by the second-best entry.

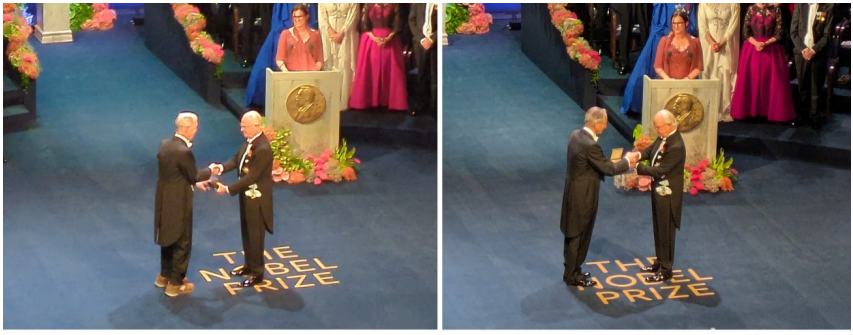
Geoffrey's 2012 paper "ImageNet Classification with Deep Convolutional Neural Networks".

Impact of Deep Learning



ImageNet Competition Results showing the impact of deep learning methods on image classification.

Recognition of Deep Learning



The 2024 Nobel Prize in Physics awarded to John Hopfield and Geoffrey Hinton.

Definition of Machine Learning

- **Arthur Samuel (1959):** "Field of study that gives computers the ability to learn without being explicitly programmed"
- **Tom Mitchell (1997):** "A computer program is said to learn from experience E with respect to some task T and some performance measure P, if its performance on T, as measured by P, improves with experience E."

- Systems improve with experience
- Learn from data, not explicit rules

Learning as a Function

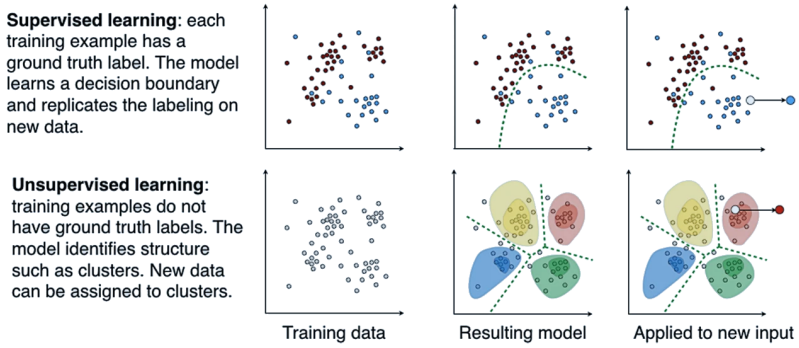
$$Y = f(X)$$

$$Y = f(X) + e$$

- X: input features
- Y: output (target)
- f: unknown function
- e: error (noise)

- Training: build the model
- Inference: use the model

Supervised vs Unsupervised



Supervised vs unsupervised machine learning methods[1].

[1] Modified from: Langs, G., Röhrich, S., Hofmanninger, J. et al., CC BY 4.0 <https://creativecommons.org/licenses/by/4.0>, via Wikimedia Commons

Supervised Learning

Key idea: Data comes with labels

Learn a mapping from inputs $\rightarrow$ outputs

Two main tasks:

- Classification (categories)
- Regression (numeric values)

Goal: generalize to new, unseen data

Supervised Learning in Practice

Training uses labeled data, where each example includes input features and a known target.

Examples:

- Spam filter (spam vs ham)
- House price prediction

Common algorithms:

- k-Nearest Neighbors
- Linear Regression
- Logistic Regression
- Support Vector Machines (SVMs)
- Decision Trees / Random Forests

Unsupervised Learning

Unsupervised learning uses unlabeled data to discover structure and patterns automatically.

Key tasks:

- **Clustering**: group similar data
- **Association**: discover relationships
- **Dimension reduction**: simplify data

Common algorithms:

- k-Means
- Hierarchical Clustering
- Expectation Maximization

Unsupervised Learning in Practice

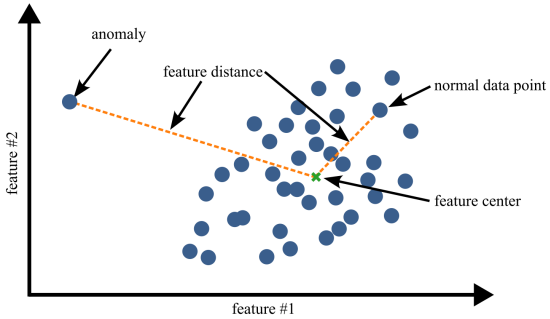
- Finds hidden structure
- No labels provided
- Visualization reveals patterns

- Example: text data
- Words grouped by frequency/meaning

A semantic word cloud of Barack Obama's First Inaugural Address [1]

[1] Modified from: GuoYongzhi, CC BY-SA 4.0 https://creativecommons.org/licenses/by-sa/4.0, via Wikimedia Commons

Anomaly Detection



Anomaly Detection for a given set of data.

Detect unusual patterns that differ from normal behavior.

- Train on normal data
- Identify outliers or rare events
- Applications: fraud detection, defects, data cleaning

Semi-Supervised Learning

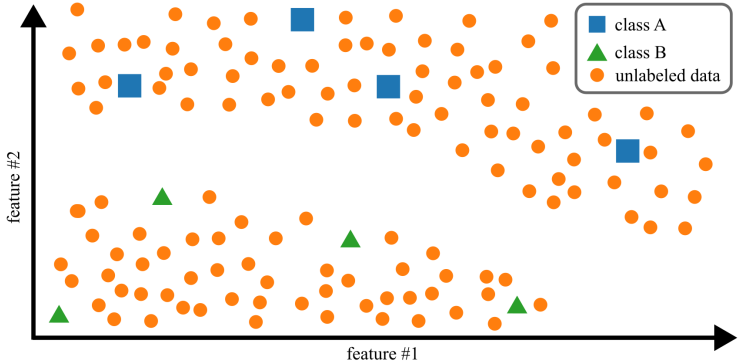
Semi-supervised learning uses a small amount of labeled data with a large amount of unlabeled data.

- Combines supervised and unsupervised learning
- Leverages structure in unlabeled data

Example:

- Group similar images automatically (clustering)
- Label a few examples
- Propagate labels to remaining data

Semi-Supervised Learning Example



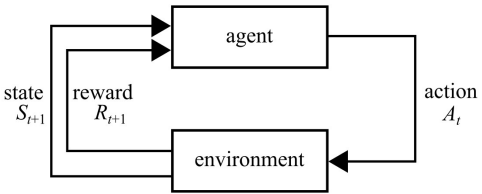
Semisupervised learning enabling the use of a limited set of labeled data to infer the labels of larger unlabeled datasets.

Reinforcement Learning

Reinforcement learning trains an agent to make decisions by interacting with an environment.

- Observe state
- Take action
- Receive reward or penalty

Goal: learn a policy that maximizes total reward

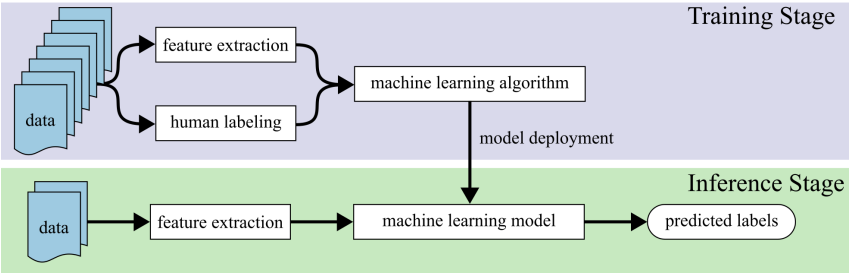


Reinforcement learning diagram.

Types of Learning (Batch and Online)

- How models learn from data
- Two main approaches:
  - Batch learning (offline)
  - Online learning (incremental)
- Trade-off: accuracy vs adaptability vs resources

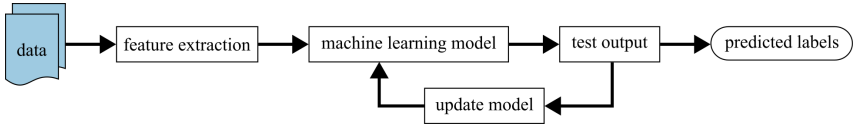
Batch Learning



Batch learning framework with separate training and inference stages.

- Train once on full dataset
- No learning after deployment
- Requires retraining from scratch to update
- Computationally expensive

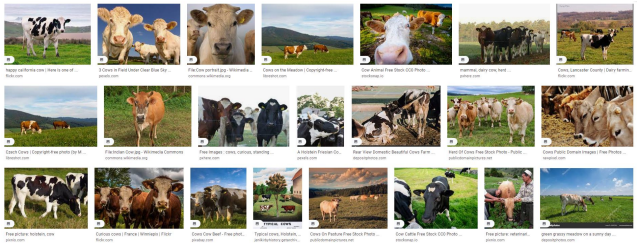
Online Learning



Online learning framework where data is used to continuously training (or update / fine-tune) the model.

- Learns incrementally from new data
- Adapts continuously
- Works for streaming or large datasets
- Learning rate controls adaptation speed

Bad Data in Machine Learning

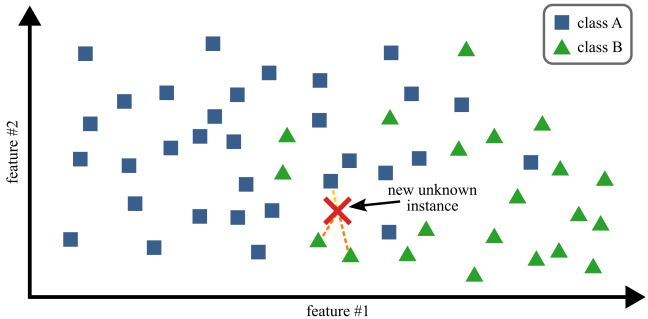


Training a machine learning algorithm to recognize cows may end up just learning to recognize grass. humorously called “short-cut learning”[1].

- Models learn patterns in data — not concepts
- Poor or biased data → wrong conclusions
- Example: learns “grass” instead of “cow”

[1] Screen shoot of a Google image search for cows taken by the Austin Downey, all images stated to be under a creative commons license per Google search tools; also assumed to be fair use under given the educational purpose of this text, via `https://www.google.com/`

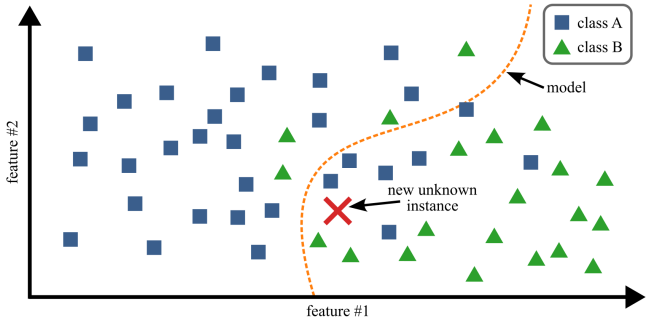
Instance-Based Learning



Instance-based learning where the class of a unknown instance is inferred from the its distance to data points with known labels.

- Memorizes training examples
- Uses similarity (distance) to make predictions
- Flexible but can be slow

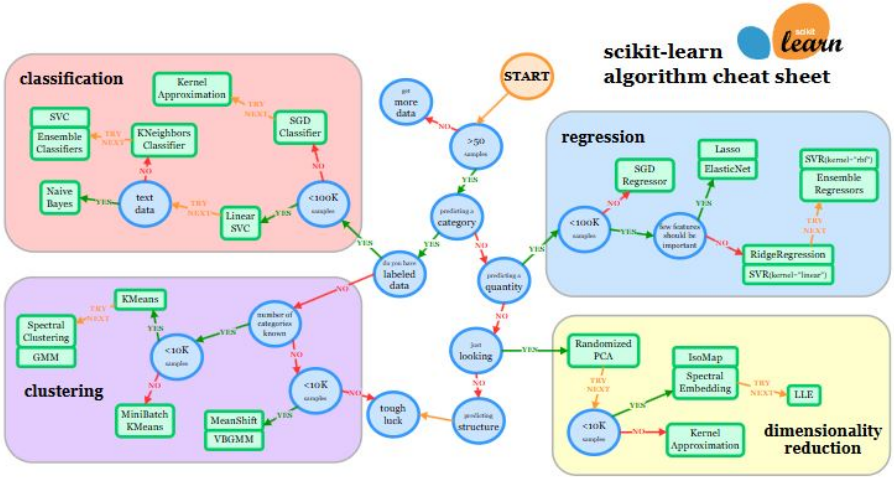
Model-Based Learning



Model-based learning where the class of a unknown instance is inferred from its location in reference to a model trained on the data with known labels.

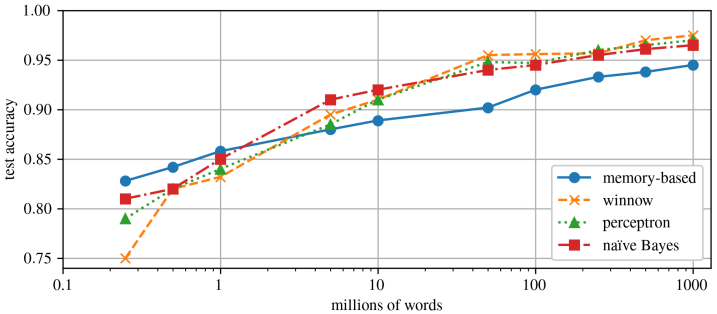
- Learns a model from data
- Uses model to make predictions
- Faster at inference, depends on model quality

Selecting Machine Learning Methods



Flowchart of estimators used in the scikit learn library that intends guide users for what algorithms to use for a given case.

The Unreasonable Effectiveness of Data



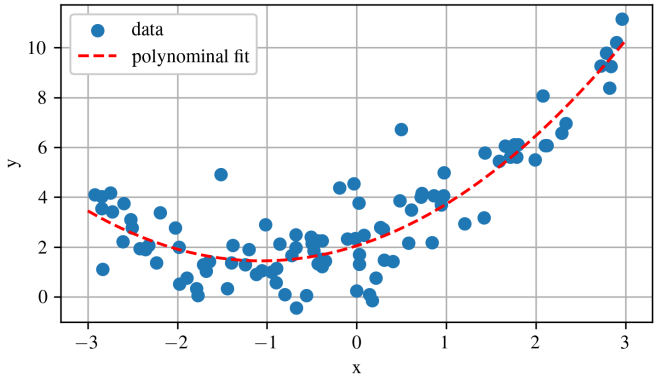
Learning curves showing the importance of the amount of data.

- Michele Banko and Eric Brill (Microsoft, 2001), "Scaling to Very Very Large Corpora for Natural Language Disambiguation"
- More data → better performance
- Simple models can compete with enough data

Chapter 2 Regression

Austin R.J. Downey

Regression



Regression models the relationship between an input variable x (data) and an output variable y (target). The goal is to learn a function that predicts y from x.

Regression Approaches

In this chapter, we examine Linear Regression and two ways to estimate parameters:

- **Closed-form solution:** Solve directly for parameters by minimizing the cost function.
- **Gradient Descent:** Iteratively update parameters to reduce error.

Gradient Descent variants:

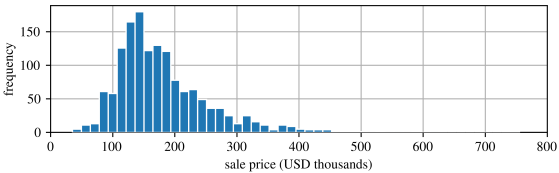
- Batch Gradient Descent
- Stochastic Gradient Descent
- Mini-batch Gradient Descent

Closed-form gives an exact solution immediately, while Gradient Descent converges through successive updates.

Data: Ames Housing Dataset

The Ames housing dataset contains:

- 2,930 home sales
- Data from 2006–2010
- Multiple feature types

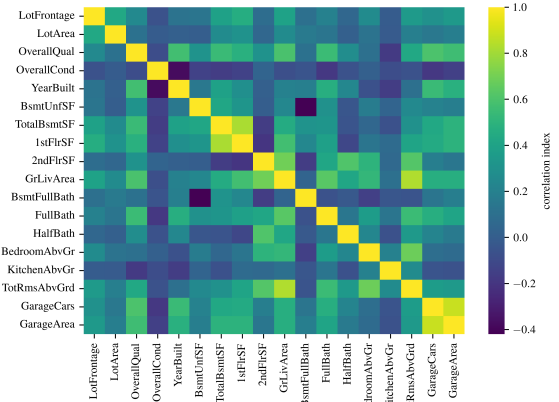


Histogram of sale prices.

Data: Ames Housing Dataset Feature Correlations

Feature relationships:

- Shows correlations between inputs
- Helps identify important variables
- Useful for feature selection

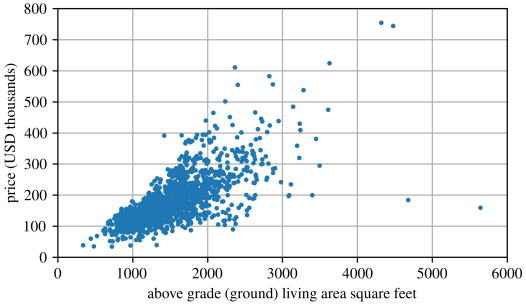


Correlation matrix of features.

Ames Housing Example

We want to predict house price using the Ames dataset.

- Input: above-ground living area
- Target: housing price
- Goal: find a linear relationship between them



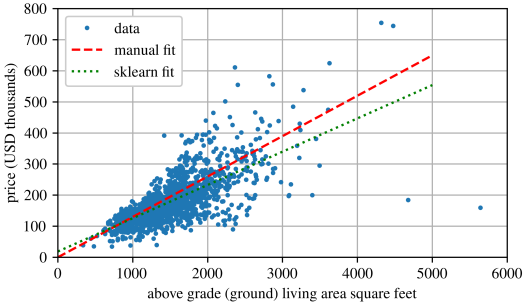
Ames housing data.

A Simple Model

A clear trend is visible: larger homes tend to cost more.

$$\text{price\_model} = \theta_1 + \theta_2 \times \text{above\_ground\_living\_area}$$

- θ_1 is the intercept
- θ_2 is the slope
- The model can represent any line



Ames housing data with trend lines.

Notation

Notations

In terms of linear algebra notation; we use:

- $\mathbf{x}$: lowercase bold symbols denote vectors.
- $\mathbf{X}$: uppercase bold symbols denote matrices.

In machine learning, we commonly use:

- $\mathbf{X}$: the feature matrix (input data).
- $\mathbf{y}$: the target vector (labels or outputs).
- $\mathbf{x}^{(i)}$: the i^{th} instance in the dataset.
- h: the hypothesis or prediction function, $\hat{\mathbf{y}} = h(\mathbf{X})$.
- n: the number of features.
- m: the number of training instances.
- $\hat{x}$: an estimated value ("x-hat").

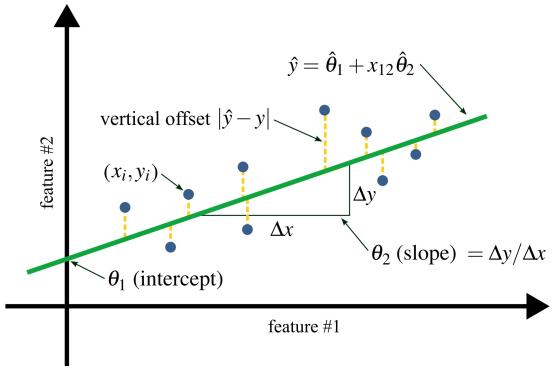
Least Squares Intuition

Goal:

- Find parameters that minimize error
- Fit a line closest to all data points

Idea:

- Measure vertical distance to line
- Square the errors
- Sum over all points



Least squares fit.

How Do We Measure Fit?

Before tuning a model, we need a way to measure how well it fits the data.

- Define a **cost function**
- Penalizes differences between predictions and true values
- Lower cost = better fit

For linear regression:

- Use **Ordinary Least Squares (OLS)**
- Minimizes prediction error over all training data

Mean Squared Error (MSE)

A common cost function is Mean Squared Error:

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

- m: number of data points
- Measures average squared prediction error
- Larger errors are penalized more heavily

For linear regression:

$$J = \frac{1}{m} \sum_{i=1}^{m} (\boldsymbol{\theta}^\top \mathbf{x}^{(i)} - y^{(i)})^2$$

Linear Regression Model

Consider a dataset with n observations and p features.

For each observation:

$$y_i = \theta_1 x_{i1} + \theta_2 x_{i2} + \cdots + \theta_p x_{ip} + \epsilon_i$$

- y_i: target value
- x_{ij}: feature values
- θ_j: model parameters
- ϵ_i: error term

Matrix Formulation

We can write the linear model compactly:

$$\mathbf{y} = \mathbf{X}\boldsymbol{\theta} + \epsilon$$

- $\mathbf{X}$: feature matrix
- $\boldsymbol{\theta}$: parameters
- $\mathbf{y}$: target values

Predictions:

$$\hat{\mathbf{y}} = \mathbf{X}\hat{\boldsymbol{\theta}}$$

Goal:

- Find $\hat{\boldsymbol{\theta}}$ that minimizes error

$$\mathbf{X} = \begin{bmatrix} x_{11} & x_{12} & \cdots & x_{1p} \\ x_{21} & x_{22} & \cdots & x_{2p} \\ \vdots & \vdots & \ddots & \vdots \\ x_{n1} & x_{n2} & \cdots & x_{np} \end{bmatrix}$$

$$\boldsymbol{\theta} = \begin{bmatrix} \theta_1 \\ \theta_2 \\ \vdots \\ \theta_p \end{bmatrix} \qquad \mathbf{y} = \begin{bmatrix} y_1 \\ y_2 \\ \vdots \\ y_n \end{bmatrix}$$

The Normal Equation

Closed-form solution:

$$\hat{\boldsymbol{\theta}} = (\mathbf{X}^\top \mathbf{X})^{-1} \mathbf{X}^\top \mathbf{y}$$

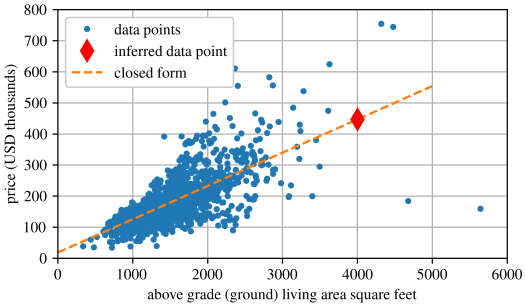
- Gives the exact best-fit parameters
- No iteration required
- Works well for small feature sets

For a simple model:

$$\hat{y} = \hat{\theta}_1 + x\hat{\theta}_2$$

This is just:

$$y = mx + b$$



Ames housing data inferred.

Computational Complexity

- Normal Equation requires inverting $\mathbf{X}^\top \mathbf{X}$ (an $n \times n$ matrix)
- Matrix inversion cost: $O(n^{2.4})$ to $O(n^3)$
- Doubling features $\Rightarrow \sim 5$–$8\times$ more computation

Warning

For very large feature sets (e.g., $n \sim 100{,}000$), the Normal Equation becomes extremely slow.

- Scales linearly with number of instances: $O(m)$
- Prediction is fast: linear in both m and n

Example 2.1: Ames Housing (Linear Regression)

- Fit a linear model to predict house price from living area
- Compare approaches:
  - Manual fit (visual intuition)
  - Closed-form (Normal Equation)
  - Gradient Descent
- In practice (scikit-learn):
  - `LinearRegression` (closed-form)
  - `SGDRegressor` (iterative)

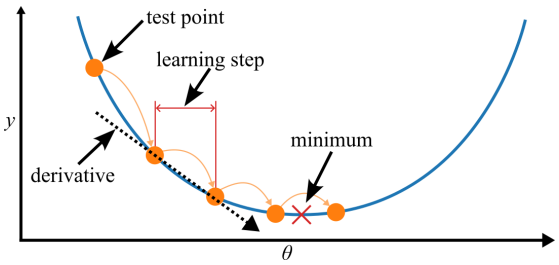
Next: methods that scale better for large feature sets or large datasets.

Gradient Descent

- Optimization algorithm to minimize a cost function
- Iteratively updates parameters

Idea:

1. Compute the gradient (slope)
2. Move in direction of steepest descent
3. Repeat until minimum is reached

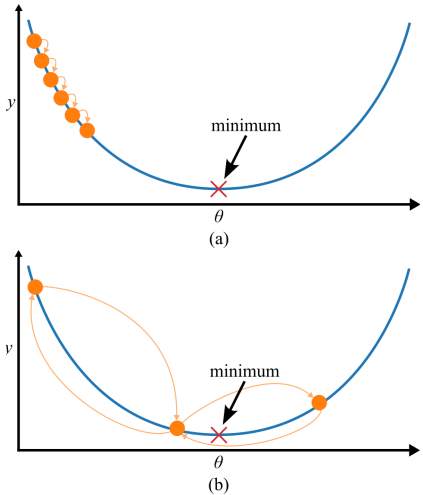


Gradient descent moving toward a minimum.

Learning Rate

- Step size controlled by learning rate η
- Small $\eta \rightarrow$ slow convergence
- Large $\eta \rightarrow$ overshooting / divergence

Choosing η is critical



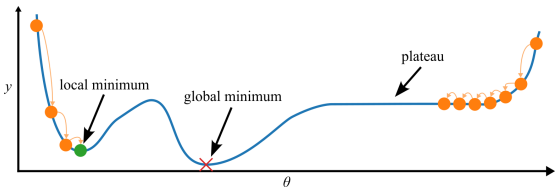
Effect of different learning rates.

Challenges

- Cost functions may be complex:
  - Local minima
  - Plateaus
  - Ridges
- Starting point matters
- Convergence may be slow

For linear regression:

- MSE is convex
- One global minimum

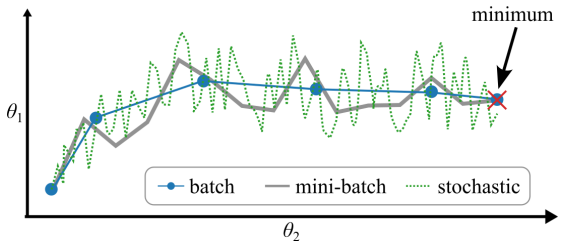


Optimization challenges in complex landscapes.

Comparing Gradient Descent Methods

- Different methods reach the minimum differently
- Trade-off between speed and stability
- Some methods are noisy but fast
- Others are stable but computationally expensive

All aim to minimize the same cost function



Paths taken by different methods.

Method Comparison

Different approaches to linear regression vary in speed, scalability, and flexibility.

algorithm	large m	in memory	large n	hyperparams	scaling	sklearn
normal equation	fast	yes	slow	0	no	`LinearRegression`
batch GD	slow	yes	fast	2	yes	n/a
stochastic GD	fast	no	fast	≥ 2	yes	`SGDRegressor`
mini-batch GD	fast	no	fast	≥ 2	yes	n/a

NOTE

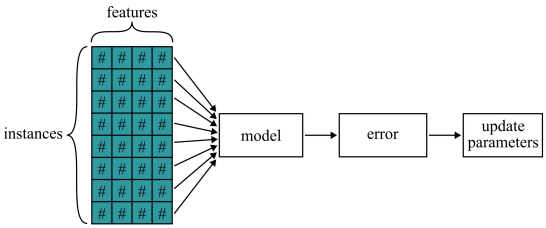
After training, there is minimal difference between the models: these algorithms converge on similar solutions and make predictions in a nearly identical manner.

Batch Gradient Descent

- Uses the entire dataset at each step
- Minimizes the cost function over all data
- Based on Mean Squared Error (MSE)

$$J = \frac{1}{m} \sum_{i=1}^{m} (\boldsymbol{\theta}^\top \mathbf{x}^{(i)} - y^{(i)})^2$$

- Computes exact gradient each iteration
- Can be slow for large datasets



Batch Gradient Descent flow.

Gradient Computation

Compute partial derivatives:

$$\frac{\partial}{\partial \theta_j} J(\theta) = \frac{2}{m} \sum_{i=1}^{m} (\boldsymbol{\theta}^\top \mathbf{x}^{(i)} - y^{(i)}) x_j^{(i)}$$

Vector form:

$$\nabla_\theta J(\boldsymbol{\theta}) = \frac{2}{m} \mathbf{X}^\top (\mathbf{X}\boldsymbol{\theta} - \mathbf{y})$$

- Computes all slopes at once
- Direction of steepest ascent

Key Idea

Move in the opposite direction of the gradient to reduce error.

Warning

Uses the full dataset at every step → can be slow for large m.

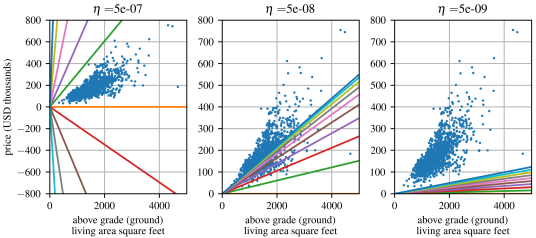
Parameter Update and Learning Rate

Update rule:

$$\boldsymbol{\theta}^{(\text{next})} = \boldsymbol{\theta} - \eta \nabla_{\theta} J(\boldsymbol{\theta})$$

- η: learning rate (step size)
- Controls speed of convergence

- Too large → divergence
- Too small → slow convergence
- Just right → fast convergence

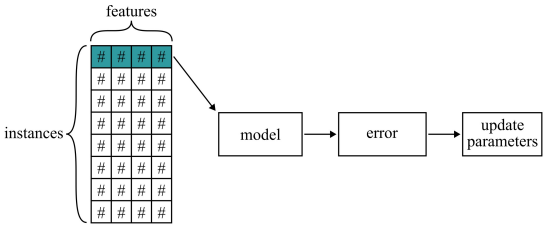


Effect of different learning rates.

Stochastic Gradient Descent

- Updates parameters using one sample at a time
- Uses $(x^{(i)}, y^{(i)})$ instead of full dataset

- Faster for large datasets
- Enables online learning



Flow of Stochastic Gradient Descent.

SGD Update Rule

Update rule:

$$\boldsymbol{\theta}^{(\text{next})} = \boldsymbol{\theta} - \eta \nabla_\theta J(\boldsymbol{\theta}; x^{(i)}, y^{(i)})$$

- Updates after every single example
- Avoids redundant computations
- More efficient than Batch GD for large m

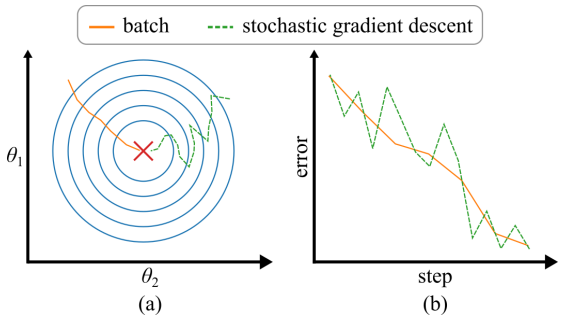
Key Idea

Instead of waiting to see all data, update immediately using each new sample.

SGD Behavior

- Fast updates, but noisy path
- High variance in cost function
- Does not decrease smoothly

- Often oscillates around minimum
- Can still reach a good solution



Batch vs. stochastic gradient descent behavior.

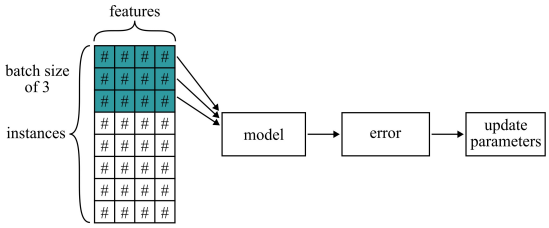
Mini-batch Gradient Descent

- Uses small random subsets (mini-batches)
- Between Batch GD and Stochastic GD

Update rule:

$$\boldsymbol{\theta}^{(\text{next})} = \boldsymbol{\theta} - \eta \nabla_\theta J(\boldsymbol{\theta}; x^{(i:i+n)}, y^{(i:i+n)})$$

- Faster than Batch GD
- More stable than SGD
- Efficient on GPUs (matrix operations)

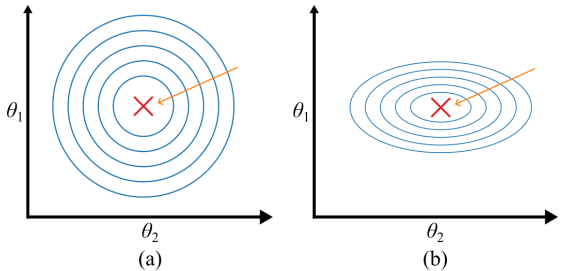


Flow of Mini-batch Gradient Descent.

Feature Scaling

- Different feature scales distort the cost surface
- Creates an elongated “valley”
- Equal scaling → fast convergence
- Unequal scaling → slow convergence

Gradient Descent becomes inefficient without scaling



Effect of feature scaling on convergence.

Scaling Methods

Min-max scaling

$$\mathbf{X}' = \frac{\mathbf{X} - \mathbf{X}_{\min}}{\mathbf{X}_{\max} - \mathbf{X}_{\min}}$$

- Rescales to $[0, 1]$
- Sensitive to outliers

Standardization

$$\mathbf{X}' = \frac{\mathbf{X} - \overline{\mathbf{X}}}{\sigma}$$

- Zero mean, unit variance
- Less sensitive to outliers

- Both improve Gradient Descent performance
- Standardization is most commonly used

Tools

`MinMaxScaler`
`StandardScaler`

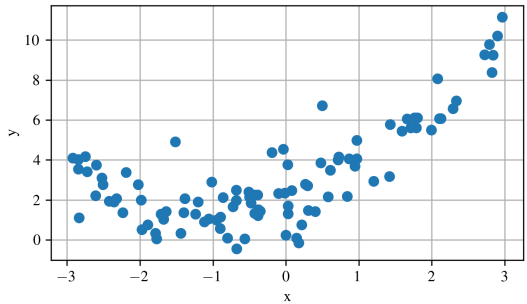
Polynomial Regression

What if the data is not linear?

A quadratic model can represent curved behavior:

$$\hat{y} = ax^2 + bx + c$$

- Linear models underfit nonlinear data
- Polynomial terms allow curved fits
- Still linear in the parameters



Dataset generated from a quadratic relationship.

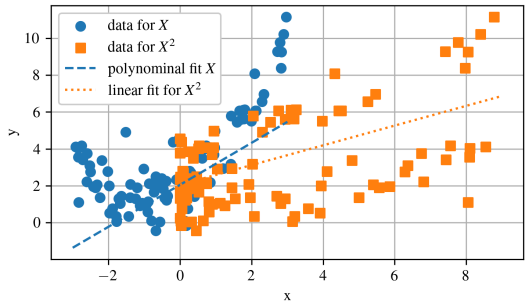
Feature Expansion

Polynomial regression works by expanding the feature space.

For example:

$$x \to [x,\ x^2]$$

- Add higher-order terms as new features
- Train a linear model on the expanded data
- Allows nonlinear behavior



Linear fits to the expanded feature set.

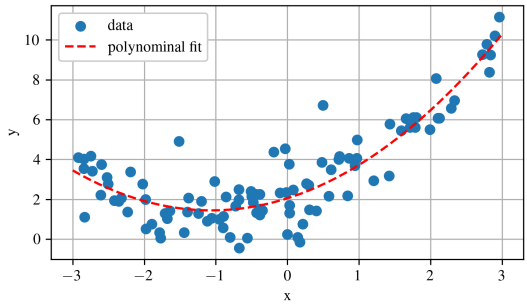
Polynomial Model Fit

The learned coefficients combine into:

$$\hat{y} = ax^2 + bx + c$$

- a: quadratic term
- b: linear term
- c: bias/intercept

The resulting model can approximate curved data.



Polynomial regression fit.

Polynomial Features

Polynomial regression can also model feature interactions.

Example with two features a and b:

$$[a,\ b]$$

Degree-3 expansion:

$$[a,\ b,\ a^2,\ b^2,\ a^3,\ b^3,\ ab,\ a^2b,\ ab^2]$$

- Captures nonlinear relationships
- Captures interactions between variables
- Implemented with:

```
PolynomialFeatures(degree=d)
```

Combinatorial Explosion

Warning

Polynomial feature expansion can rapidly increase the number of features.

Number of expanded features:

$$\frac{(n+d)!}{d!n!}$$

- n: original number of features
- d: polynomial degree

Higher polynomial degrees can:

- Increase memory usage
- Slow training
- Cause overfitting

Example 2.2: Polynomial Regression

- Fit a nonlinear dataset using polynomial regression
- Add x^2 as an additional feature
- Train a linear model on the expanded feature set

Demonstrates how:

- Linear models can approximate curved relationships
- Polynomial terms improve model flexibility
- The resulting fit compares to the original data

Tools used:

`PolynomialFeatures` `LinearRegression`

Chapter 3 Machine Learning Workflows

Austin R.J. Downey

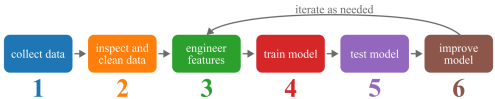
Machine Learning Workflows

A machine learning workflow transforms raw data into a trained and evaluated model.

Important steps:

- Collect data
- Inspect and clean data
- Engineer features
- Train and test model
- Improve model

The workflow is almost always iterative.



Typical machine learning workflow.

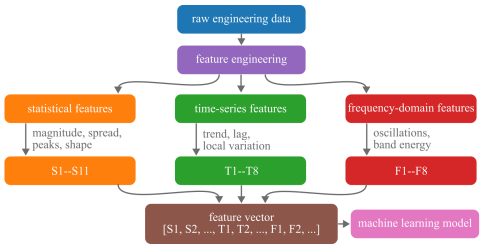
Feature Engineering

Feature engineering transforms raw data into useful model inputs.

Good features include quantities, categorical, or countable.

They can also be extracted from signals:

- magnitude
- variability
- time-dependent behavior
- frequency content
- physical relationships



Transforming raw data into a feature vector.

Statistical Features

Statistical features summarize the distribution of values within a dataset, observation, or time window.

No.	Feature	Interpretation
S1	Maximum value	Largest response in the signal.
S2	Minimum value	Smallest response in the signal.
S3	Mean value	Average level of the signal.
S4	Absolute mean value	Average magnitude of the signal.
S5	Root mean square value	Effective signal magnitude.
S6	Standard deviation	Variation in the signal.
S7	Skewness	Bias toward one side.
S8	Kurtosis	Sharp peaks or outliers.
S9	Shape factor	Effective magnitude relative to average magnitude.
S10	Crest factor	Largest peak relative to effective level.
S11	Impulse factor	Largest peak relative to average magnitude.

* The full equations are provided in Table 1 of Chapter 3 in the accompanying text.

Time-Series Features

Time-series features describe how a signal changes with time.

No.	Feature	Interpretation
T1	Lagged value	Recent history of the signal.
T2	First difference	Step-to-step change in the signal.
T3	Rate of change	How quickly the signal changes with time.
T4	Moving average	Local trend of the signal.
T5	Rolling standard deviation	Local variability of the signal.
T6	Rolling maximum	Recent peak behavior.
T7	Rolling minimum	Recent low-response behavior.
T8	Zero-crossing rate	How often the signal changes sign.

* The full equations are provided in Table 2 of Chapter 3 in the accompanying text.

Frequency-Domain Features

Frequency-domain features describe how signal content is distributed across frequency.

No.	Feature	Interpretation
F1	Dominant frequency	Strongest periodic behavior.
F2	Spectral centroid	Center of the spectrum.
F3	Spectral bandwidth	Spread of energy across frequency.
F4	Low-frequency energy ratio	Share of energy in slow behavior.
F5	Middle-frequency energy ratio	Share of energy in mid-range behavior.
F6	High-frequency energy ratio	Share of energy in rapid behavior.
F7	High-to-low frequency energy ratio	Rapid behavior relative to slow behavior.
F8	Peak-to-total spectral energy ratio	Dominance of the largest frequency peak.

* The full equations are provided in Table 3 of Chapter 3 in the accompanying text.

Overfitting

Training and testing on the same dataset is a major mistake.

- The model can memorize the training data
- Training accuracy may appear perfect
- Performance on new data may be poor

This problem is called:

Overfitting

Goal:

- Build models that generalize to unseen data

Training and Testing Sets

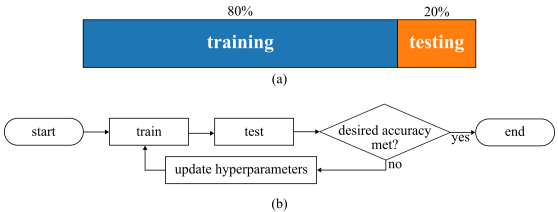
Standard supervised learning workflow:

- Train model using training data
- Evaluate model using test data
- Test data must remain unseen during training

Notation:

$$\mathbf{X}_{\text{train}}, \quad \mathbf{y}_{\text{train}}$$

$$\mathbf{X}_{\text{test}}, \quad \mathbf{y}_{\text{test}}$$



Training/testing split.

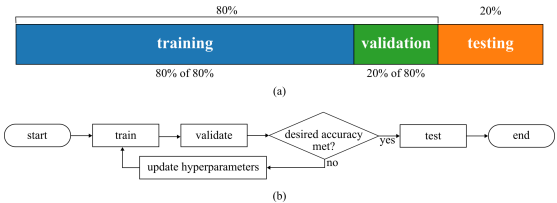
Validation Sets

A validation set is often added to tune models.

- Training set:
  - Learn model parameters
- Validation set:
  - Tune hyperparameters
- Test set:
  - Final evaluation

Scikit-Learn:

`train_test_split`



Training, validation, and testing workflow.

Machine Learning Pipelines

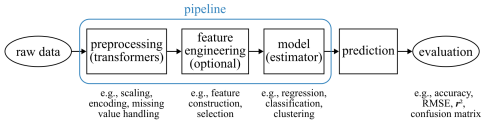
Scikit-Learn pipelines combine multiple processing steps into a single workflow.

Typical steps:

- Data preprocessing
- Feature scaling
- Feature engineering
- Model training

Benefits:

- Cleaner code
- Consistent preprocessing
- Easier cross-validation
- Simplified hyperparameter tuning



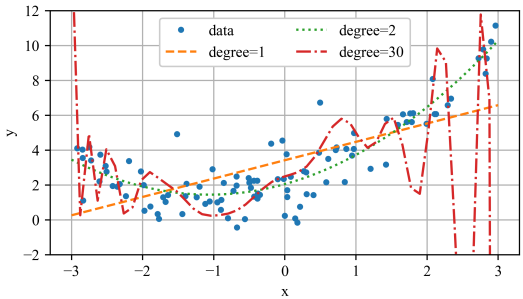
Example machine learning pipeline.

Learning Curves

Model complexity strongly affects performance.

- Low complexity:
  - Underfitting
- High complexity:
  - Overfitting
- Intermediate complexity:
  - Better generalization

Learning curves help diagnose model behavior.



Underfitting, good fit, and overfitting.

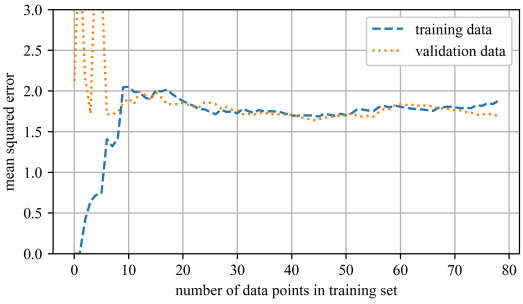
Underfitting

Characteristics of underfitting:

- Model is too simple
- Cannot capture data complexity
- Training and validation errors remain high

Learning curves:

- Curves are close together
- Errors plateau at high values



Learning curves for an underfitting model.

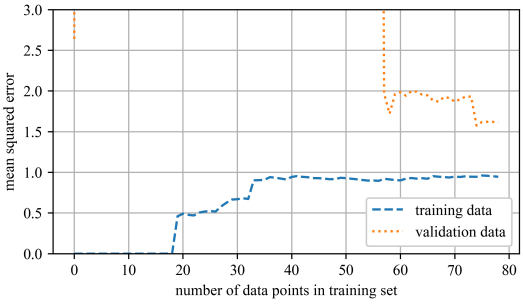
Overfitting

Characteristics of overfitting:

- Model is too complex
- Fits noise in the training data
- Excellent training performance
- Poor validation performance

Learning curves:

- Large gap between curves
- High variance



Learning curves for an overfitting model.

Well-Fit Model

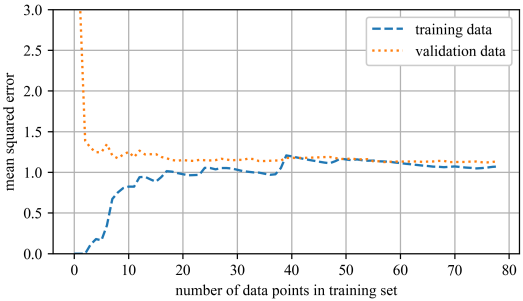
A good model balances:

- Bias
- Variance

Characteristics:

- Low training error
- Low validation error
- Small gap between curves

The model generalizes well to unseen data.



Learning curves for a well-fit model.

Interpreting Learning Curves

Learning curves compare:

Training Error vs. Validation Error

- Underfitting:
  - Both errors high
  - Curves close together
- Overfitting:
  - Training error low
  - Validation error much higher
- Good fit:
  - Both errors low
  - Curves converge

Example 3.1: Learning Curves

- Compare learning curves for:
  - Linear regression
  - Polynomial regression
- Plot training and validation errors as the training set grows
- Demonstrates:
  - Underfitting
  - Overfitting
  - Model generalization
  - Bias–variance tradeoff
- Shows how model complexity affects performance

Regularized Linear Models

Overfitting can be reduced through:

Regularization

Idea:

- Constrain model complexity
- Prevent fitting noise in the data
- Improve generalization

For polynomial models:

- Use a lower polynomial degree

For linear models:

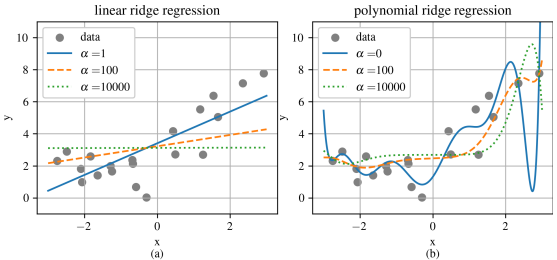
- Penalize large coefficient values

Ridge Regression

Ridge Regression adds a penalty term to the cost function:

$$J(\theta) = \text{MSE}(\theta) + \alpha \frac{1}{2} \sum_{i=1}^{n} \theta_i^2$$

- α: regularization strength
- Large $\alpha \rightarrow$ smaller weights
- Reduces overfitting



Ridge regression with different α values.

Note

The bias term θ_0 is not regularized.

Example 3.2: Ridge Regression

- Apply Ridge Regression to a high-degree polynomial model
- Add regularization to reduce overfitting
- Use a pipeline with:
  - `PolynomialFeatures`
  - `StandardScaler`
  - `Ridge`
- Demonstrates:
  - Smoother predictions
  - Reduced variance
  - Improved model stability
- Compare different values of α

Early Stopping

Early stopping is a form of regularization for iterative learning methods.

Idea:

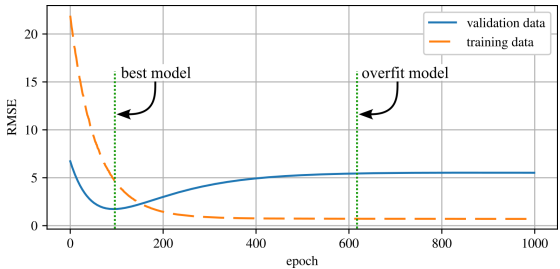
- Monitor validation error during training
- Stop training when validation error begins increasing

Benefits:

- Prevents overfitting
- Improves generalization
- Simple and computationally inexpensive

Key Idea

Stop training at the minimum validation error.



Early stopping prevents overfitting.

Example 3.3: Early Stopping

- Train a high-degree polynomial model using:
  - Stochastic Gradient Descent
- Track:
  - Training error
  - Validation error
- Demonstrates:
  - How overfitting develops during training
  - How validation error identifies the optimal stopping point
  - Improved generalization through early stopping
- Training halts once validation performance begins degrading

Chapter 4 Classification

Austin R.J. Downey

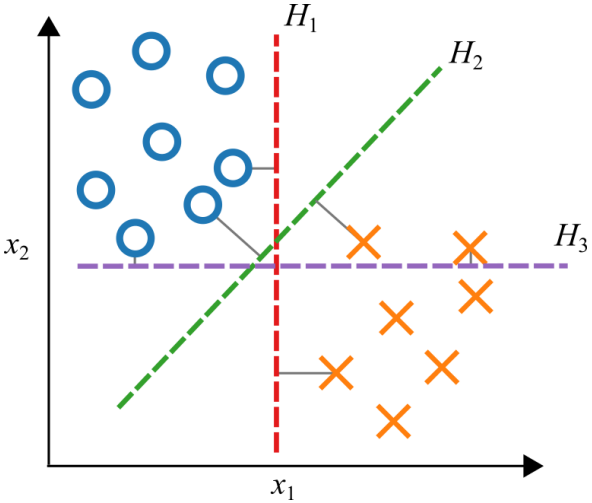
Classification

Machine learning tasks often fall into two categories:

- Regression
  - Predict continuous values
- Classification
  - Predict discrete classes

In this chapter we focus on:

- Binary classification
- Multiclass classification
- Performance metrics



Linear decision boundaries separating two classes.

Binary Classifier

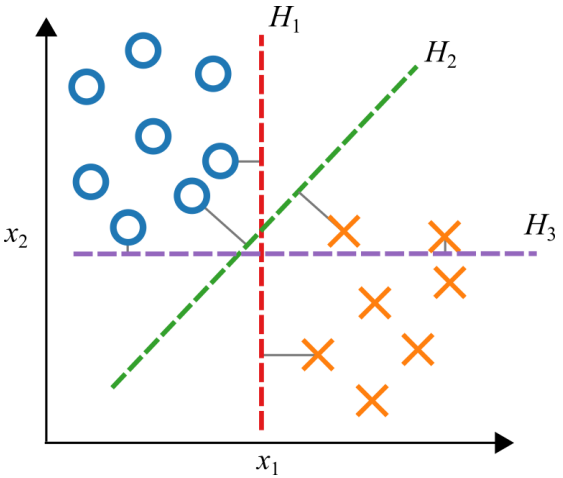
A binary classifier assigns inputs to one of two classes.

Decision rule:

$$\hat{y} = \begin{cases} 1 & \text{if } \boldsymbol{\theta}^\top \mathbf{x} + b \geq 0 \\ 0 & \text{otherwise} \end{cases}$$

- $\mathbf{x}$: input features
- $\boldsymbol{\theta}$: model weights
- b: bias term

Training adjusts $\boldsymbol{\theta}$ and b to minimize classification error.



Multiple linear decision boundaries can correctly separate the data.

Data: MNIST Dataset

The MNIST dataset contains:

- 70,000 handwritten digits
- Classes 0 through 9
- Standard benchmark for classification

Typical split:

- 60,000 training images
- 10,000 testing images

Examples of MNIST digits.

Data: What is a Feature?

Each MNIST image is:

$$28 \times 28 = 784$$

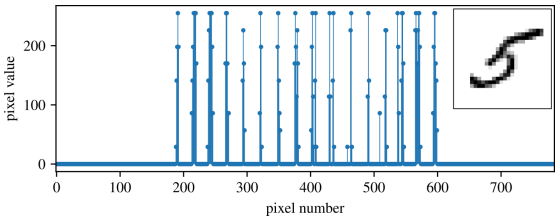
pixels.

In machine learning:

- Each pixel is a feature
- Pixel intensity ranges from 0 to 255
- One image becomes a vector with 784 features

For MNIST:

$$\mathbf{x} = [x_1, x_2, \ldots, x_{784}]$$



Each pixel becomes a feature used by the classifier.

Example 4.1: MNIST Dataset

To simplify the classification problem, we begin by building a: **5-detector**

Goal:

- Determine whether a digit is a 5
- Binary output:
  - 1 = digit is a 5
  - 0 = digit is not a 5

Why choose 5s?

- One of the most difficult digits to classify
- Frequently confused with 3s and 8s
- Good benchmark for binary classification

This example demonstrates how to use:

`sklearn.datasets.fetch_openml`

to load and visualize the MNIST dataset.

Regularized Linear Classifier

A practical way to build the MNIST "5-detector" is with a regularized linear classifier trained using Stochastic Gradient Descent.

- Efficient on large datasets
- Simple to implement
- Updates one example at a time

Main drawbacks:

- Needs careful hyperparameter tuning
- Sensitive to feature scaling

Training Objective

Given labeled samples $(\mathbf{x}^{(i)}, y^{(i)})$ with $y^{(i)} \in \{0, 1\}$, the "5-detector" learns parameters by minimizing:

$$J(\boldsymbol{\theta}) = \frac{1}{m} \sum_{i=1}^{m} \max\bigl(0, 1 - y^{(i)} \boldsymbol{\theta}^{\mathrm{T}} \mathbf{x}^{(i)}\bigr) + \frac{\lambda}{2} \left\| \boldsymbol{\theta}_{1:} \right\|_2^2$$

- First term: hinge loss
- Second term: ℓ_2 regularization
- Bias term is not regularized

Note

In scikit-learn, this model is implemented as `SGDClassifier`.

Example 4.2: Regularized Linear Classifier

- Train a regularized linear classifier with `SGDClassifier`
- Use the MNIST dataset to distinguish the digit "5" from all others
- Show how regularization improves generalization
- Demonstrate prediction on a selected handwritten digit

This example illustrates how a sparse linear decision boundary can work well for binary digit classification.

Performance Measures for Classification

Evaluating a classifier is often more challenging than evaluating a regressor.

Questions we might ask:

- How many predictions were correct?
- What kinds of mistakes were made?
- Which classes are most often confused?

A common tool for answering these questions is the:

Confusion Matrix

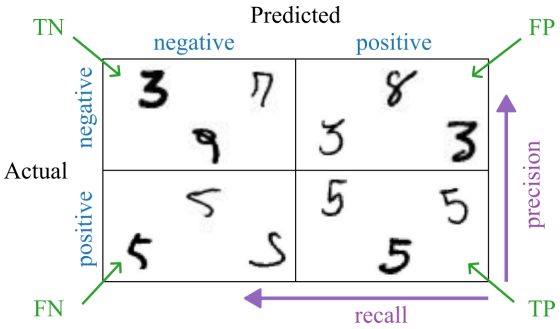
Binary Confusion Matrix

Two common error types:

- Type I Error:
  - False Positive (FP)
- Type II Error:
  - False Negative (FN)

A confusion matrix summarizes:

- True Positives (TP)
- True Negatives (TN)
- False Positives (FP)
- False Negatives (FN)



Binary confusion matrix.

Confusion Matrix

Rows correspond to:

- Actual classes

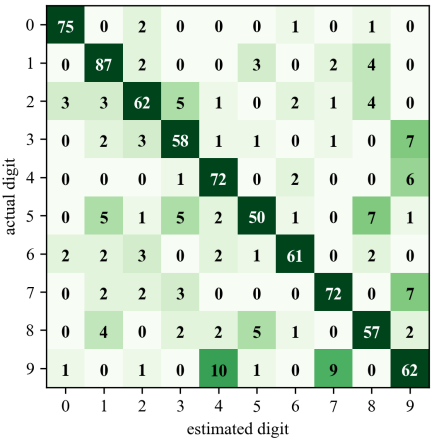
Columns correspond to:

- Predicted classes

A good classifier has:

- Large values on the diagonal
- Small values elsewhere

Off-diagonal entries indicate misclassifications.



MNIST confusion matrix.

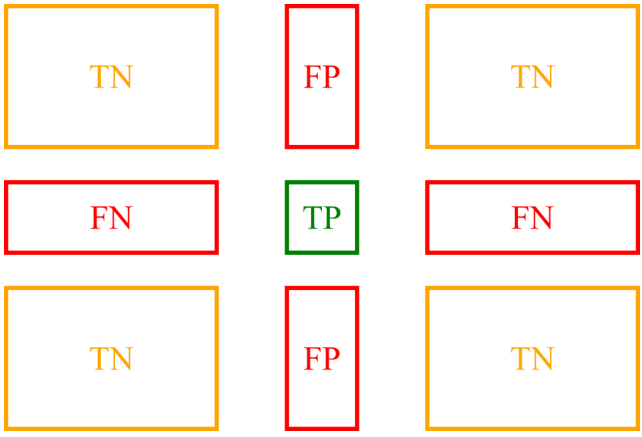
Confusion Matrix Terminology

A confusion matrix summarizes classifier predictions.

- Rows = actual class
- Columns = predicted class

- TP: True Positive
- TN: True Negative
- FP: False Positive
- FN: False Negative

A perfect classifier contains values only along the main diagonal.



Example 4.3: Binary Confusion Matrix

- Train a binary classifier to detect the digit “5”
- Generate a confusion matrix
- Compute:
  - True Positives (TP)
  - True Negatives (TN)
  - False Positives (FP)
  - False Negatives (FN)
- Examine examples of:
  - Missed 5s (false negatives)
  - Incorrectly detected 5s (false positives)
- Use the confusion matrix to understand where the classifier fails

Accuracy

Accuracy is defined as

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

or

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

- Simple and intuitive
- Often the first metric reported
- Does not distinguish between error types

Why Accuracy Isn't Enough

A classifier may achieve high accuracy while still making costly mistakes.

Examples:

- Credit card fraud detection
- Medical diagnosis
- Network intrusion detection

A false negative may be much more costly than a false positive.

We therefore introduce:

Precision **Recall**

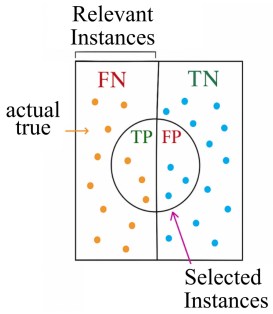
Selected and Relevant Instances

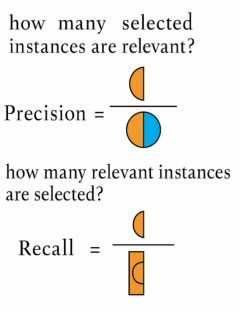
Definitions:

$$\text{Selected Instances} = TP + FP$$

$$\text{Relevant Instances} = TP + FN$$

These quantities form the basis of precision and recall.





Precision

Precision answers:

Of all positive predictions, how many were correct?

$$\text{Precision} = \frac{TP}{TP + FP}$$

High precision implies:

- Few false positives
- Positive predictions are trustworthy

Recall

Recall answers:

Of all actual positives, how many were found?

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

High recall implies:

- Few false negatives
- Important cases are not missed

F1 Score

The F1 score balances precision and recall.

$$F_1 = 2\frac{\text{Precision} \times \text{Recall}}{\text{Precision} + \text{Recall}}$$

Equivalent form:

$$F_1 = \frac{TP}{TP + \frac{FP+FN}{2}}$$

Useful when both precision and recall are important.

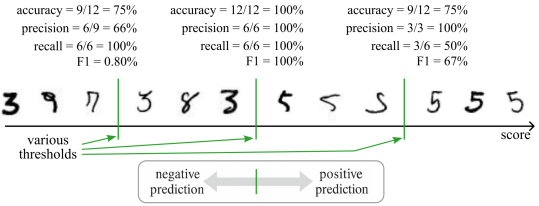
Precision–Recall Tradeoff

Changing the classification threshold changes:

- Precision
- Recall
- F1 score

Higher threshold:

- Higher precision
- Lower recall



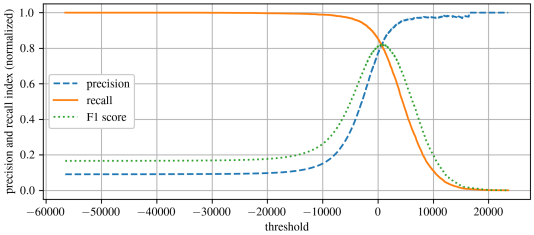
Choosing a Threshold

Threshold selection depends on the application.

Examples:

- Medical diagnosis → high recall
- Spam detection → high precision

The F1 score often helps identify a good compromise.



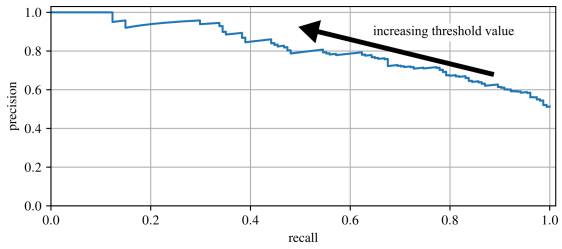
Precision–Recall Curve

The precision–recall curve summarizes classifier performance across all thresholds.

Better classifiers typically have:

- Higher precision
- Higher recall
- Larger AUC-PR

Applications determine the desired operating point.



Example 4.4: Accuracy, Precision, and Recall

- Train a binary classifier to detect the digit “5”
- Compute:
  - Accuracy
  - Precision
  - Recall
  - F1 Score
- Compare manual and sklearn calculations
- Visualize threshold effects
- Explore precision–recall tradeoffs

k-fold Cross-Validation

Training an SGD classifier does not always produce identical results.

Reasons:

- Random initialization
- Random ordering of samples
- Stochastic optimization

Possible solutions:

- More training epochs
- More data
- Cross-validation

Goal:

- Obtain a more reliable estimate of model performance

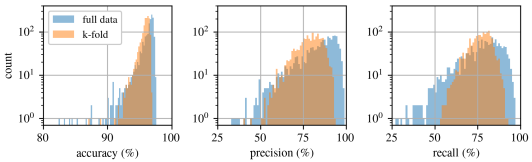
Performance Variability

Repeated training can produce different results.

Observations:

- Accuracy varies
- Precision varies
- Recall varies

Cross-validation reduces this variability and produces more stable estimates.



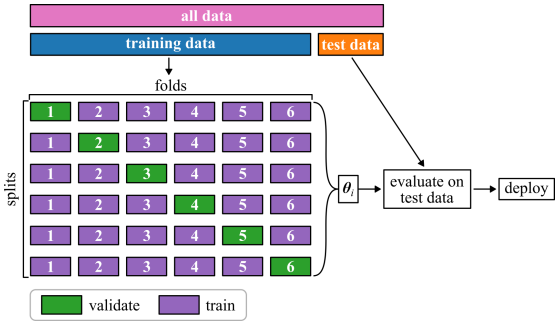
How k-fold Cross-Validation Works

Procedure:

1. Split data into k folds
2. Train on $k-1$ folds
3. Validate on the remaining fold
4. Repeat for all folds
5. Average the results

Advantages:

- Uses all available data
- Better estimate of performance
- Reduces dependence on a single train/test split



Example 4.5: k-fold Cross-Validation

- Train a binary classifier to detect the digit "5"
- Repeatedly train an SGD classifier
- Observe variability in:
  - Accuracy
  - Precision
  - Recall
- Apply:

  `sk.model_selection.cross_val_predict`

- Demonstrate how k-fold cross-validation provides more reliable performance estimates

Multiclass Classification

Binary classifiers distinguish between:

$$2 \text{ classes}$$

Multiclass classifiers distinguish between:

$$N > 2 \text{ classes}$$

For MNIST:

$$10 \text{ classes}$$

corresponding to the digits:

$$0, 1, 2, \ldots, 9$$

A common strategy is to combine multiple binary classifiers.

One-vs-Rest (OvR)

One-vs-Rest trains:

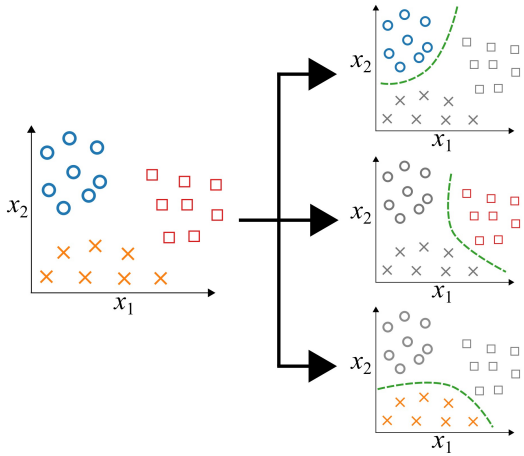
$$N$$

binary classifiers.

For MNIST:

- 0-detector
- 1-detector
- 2-detector
- ...
- 9-detector

The class with the highest score is selected.



One-vs-One (OvO)

One-vs-One trains a classifier for every pair of classes.

Number of classifiers:

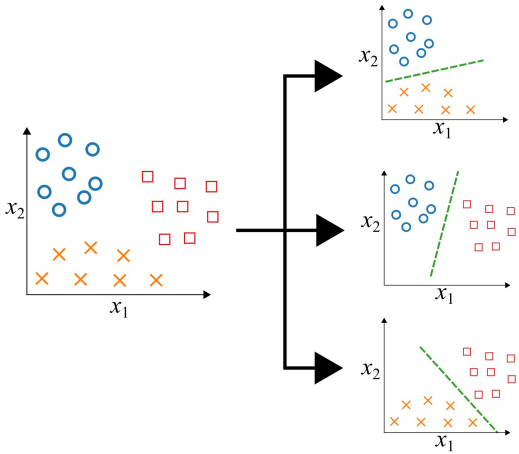
$$\frac{N(N-1)}{2}$$

For MNIST:

$$\frac{10(10-1)}{2} = 45$$

classifiers.

The class with the most wins is selected.



Example 4.6: Multiclass Regularized Linear Classifier

- Extend `SGDClassifier` to multiclass classification
- Compare:
  - One-vs-Rest (OvR)
  - One-vs-One (OvO)
  - Scikit-Learn automatic strategy
- Train classifiers on the MNIST dataset
- Measure computational cost and performance
- Compare strategies for distinguishing the 10 digit classes

Performance Measures for Multiclass Classification

Many of the metrics used for binary classification extend naturally to multiclass problems.

Common evaluation tools include:

- Confusion matrices
- Accuracy
- Precision and recall
- Error analysis

For multiclass problems, the confusion matrix is often the most informative starting point.

Multiclass Confusion Matrix

Rows correspond to:

- Actual digit

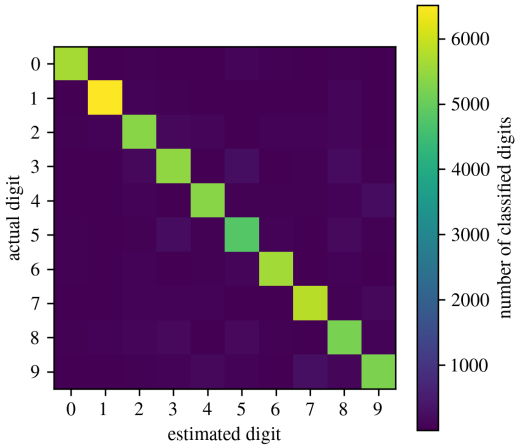
Columns correspond to:

- Predicted digit

A good classifier has:

- Large values on the diagonal
- Small values elsewhere

The MNIST classifier performs well because most entries lie on the diagonal.



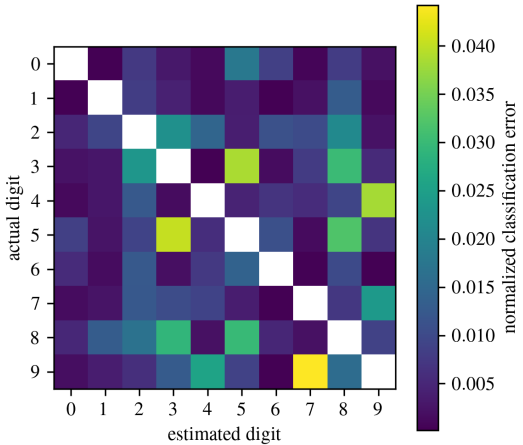
Normalized Error Matrix

To better visualize mistakes:

- Normalize each row
- Remove the diagonal entries
- Display only classification errors

This highlights which digits are most often confused.

Brighter regions correspond to larger error rates.



Analyzing Individual Errors

The classifier frequently confuses:

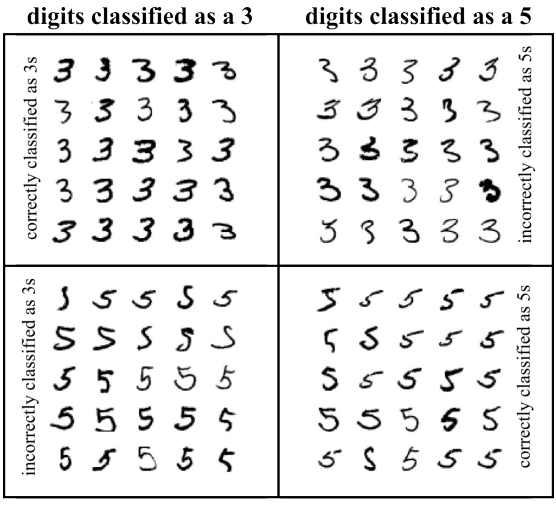
$$3 \leftrightarrow 5$$

Reasons:

- Similar handwritten shapes
- Small stroke differences
- Sensitivity to translation and rotation

Improvement strategies:

- Better preprocessing
- Additional features
- Nonlinear models



Example 4.7: Multiclass Confusion Matrix

- Train a One-vs-One classifier using SGD
- Perform 3-fold cross-validation
- Construct a multiclass confusion matrix
- Visualize:
  - Raw classification counts
  - Normalized classification errors
- Identify commonly confused digits
- Analyze classifier weaknesses
- Use the results to guide model improvements

Chapter 5 Regression-Based Classification

Austin R.J. Downey

Regression-Based Classification

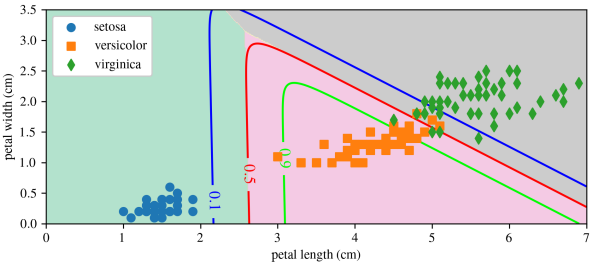
Regression models can often be adapted for classification.

Examples:

- Linear Regression → Logistic Regression
- Logistic Regression → Softmax Regression

Advantages:

- Interpretable models
- Familiar hyperparameters
- Probabilistic predictions



Classification regions for the three Iris species.

Logistic Regression

Logistic Regression estimates the probability that an input belongs to a class.

Predicted probability:

$$\hat{p} = h_\theta(X) = \sigma(\boldsymbol{\theta}^\top X)$$

Classification rule:

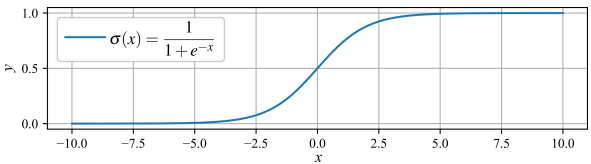
$$\hat{y} = \begin{cases} 0 & \hat{p} < 0.5 \\ 1 & \hat{p} \geq 0.5 \end{cases}$$

- Binary classifier
- Output is a probability
- Threshold typically set to 50%

The Sigmoid Function

The sigmoid function maps any real number to the interval $(0, 1)$: :contentReference[oaicite:0]index=0

- S-shaped curve
- Converts scores into probabilities
- Central component of Logistic Regression



Sigmoid function.

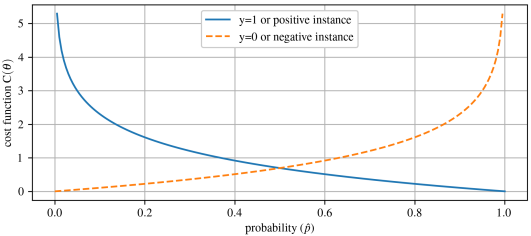
Classification Cost Function

Training Logistic Regression requires a cost function that:

- Rewards correct predictions
- Penalizes incorrect predictions
- Produces probabilities near:
  - 1 for positive examples
  - 0 for negative examples

Observations:

- Large penalty when a confident prediction is wrong
- Small penalty when a confident prediction is correct



Classification cost function.

Training Logistic Regression

Cost for a single sample:

$$C(\theta) = \begin{cases} -\log(\hat{p}) & y = 1 \\ -\log(1 - \hat{p}) & y = 0 \end{cases}$$

Overall cost (log-loss):

$$J(\theta) = -\frac{1}{m} \sum_{i=1}^{m} \left[y^{(i)} \log(\hat{p}^{(i)}) + (1 - y^{(i)}) \log(1 - \hat{p}^{(i)}) \right]$$

- No closed-form solution
- Train using Gradient Descent
- Convex cost function
- Guaranteed global minimum

Data: Iris Flower Dataset

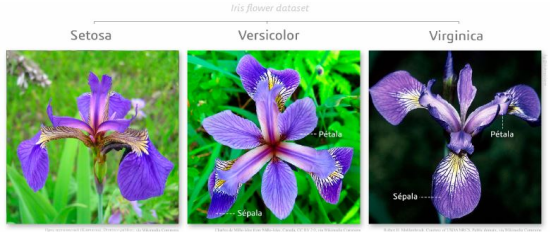
The Iris dataset contains measurements from:

- 150 flowers
- 3 species:
  - Iris-Setosa
  - Iris-Versicolor
  - Iris-Virginica

Features:

- Sepal length
- Sepal width
- Petal length
- Petal width

A classic classification dataset.



Three Iris flower species.

Data: Iris Measurements

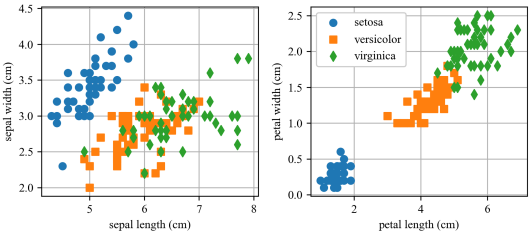
The four measured features can be used to classify flower species.

Observations:

- Setosa is easily separated
- Versicolor and Virginica overlap
- Petal measurements provide better separation than sepal measurements

Goal:

- Predict species from measurements



Sepal and petal measurements for the Iris dataset.

Example 5.1: Iris Dataset Exploration

- Load the Iris dataset from Scikit-Learn
- Examine feature and label information
- Visualize sepal and petal measurements
- Explore relationships between the three species

Demonstrates:

- Classification datasets
- Feature exploration
- Data visualization
- Class separation

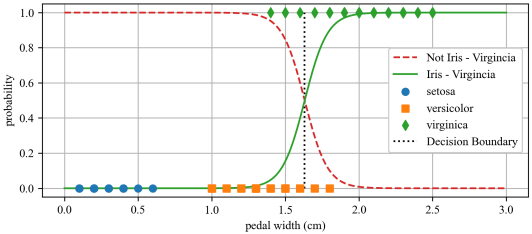
1-D Decision Boundaries

Logistic Regression can classify Iris-Virginica using a single feature:

Petal Width

Observations:

- Below 1 cm:
  - Confidently not Virginica
- Above 2 cm:
  - Confidently Virginica
- Between:
  - Model uncertainty



1-D decision boundary.

Decision Boundary

A Logistic Regression model predicts probabilities:

$$\hat{p} = P(y = 1|X)$$

where P denotes probability. In this case, $P(y = 1|X)$ means the probability that the sample belongs to class 1, given the input features X.

The classification boundary occurs when:

$$\hat{p} = 0.5$$

For the Iris dataset:

- Decision boundary occurs near:

$$\text{Petal Width} \approx 1.6 \text{ cm}$$

- Above the boundary: predict Iris-Virginica; below the boundary: predict not Iris-Virginica.

Example 5.2: 1D Logistic Regression

- Train a Logistic Regression model using:
  - Petal width only
- Estimate:
  - Probability of Iris-Virginica
- Visualize:
  - Predicted probabilities
  - 50% decision boundary
- Demonstrates how probabilities become class predictions

2-D Decision Boundaries

Using two features:

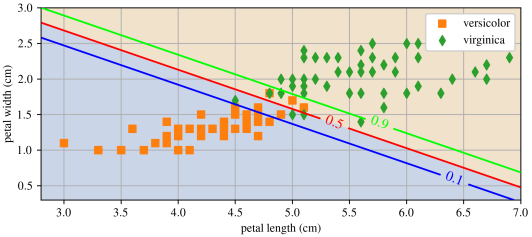
- Petal length
- Petal width

The model predicts:

$$P(\text{Virginica}|X)$$

Features:

- Dashed line:
  - 50% decision boundary
- Contours:
  - Equal probability levels
- Upper-right region:
  - High confidence Virginica



2-D decision boundary and probability contours.

Example 5.3: 2D Decision Boundary

- Train a Logistic Regression model using:
  - Petal length
  - Petal width
- Visualize:
  - Classification regions
  - Decision boundary
  - Probability contours
- Demonstrates how Logistic Regression separates classes in a two-dimensional feature space

Softmax Regression

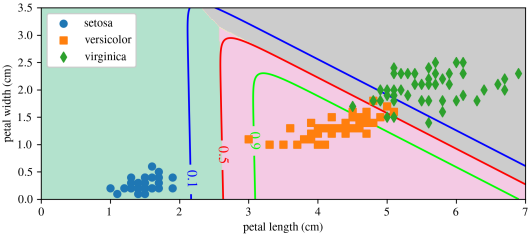
Softmax Regression extends Logistic Regression to multiple classes.

- No need to train several binary classifiers
- Works when each input belongs to exactly one class
- Also called Multinomial Logistic Regression

For an input vector $\mathbf{x}$, the model computes a score for each class.

Example:

- Classify all three Iris species simultaneously



Softmax classification of the three Iris species.

Class Scores and Probabilities

For class k, the score is

$$s_k(\mathbf{x}) = \mathbf{x}^\top \boldsymbol{\theta}^{(k)}$$

The softmax function converts scores into probabilities:

$$\hat{p}_k = \frac{\exp\left(s_k(\mathbf{x})\right)}{\sum_{j=1}^{K} \exp\left(s_j(\mathbf{x})\right)}$$

- $\mathbf{s}(\mathbf{x})$ is the vector of class scores
- K is the number of classes
- All predicted probabilities sum to 1

Prediction Rule

Softmax Regression predicts the class with the largest probability:

$$\hat{y} = \underset{k}{\operatorname{argmax}}\, \hat{p}_k = \underset{k}{\operatorname{argmax}}\, s_k(\mathbf{x}) = \underset{k}{\operatorname{argmax}} \left((\boldsymbol{\theta}^{(k)})^\top \mathbf{x} \right)$$

Note

The argmax operator returns the index of the largest value.

- Softmax Regression is multiclass, not multioutput
- It predicts one class at a time
- Useful for problems with mutually exclusive categories

Training Softmax Regression

The model is trained by minimizing the cross-entropy cost:

$$J(\Theta) = -\frac{1}{m}\sum_{i=1}^{m}\sum_{k=1}^{K} y_k^{(i)} \log\left(\hat{p}_k^{(i)}\right)$$

- Penalizes low probability for the true class
- When $K = 2$, this reduces to log loss
- No closed-form solution
- Train using Gradient Descent or another optimizer

Gradient with respect to class k:

$$\nabla_{\boldsymbol{\theta}^{(k)}} J(\Theta) = \frac{1}{m}\sum_{i=1}^{m}\left(\hat{p}_k^{(i)} - y_k^{(i)}\right)\mathbf{x}^{(i)}$$

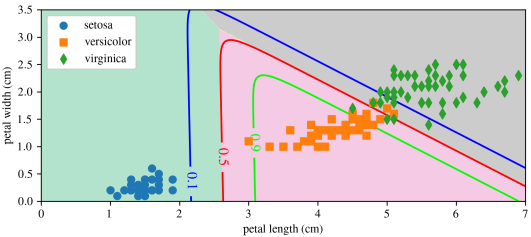
Example 5.4: Softmax Classification

The Iris dataset can be classified using petal length and petal width.

This example:

- Trains a multinomial logistic regression model
- Uses `multi_class="multinomial"`
- Requires a solver such as `lbfgs`

After training, a flower with 5 cm petals is classified as Iris-Virginica with probability 94.2%.



Softmax classification for the three Iris species.

Chapter 6 Decision Trees

Austin R.J. Downey

Decision Trees

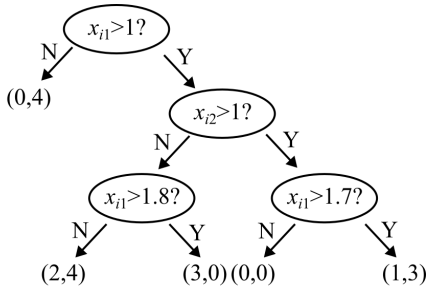
Decision Trees are flexible models used for:

- Classification
- Regression
- Multi-output prediction

Key ideas:

- Split data using simple rules
- Build a tree of decisions
- Make predictions by following a path

They are also the building blocks of Random Forests.



Basic concept of a decision tree.

Decision Trees

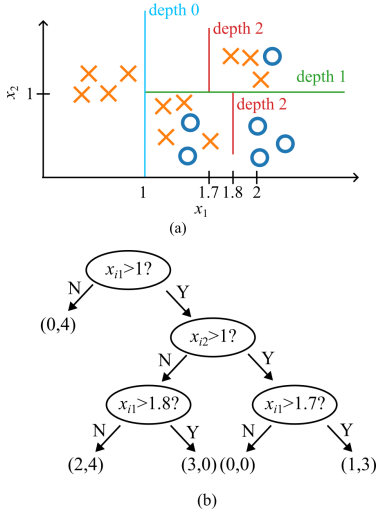
Decision Trees are flexible models used for:

- Classification
- Regression
- Multi-output prediction

Key ideas:

- Split data using simple rules
- Build a tree of decisions
- Make predictions by following a path

They are also the building blocks of Random Forests.



Basic concept of a decision tree.

Decision Tree Classification

A decision tree partitions the feature space into regions.

For the Iris dataset:

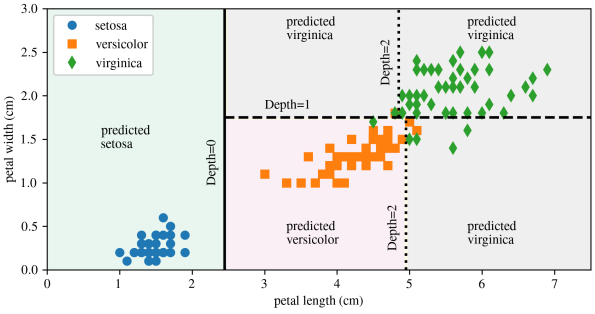
- Root split:

$$\text{petal length} \leq 2.45$$

- Second split:

$$\text{petal width} \leq 1.75$$

Each region is assigned a predicted class.



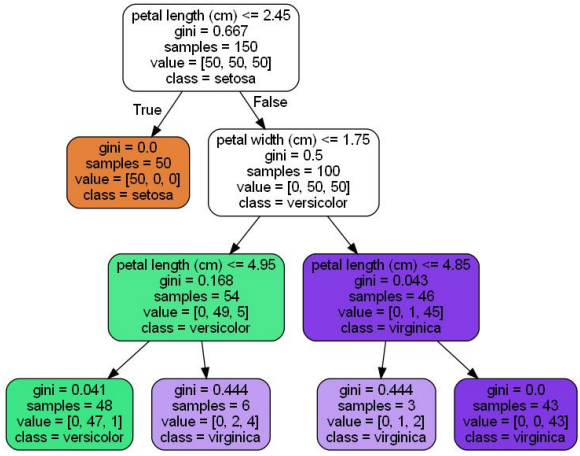
Decision tree decision boundaries.

Reading a Decision Tree

To classify a new sample:

1. Start at the root node
2. Answer the split question
3. Move left or right
4. Continue until a leaf node is reached

The leaf node gives the predicted class.



Decision tree trained on the Iris dataset.

Node Information

Each node in a classification tree stores useful information.

- **class**: predicted class at the node
- **samples**: number of training instances that reach the node
- **value**: class counts at the node
- **gini**: impurity of the node

A node is pure when all samples belong to the same class:

$$G_i = 0$$

Darker node colors indicate purer predictions.

Gini Impurity

Gini impurity measures how mixed the classes are at a node.

$$G_i = 1 - \sum_{k=1}^{n} p_{i,k}^2$$

where:

- $p_{i,k}$ is the proportion of class k at node i
- $G_i = 0$ means the node is pure
- Larger values indicate more mixed classes

Example:

$$1 - \left(\frac{0}{54}\right)^2 - \left(\frac{49}{54}\right)^2 - \left(\frac{5}{54}\right)^2 \approx 0.168$$

Class Probability Estimation

Decision Trees can estimate class probabilities.

Procedure:

1. Follow the sample to a leaf node
2. Count the training samples in that leaf
3. Compute the fraction belonging to each class

Example leaf node:

$$\text{value} = [0,\ 49,\ 5]$$

Probabilities:

$$P(\text{setosa}) = 0\%,\ P(\text{versicolor}) = \frac{49}{54} = 90.7\%, \text{ and } P(\text{virginica}) = \frac{5}{54} = 9.3\%$$

Example 6.1: Decision Tree Classifier

- Use `DecisionTreeClassifier`
- Classify Iris species using petal features
- Train a tree with a limited depth
- Visualize the learned decision rules with Graphviz

The exported tree shows:

- Feature thresholds
- Gini impurity
- Sample counts
- Class distributions

Graphviz viewer:

`https://dreampuf.github.io/GraphvizOnline/`

The CART Training Algorithm

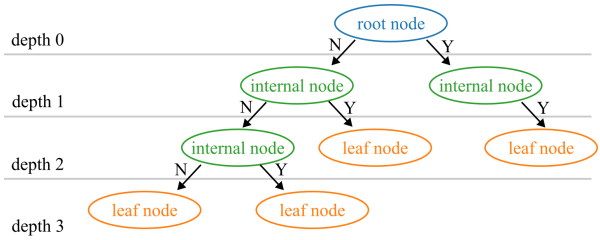
Scikit-Learn trains decision trees using CART:

Classification and Regression Trees

At each node, CART chooses:

- A feature k
- A threshold t_k

The goal is to create child nodes that are as pure as possible.



CART recursively grows the tree.

CART Objective

At each split, CART minimizes:

$$J(k, t_k) = \frac{m_{\text{left}}}{m} G_{\text{left}} + \frac{m_{\text{right}}}{m} G_{\text{right}}$$

where:

- G_{left} and G_{right} are impurity values
- m_{left} and m_{right} are sample counts
- m is the total number of samples at the current node

The tree grows recursively until:

- A maximum depth is reached
- No useful impurity-reducing split remains

Regularizing Decision Trees

Decision Trees can easily overfit when they are allowed to grow too freely.

Regularization works by limiting the complexity of the tree.

Important Scikit-Learn hyperparameters:

- `max_depth`: limits how deep the tree can grow
- `min_samples_split`: requires enough samples before splitting
- `min_samples_leaf`: requires enough samples in each leaf
- `max_leaf_nodes`: limits the number of leaf nodes
- `max_features`: limits features considered at each split

General rule:

- Increase `min_[...]` values to require more data before splitting
- Decrease `max_[...]` values to cap tree size and complexity

Regularization Effects

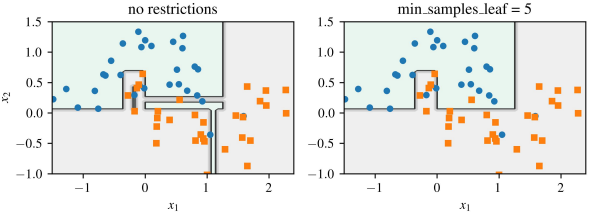
An unrestricted tree may fit noise in the training data.

Adding restrictions can improve generalization.

Example:

$$\texttt{min\_samples\_leaf} = 4$$

This prevents leaves from being created from very small groups of samples.



Regularization using `min_samples_leaf`.

Computational Complexity

Prediction is fast.

A prediction follows one path from the root to a leaf:

$$T_{\text{predict}} = O(\log_2 m)$$

Training is more expensive:

$$T_{\text{train}} = O(nm \log m)$$

where:

- m: number of training instances
- n: number of input features

Decision Trees are especially efficient at prediction time.

Gini Impurity vs. Entropy

Scikit-Learn uses Gini impurity by default.

Entropy is another impurity measure:

$$H_i = -\sum_{k=1}^{n} p_{i,k} \log_2(p_{i,k})$$

Comparison:

- Gini is slightly faster to compute
- Gini often isolates the most frequent class
- Entropy tends to produce more balanced trees

In practice, both usually produce similar trees.

Decision Tree Regression

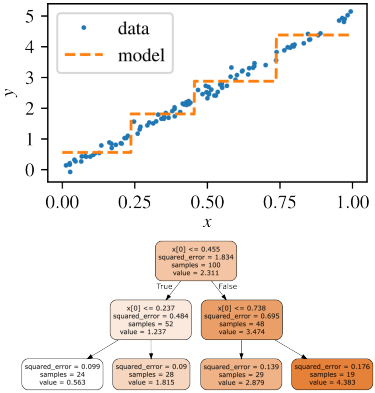
Decision Trees can also perform regression.

Instead of predicting a class, each leaf predicts:

$$\hat{y}_{\text{node}}$$

This value is the average target value of the training instances in that leaf.

The model creates piecewise-constant predictions.



Decision Tree regression model and tree.

Tree Depth in Regression

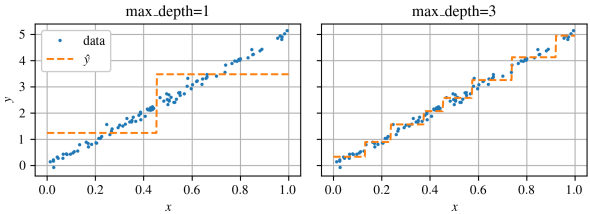
Increasing tree depth creates more prediction regions.

Shallow tree:

- Fewer regions
- Simpler model
- More bias

Deeper tree:

- More regions
- More flexible model
- Higher risk of overfitting



Regression trees with different depths.

Regression Tree Cost Function

For regression, CART minimizes MSE instead of impurity.

$$J(k, t_k) = \frac{m_{\text{left}}}{m}\text{MSE}_{\text{left}} + \frac{m_{\text{right}}}{m}\text{MSE}_{\text{right}}$$

At each node:

$$\text{MSE}_{\text{node}} = \sum_{i \in \text{node}} \left(\hat{y}_{\text{node}} - y^{(i)}\right)^2$$

$$\hat{y}_{\text{node}} = \frac{1}{m_{\text{node}}} \sum_{i \in \text{node}} y^{(i)}$$

The tree chooses splits that reduce prediction error.

Regularizing Regression Trees

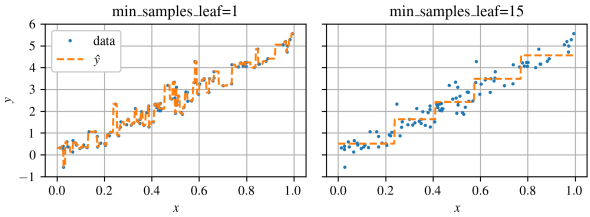
Regression trees can also overfit.

Without regularization:

- Tree follows noise
- Predictions become jagged

With regularization:

- Leaves contain more samples
- Predictions are smoother
- Generalization improves



Effect of `min_samples_leaf` on regression trees.

Example 6.2: Decision Tree Regression

- Use `DecisionTreeRegressor`
- Fit a nonlinear relationship between input and output data
- Train a tree with limited depth
- Visualize the regression tree using Graphviz

This example shows how a tree partitions the input space into regions.

Each region predicts the average target value of the samples it contains.

Instability: Feature Orientation

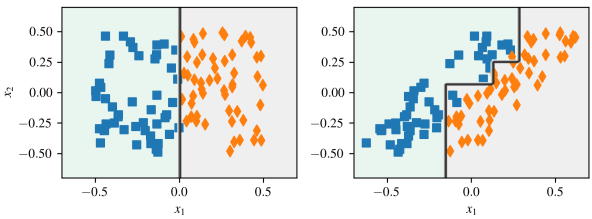
Decision Trees prefer orthogonal splits.

This makes them sensitive to feature orientation.

A simple rotation of the data can produce:

- More complex boundaries
- Poorer generalization
- Less stable predictions

PCA can sometimes help reorient the data.



Sensitivity to feature rotation.

Instability: Training Variation

Decision Trees can change significantly with small training variations.

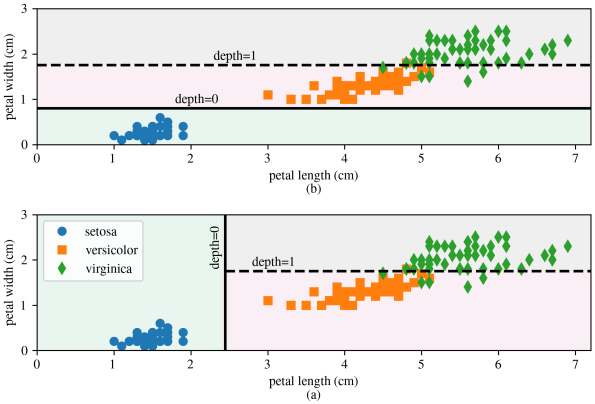
Sources of variation:

- Random number seed
- Equal-quality candidate splits
- Small changes in training data

To improve reproducibility:

`random_state`

can be fixed during training.



Sensitivity to initial conditions.

Random Forests

A Random Forest builds many Decision Trees.

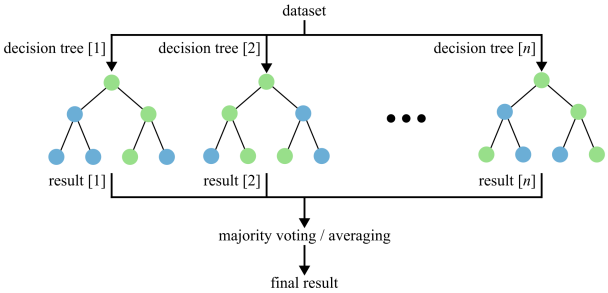
Each tree is trained using:

- A bootstrap sample of the data
- A random subset of features

Predictions are combined by:

- Majority vote for classification
- Averaging for regression

This reduces variance compared with a single tree.



Random forest with majority voting.

Chapter 7 Support Vector Machines

Austin R.J. Downey

Support Vector Machines

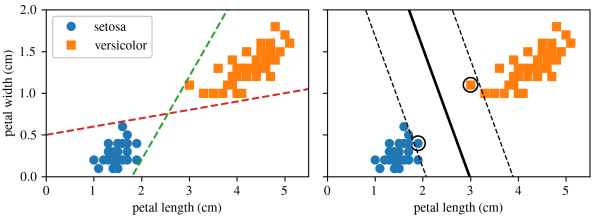
Support Vector Machines are geometric classifiers.

Key idea:

- Separate classes with a decision boundary
- Maximize the margin around that boundary
- Use nearby points to define the boundary

Those nearby points are called:

Support Vectors



Large margin classification.

Large Margin Classification

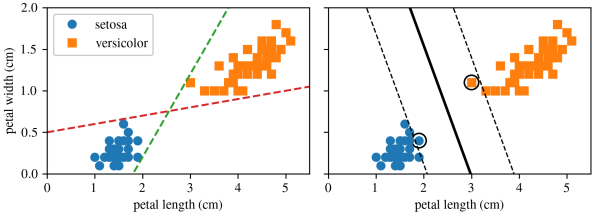
Many linear classifiers may separate the training data.

SVMs choose the classifier that creates the widest possible margin.

This margin is often called the:

street

Only points on the edge of the street directly affect the boundary.



The boundary is shaped by the support vectors.

SVMs Need Feature Scaling

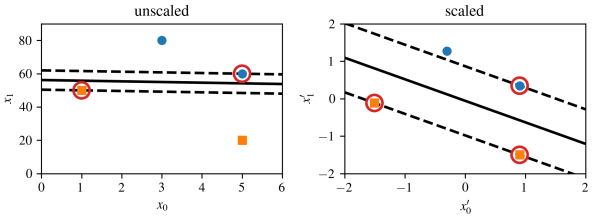
SVMs are sensitive to feature scale.

If one feature dominates the scale, the margin may be distorted.

A common preprocessing step is:

`StandardScaler`

Scaling usually produces a more appropriate decision boundary.



Effect of feature scaling.

Hard Margin Classification

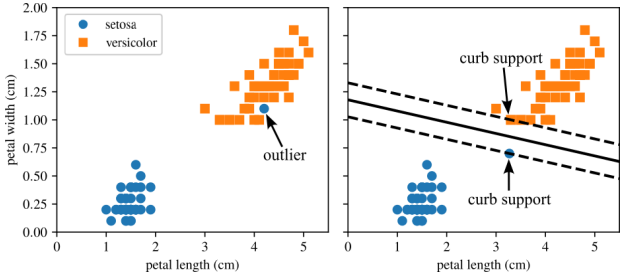
Hard margin classification requires:

- All points correctly classified
- No margin violations

Limitations:

- Only works for linearly separable data
- Very sensitive to outliers

A single outlier can make the hard-margin problem impossible or unstable.



Hard margin sensitivity to outliers.

Soft Margin Classification

Soft margin classification allows some margin violations.

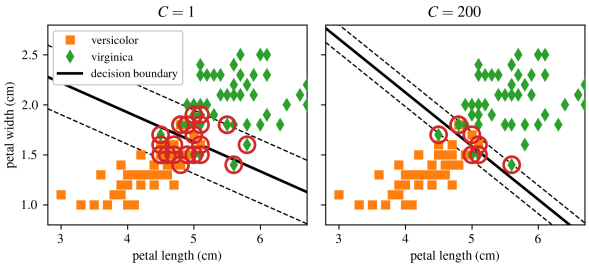
Goal:

- Keep the margin wide
- Limit classification errors
- Improve generalization

The balance is controlled by:

$$C$$

Smaller C gives stronger regularization.



Effect of the C hyperparameter.

The Role of C

The hyperparameter C controls the tradeoff between margin width and violations.

- Smaller C:
  - Wider margin
  - More violations allowed
  - More regularization
- Larger C:
  - Narrower margin
  - Fewer violations allowed
  - Less regularization

Note

Overfitting in an SVM can often be reduced by decreasing C.

SVM Notation

In earlier chapters, model parameters were often written as:

$$\boldsymbol{\theta}$$

For SVMs, we usually write:

$$\mathbf{w} \quad \text{and} \quad b$$

where:

- $\mathbf{w}$: weight vector
- b: bias term
- No extra bias feature $x_0 = 1$ is appended

The decision function is built directly from $\mathbf{w}$, $\mathbf{x}$, and b.

Decision Function and Predictions

A linear SVM computes:

$$\mathbf{w}^\top \mathbf{x} + b = w_1 x_1 + \cdots + w_n x_n + b$$

Prediction rule:

$$\hat{y} = \begin{cases} 0 & \text{if } \mathbf{w}^\top \mathbf{x} + b < 0 \\ 1 & \text{if } \mathbf{w}^\top \mathbf{x} + b \geq 0 \end{cases}$$

The decision boundary occurs when:

$$\mathbf{w}^\top \mathbf{x} + b = 0$$

SVM Decision Function

For two features, the decision function is a plane.

Important levels:

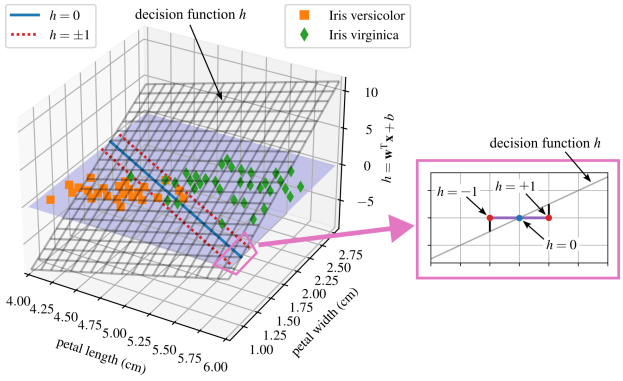
$$h = 0$$

defines the decision boundary.

$$h = 1 \quad \text{and} \quad h = -1$$

define the margin boundaries.

Training adjusts $\mathbf{w}$ and b to make the margin wide.



Decision function for the Iris dataset.

Training Objective

The slope of the decision function depends on:

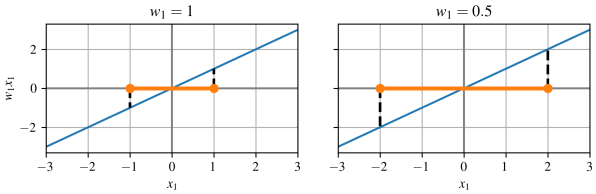
$$\|\mathbf{w}\|$$

Reducing $\|\mathbf{w}\|$ increases the margin.

Therefore, SVM training tries to:

$$\text{minimize } \|\mathbf{w}\|$$

This maximizes the margin between classes.



Smaller weights produce a larger margin.

Hard Margin Objective

Let $t^{(i)} = -1$ for negative samples and $t^{(i)} = 1$ for positive samples.

The hard margin constraint is:

$$t^{(i)} \left(\mathbf{w}^\top \mathbf{x}^{(i)} + b \right) \geq 1$$

The optimization problem is:

$$\begin{aligned} \underset{\mathbf{w}, b}{\text{minimize}} \quad & \frac{1}{2} \mathbf{w}^\top \mathbf{w} \\ \text{subject to} \quad & t^{(i)} \left(\mathbf{w}^\top \mathbf{x}^{(i)} + b \right) \geq 1 \end{aligned}$$

This enforces a margin with no violations.

Why Minimize $\frac{1}{2}\mathbf{w}^\top\mathbf{w}$?

The objective

$$\frac{1}{2}\mathbf{w}^\top\mathbf{w}$$

is equivalent to

$$\frac{1}{2}\|\mathbf{w}\|^2$$

Why use the squared norm?

- It is smooth and differentiable
- Its derivative is simple
- It makes optimization easier

Note

Minimizing $\|\mathbf{w}\|$ directly is harder because it is not differentiable at $\mathbf{w} = 0$.

Soft Margin Objective

Soft margin classification introduces slack variables:

$$\zeta^{(i)} \geq 0$$

Each $\zeta^{(i)}$ measures how much sample i violates the margin.

The optimization problem becomes:

$$\begin{aligned} \underset{\mathbf{w},b,\zeta}{\text{minimize}} \quad & \frac{1}{2}\mathbf{w}^\top \mathbf{w} + C\sum_{i=1}^{m}\zeta^{(i)} \\ \text{subject to} \quad & t^{(i)}\left(\mathbf{w}^\top \mathbf{x}^{(i)} + b\right) \geq 1 - \zeta^{(i)} \end{aligned}$$

The parameter C controls the penalty for violations.

Quadratic Programming

Both hard-margin and soft-margin SVMs are convex quadratic optimization problems.

General quadratic programming form:

$$\begin{aligned} \underset{\mathbf{p}}{\text{minimize}} \quad & \frac{1}{2}\mathbf{p}^\top \mathbf{H}\mathbf{p} + \mathbf{f}^\top \mathbf{p} \\ \text{subject to} \quad & \mathbf{A}\mathbf{p} \leq \mathbf{b} \end{aligned}$$

For a linear SVM:

- $\mathbf{p}$ contains the weights and bias
- The objective encodes margin maximization
- The constraints enforce correct classification or slack penalties

Example 7.1: Support Vector Machine Classification

- Train a linear SVM classifier on Iris petal features
- Scale the data before training
- Derive and plot the decision boundary
- Visualize the margins
- Identify margin violations and support-vector-like points

Performance is evaluated using:

- Confusion matrix
- F1 score

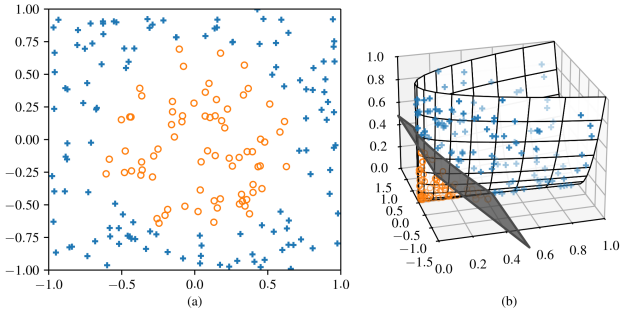
This example demonstrates large-margin classification on real data.

Nonlinear SVM Classification

Many datasets are not linearly separable in their original feature space.

A common strategy is to transform the data into a higher-dimensional feature space.

In the transformed space, a linear boundary may separate the classes.



Nonlinear data transformed to a separable space.

Adding Polynomial Features

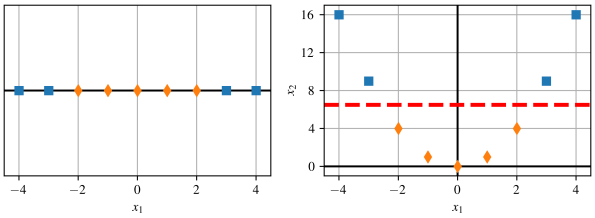
A simple feature transformation can make data linearly separable.

Example:

$$x_2 = x_1^2$$

The original one-dimensional data are not separable.

After adding x_2, the data become separable in two dimensions.



SVM in higher dimensions.

Polynomial Features with LinearSVC

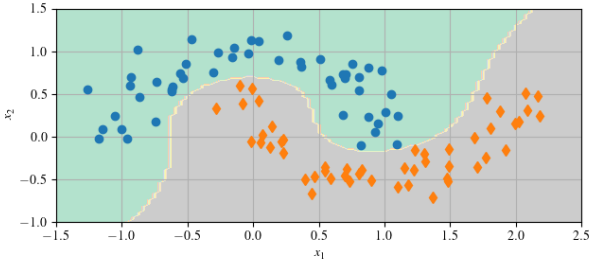
In Scikit-Learn, this can be implemented with a pipeline:

- `PolynomialFeatures`
- `StandardScaler`
- `LinearSVC`

This works well for datasets like:

`make_moons()`

The classifier is still linear in the transformed feature space.



SVM classifier with polynomial features.

Example 7.2: Nonlinear Classification

- Generate nonlinear data using `make_moons`
- Build a pipeline with:
  - `PolynomialFeatures`
  - `StandardScaler`
  - `LinearSVC`
- Train a linear SVM in the expanded feature space
- Visualize the resulting nonlinear decision boundary

This example shows how feature transformations can make nonlinear classification possible.

The Kernel Trick

Polynomial feature expansion can become expensive.

Problems:

- Low-degree features may underfit
- High-degree features may create too many variables
- Computation can become slow

The kernel trick avoids explicitly expanding the feature space.

It replaces:

$$\mathbf{x}^\top \mathbf{z}$$

with:

$$k(\mathbf{x}, \mathbf{z})$$

Kernel Trick Interpretation

The kernel function

$$k(\mathbf{x}, \mathbf{z})$$

computes similarity between two data points.

It allows the SVM to behave as if it were working in a higher-dimensional feature space.

Benefits:

- Curved decision boundaries
- No explicit feature expansion
- Powerful nonlinear classification

The kernel trick is implemented in:

`SVC`

Polynomial Kernel

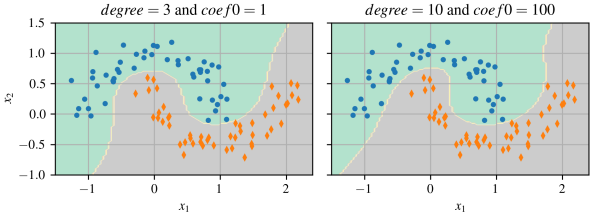
A polynomial kernel can produce nonlinear decision boundaries.

The degree controls model flexibility.

Higher degree:

- More flexible
- More risk of overfitting

The `coef0` parameter controls the influence of lower- vs. higher-degree terms.



Polynomial-kernel SVMs.

Tuning Kernel SVMs

Kernel SVMs can be sensitive to hyperparameters.

Common tuning approach:

- Start with a broad grid search
- Identify promising regions
- Refine with a narrower grid search

Important hyperparameters:

- C
- polynomial degree
- `coef0`
- γ for RBF kernels

Understanding each hyperparameter helps guide the search.

Standard Kernels

Three kernels cover most practical cases.

Polynomial kernel:

$$k_{\text{poly}}(\mathbf{x}, \mathbf{z}) = (\mathbf{x}^\top \mathbf{z} + c)^d$$

RBF kernel:

$$k_{\text{rbf}}(\mathbf{x}, \mathbf{z}) = \exp\left(-\gamma \|\mathbf{x} - \mathbf{z}\|^2\right)$$

Sigmoid kernel:

$$k_{\text{sig}}(\mathbf{x}, \mathbf{z}) = \tanh\left(\kappa \mathbf{x}^\top \mathbf{z} + \theta\right)$$

Choosing a Kernel

A practical rule of thumb:

- Start with the RBF kernel
- Try polynomial kernels when interaction order matters
- Treat the sigmoid kernel as experimental

Note

Kernel methods require scaled features. Standardization is usually important before training.

The best kernel depends on:

- Dataset size
- Noise level
- Feature interactions
- Desired model flexibility

Example 7.3: Polynomial Kernel Trick

- Use `SVC` instead of `LinearSVC`
- Apply a third-degree polynomial kernel
- Classify nonlinear `make_moons` data
- Use a Scikit-Learn pipeline
- Visualize the curved decision boundary

This example shows how the kernel trick produces nonlinear classification without explicitly building polynomial features.

Computational Complexity

Scikit-Learn provides several SVM-related classifiers.

class	time complexity	out-of-core	kernel trick
`LinearSVC`	$O(mn)$	no	no
`SVC`	$O(m^2n)$ to $O(m^3n)$	no	yes
`SGDClassifier`	$O(mn)$	yes	no

Scaling is required for all three methods.

LinearSVC vs. SVC

`LinearSVC`

- Uses `liblinear`
- Linear decision boundary only
- Scales approximately as $O(mn)$
- Good for large, high-dimensional datasets

`SVC`

- Uses `libsvm`
- Supports kernels
- Can model nonlinear boundaries
- Slower for large datasets

Support Vector Regression

Support Vector Regression adapts SVMs to continuous outputs.

Instead of separating classes, SVR fits a prediction curve surrounded by a tube.

Tube width:

$$\epsilon$$

Points inside the tube do not affect the loss.

The regularization parameter C controls how strongly violations outside the tube are penalized.

Linear SVR

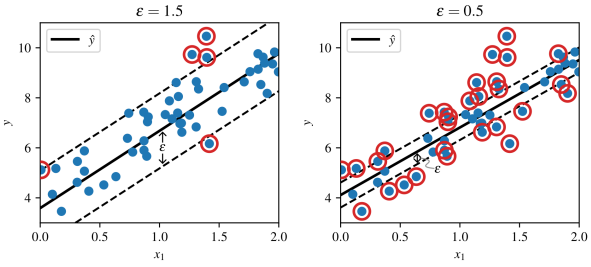
Linear SVR fits a linear function with an ϵ-insensitive margin.

Larger ϵ:

- Wider tube
- More tolerance

Smaller ϵ:

- Narrower tube
- More sensitive fit



Linear SVR with different ϵ values.

Kernel SVR

When the relationship is nonlinear, SVR can use kernels.

Prediction function:

$$f(x) = \sum_{i=1}^{m} (\alpha_i - \alpha_i^*) k(x_i, x) + b$$

Common kernels:

- RBF
- Polynomial

Kernel SVR is powerful for small to medium datasets but can become slow as the dataset grows.

Polynomial SVR

A polynomial kernel can model nonlinear regression behavior.

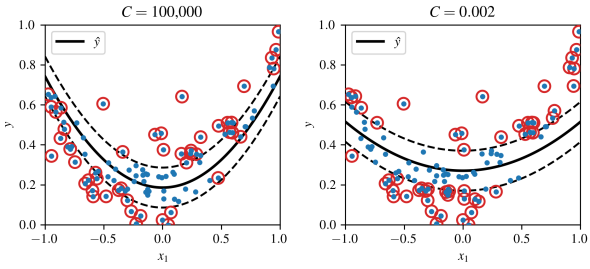
The parameter C controls regularization.

Larger C:

- Less regularization
- Tighter fit

Smaller C:

- More regularization
- Smoother fit



Polynomial-kernel SVR.

Example 7.4: SVM Polynomial Regression

- Fit noisy quadratic data using SVR
- Use a polynomial kernel
- Introduce an ϵ-insensitive margin
- Visualize the regression curve and tolerance region

This example demonstrates how SVR can model nonlinear relationships while controlling how closely the model follows the data.