Machine Learning and Data Mining

Jil Zerndt
FS 2025

Intro

Definition of Machine Learning

Machine Learning (ML) is a branch of artificial intelligence (AI) devoted to developing and understanding methods that "learn", i.e., that leverage data to make predictions or decisions without being explicitly programmed to do so.
⇒ Study of algorithms and statistical models without using explicit instructions, relying on patterns and inference instead

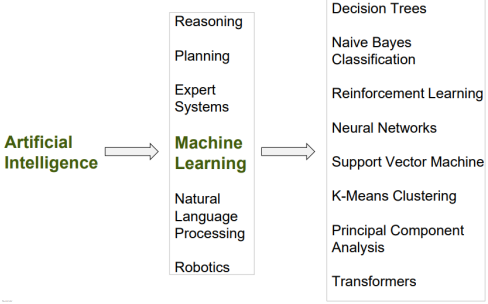
Purpose of Machine Learning

The computational methods in ML are used to discover patterns in data and/or derive a corresponding generating process to:

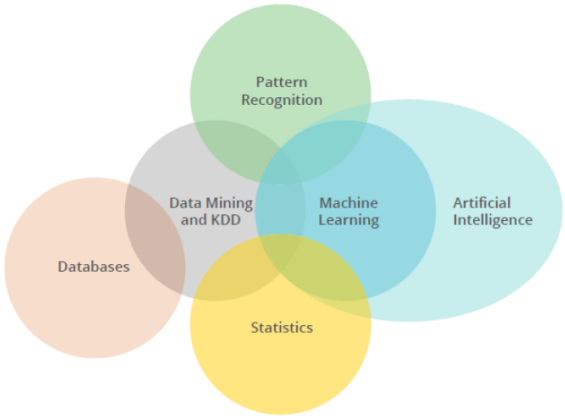
- Gain insights and predict events
- Provide a quantitative basis for decisions (actionable insights)
- Influence the underlying process of the data

⇒ Branch of AI devoted to developing and understanding methods that "learn", i.e. that leverage fata to make predictions or decisions (act like humans) without being explicity programmed to do so

What's behind the Magic?



Overview of Machine Learning



Model

A model is a logical, mathematical or probabilistic relationship between several variables.

Learning (Training)

Machine Learning employs adaptive models, which are configured and parameterized automatically based on the training data.

Overview of important Concepts

Data Mining

- Discovering patterns in large data sets
- Extraction of patterns and knowledge from large amounts of data

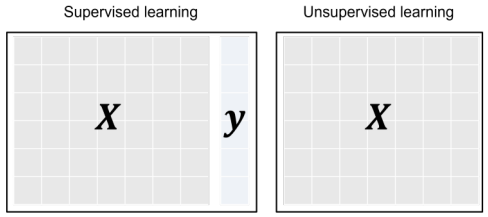
Deep Learning

- Subset of machine learning where artificial neural networks (Deep Neural Networks), algorithms inspired by the human brain, learn from large amounts of data
- Uses multiple layers to progressively extract higher level features (attributes) from the raw input

Reinforcement Learning (Trial and Error)

- Concerned with how software agents take actions in an environment in order to maximize some notion of cumulative reward

Supervised vs. Unsupervised Learning

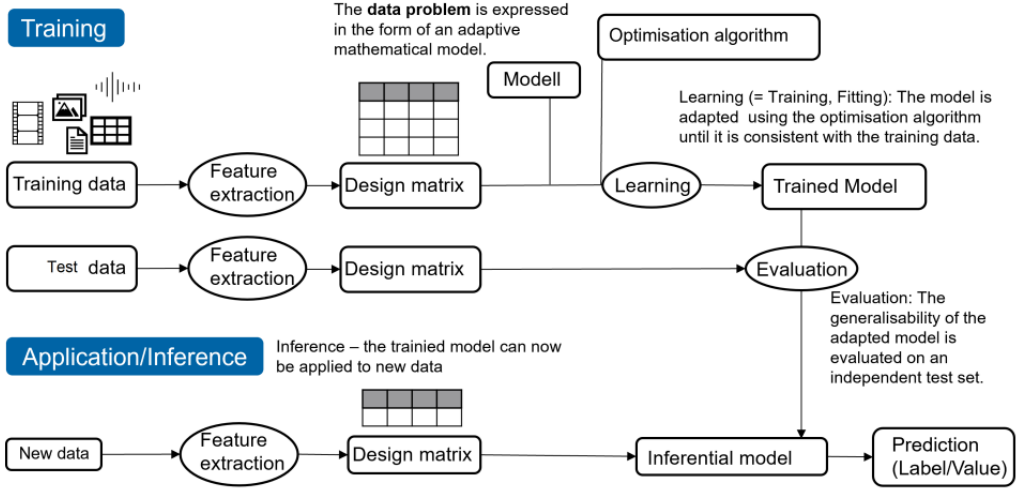


Machine Learning Paradigms

Supervised Learning

In Supervised Learning, we use a dataset with annotated training samples to 'teach' a machine to perform a certain task. The training data consists of pairs of inputs and their associated output values.

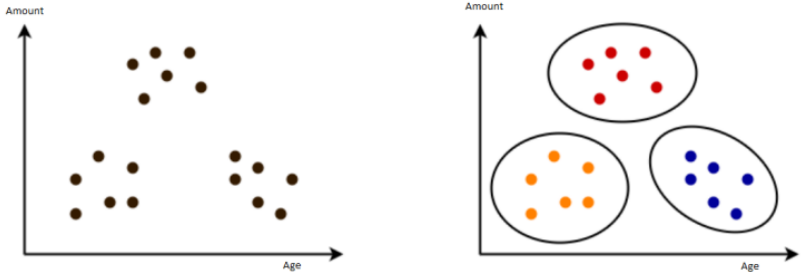
- The goal is to find a function f that maps input data to their corresponding outputs.
- Input features are also called Independent Variables, Predictors, Attributes, or Covariates.
- The algorithm learns from labeled training data, and makes predictions (class, value) on unseen data
- Example: Classification, Regression (see script for math shit)



Unsupervised Learning

In Unsupervised Learning, the training data does not contain any expected output values. The goal is to model the underlying distribution of the data to explain it and apply the model to new data.

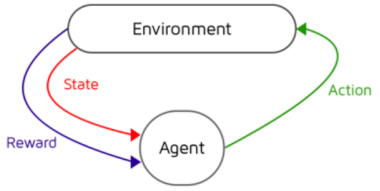
- The problem statement is fuzzier than in supervised learning
- Evaluation is more difficult without test data including expected output values
- The algorithm learns from unlabeled data, and determines data patterns / groupings / clusters



Reinforcement Learning

In Reinforcement Learning, the learning system (called an Agent) can observe the Environment, select and perform Actions, and get Rewards in return. It must learn by itself what is the best strategy (called a Policy) to get the most reward over time.

- The algorithm learns to perform an action from experience
- Example: Game playing, Robotics



Supervised Learning Details

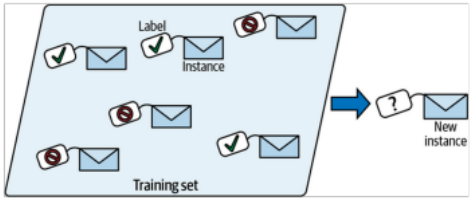
Clustering is the task of grouping a set of objects in such a way that objects in the same group (called a cluster) are more similar (in some sense) to each other than to those in other groups.

Classification is the problem of identifying to which of a set of categories a new observation belongs, on the basis of a training set of data containing observations (or instances) whose category membership is known. E.g. correctly classifying (assigning a label to) an email as spam or not spam.

Classification

Target variable y: categorical

$$y_m \in \{C_1, C_2, \ldots, C_K\}$$

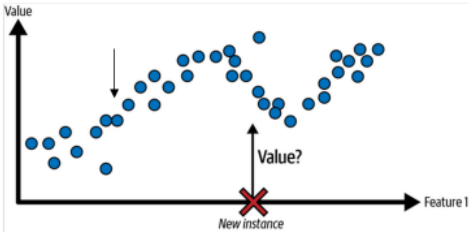


Regression is the problem of predicting and forecasting a concrete number based on a set of data containing observations whose category is known.

Regression

Target variable y: numerical - continuous

$$y_m \in \mathbb{R}$$



Classification vs. Regression

- **Classification**: The target variable is categorical (typically nominal scale). The output values belong to a set of discrete classes

$$y^{(m)} \in \{C_1, C_2, \ldots, C_K\}$$

Example: Spam filter classifying emails as 'spam' or 'not spam'.

- **Regression**: The target variable is a numeric (continuous) value:

$$y^{(m)} \in \mathbb{R}$$

Example: Predicting the price of a car from features like mileage, age, brand, etc.

Evaluating Supervised ML Models

We typically estimate model performance by measuring the quality on a Testset, which contains data samples not used during training.

- The data is split into Training set and Testset
- The model is trained on the training set and evaluated on the testset
- For regression, we use metrics like Mean Squared Error
- For classification, we use metrics like Accuracy, Precision, and Recall

Supervised Learning Pipeline

A typical supervised learning pipeline includes:

1. Computing features for the samples (creating the Design Matrix)
2. Selecting and training a machine learning algorithm/model
3. Evaluating model performance on a test set
4. Applying the final model to predict outcomes for new data

This process is typically iterated several times to improve model quality.

Building a Supervised Learning Model

Define the problem

Clearly specify what you want to predict and what data you have available.

Prepare the data

- Collect relevant data
- Clean the data (handle missing values, outliers)
- Split into training, validation, and test sets (e.g., 70%, 15%, 15%)

Feature engineering

- Select relevant features
- Transform features if needed (normalization, encoding categorical variables)
- Create new features if beneficial

Model selection and training

- Choose appropriate algorithms based on the problem
- Train models with different hyperparameters
- Evaluate on validation set and tune hyperparameters

Final evaluation and deployment

- Evaluate final model on test set
- Deploy model for making predictions on new data
- Monitor performance and retrain periodically if needed

Supervised Learning Example Consider a spam filter:

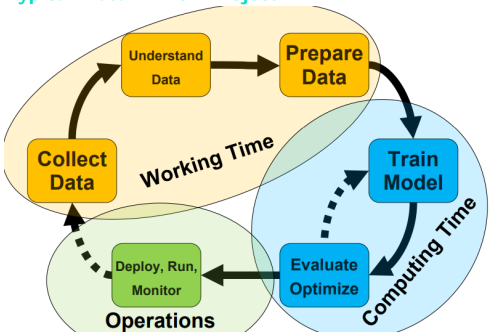
- Input: Email text (converted to features like word frequencies)
- Output: Binary classification (spam or not spam)
- Training: The model learns patterns from labeled emails
- Inference: New emails are classified based on learned patterns
- Evaluation: Accuracy might be 98% (98 out of 100 emails correctly classified)

Data Preprocessing

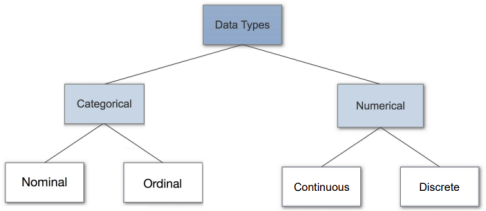
Learning Objectives:
- Understand fundamental importance of data preprocessing
- Know basic algorithms for data cleaning, (near) duplicate detection and filling missing values

Data

Typical Data Driven Project



Data Types



Overview Data Types

Categorial Data
- Nominal: no order, Scale ("labels") → e.g. hair colour, gender
- Ordinal: ordered → e.g. military rank, star rating

Numerical Data (ordered)
- Discrete: countable, ratio → e.g. number of persons in a room
- Continuous: interval, numeric scale → e.g. temperature, weight

Data has many sources, e.g.: sensor, survey, simulation, social media, textual, financial, multimedia, ERP systems data, etc.
Independent of the data source, each data point has a data type

Nominal Data
- Nominal scales are used for **labelling** variables, without any quantitative value
- No numerical significance
- Nominal data has no order
- Scales could simply be called labels
- Examples: gender, hair colour, race, marital status

Ordinal Data
- Represents **discrete and ordered** units
- Nearly the same as nominal data, but **order matters**
- No distance between the different categories
- Examples: military rank, star rating, education level

Discrete Numeric Data
- Represents items that can be **counted**
- Values may go from 0, 1, 2, on to infinity (making it countably infinite)
- Examples: number of persons in a room, number of "headsïn 60 coin flips, time elapsed in minutes

Continuous Numeric Data
- Also known as **interval data**
- Often measurements
- Possible values **cannot be counted** and can only be described using intervals on the real number line
- Examples: temperature, weight, height, time, ...

Data Quality

Standard error measure

$$E = \frac{1}{N}\sum_{i=1}^{N}\left(1 - \text{ id } (\hat{y}_i, y_i)\right), \quad \text{id } (a,b) = \begin{cases} 1 & \text{if } a = b \\ 0 & \text{else} \end{cases}$$

$$\text{data } a_{\text{size}} = 9, \quad \text{correct } = 6, \quad \text{wrong } = 3$$

$$E = \frac{1}{9} \cdot 3 = 0.33$$

Data Cleaning is the process of improving the data quality by removing or improving incorrect or improperly formatted data.

(near) duplicate detection is the process of identifying and removing or merging duplicate data points.
- compare attributes of the tuple
- compare content of the attributes

Filling missing values is the process of replacing missing values with substituted values.
- ignore tuple
- fill in missing value manually
- use global constant such as ünknownör 1"
- use attribute mean, median, mode
- use most probable value

Noisy data is data with errors or outliers.
- Binning: divide the range of attribute values into bins
- Regression: smooth data by fitting the data into a function
- Clustering: detect and remove outliers

Data Sampling Represent large dataset by smaller subset to speed up automatic calculations.

Non-Probabilistic
- Convenience: easiest to obtain
- Judgement: based on experts' knowledge and judgement
- Snowball: purely based on referrals
- Quota: based on attribute values

Probabilistic
- Simple Random: each item has an equal probability of being chosen
- Systematic: select some starting point and then select every kth element
- Stratified: divide the population into subgroups (strata) based on attributes and then draw a sample from each stratum
- Cluster: divide the population into clusters and then randomly select some of the clusters

Data Partitioning is the process of dividing the dataset into two or more parts.
- Training set: used to train the model
- Validation set: used to tune the model
- Test set: used to evaluate the model

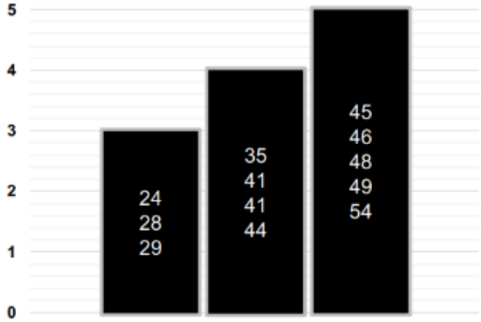
K-Fold Cross-Validation
- Divide the dataset into k subsets (folds)
- Train the model k times, each time using a different subset as the test set and the remaining points as the training set → train on k-1 folds, test on the remaining fold, repeat for each fold
- Average the results to get the final model → calculate average errors

Equal Width Binning

Divide the range into N intervals of equal size (= width)

$$width = \frac{max - min}{N}$$

$$bin_i = [min + i \cdot width, min + (i+1) \cdot width]$$

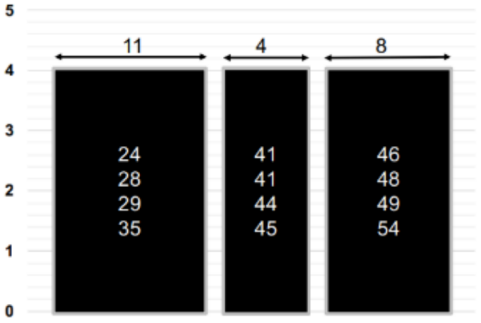


Equal Depth/Frequency Binning

Divide the range into N intervals with equal number of data points/records (= depth/frequency)

$$depth = \frac{N}{n}$$

$$bin_i = [data_{(i-1)\cdot depth}, data_{i \cdot depth}]$$



Data Normalization is the process of transforming values of several variables into a similar range.

$$\text{Min-Max Normalization: } x_{norm} = \frac{x - min(x)}{max(x) - min(x)}$$

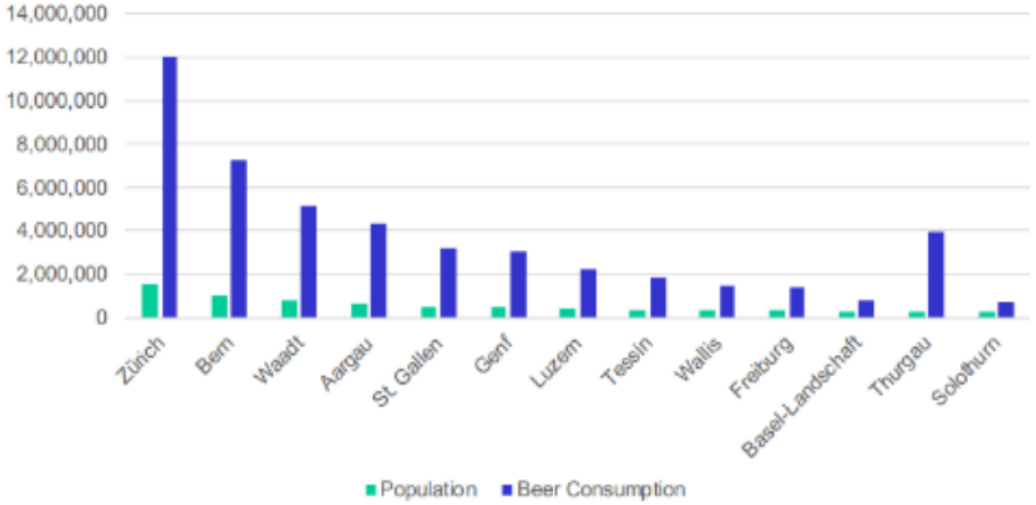
$$\text{Z-Score Normalization: } x_{norm} = \frac{x - \bar{x}}{\sigma}$$

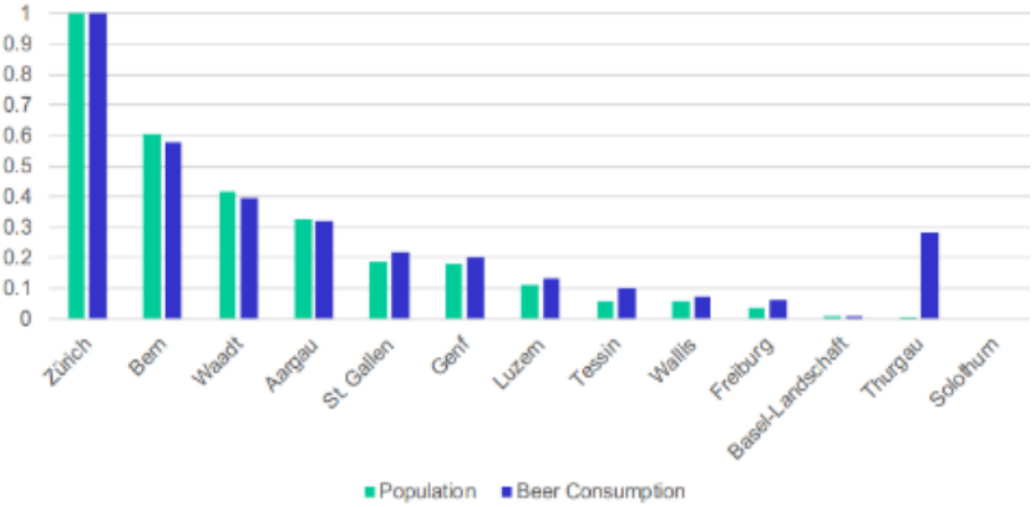
Change the values of numeric columns to a common scale (e.g., between 0 and 1), without distorting differences in the ranges of values.

$$\text{Linear Normalization: } f_{lin}(v) = \frac{v - min}{max - min}$$

$$\text{Square Root Normalization: } f_{sq}(v) = \frac{\sqrt{v} - \sqrt{min}}{\sqrt{max} - \sqrt{min}}$$

$$\text{Logarithmic Normalization: } f_{ln}(v) = \frac{\ln(v) - \ln(min)}{\ln(max) - \ln(min)}$$



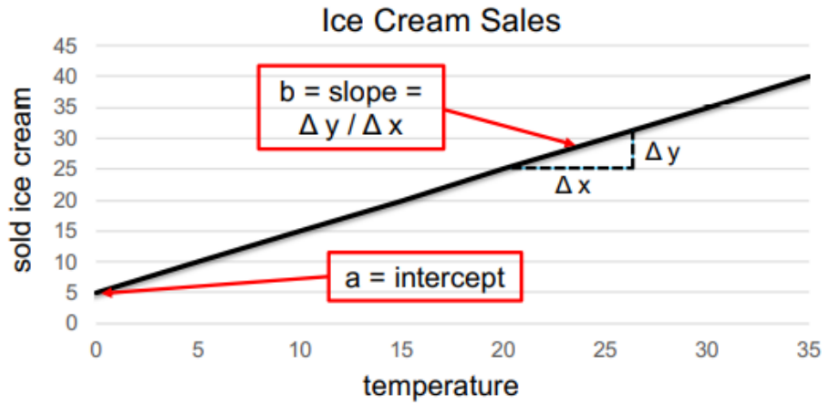


Basics of Linear Regression

Linear Regression Discover the parameters of the straight-line equation that best fits the data points.

- $\bar{x}$ is the mean value of x
- $\bar{y}$ is the mean value of y

$$b = \frac{\sum_{i=1}^{n}(x_i - \bar{x})(y_i - \bar{y})}{\sum_{i=1}^{n}(x_i - \bar{x})^2}, \quad a = \bar{y} - b\bar{x}, \quad y = a + bx$$



Usage: smooth out noise, fill in missing values, predict future/unknown values

Univariate Linear Regression

In Univariate (Simple) Linear Regression, the output depends on one input variable x, and the hypothesis is a linear combination:

$$h_{\theta_0,\theta_1}(x) = \theta_0 + \theta_1 x$$

where θ_0 represents the intercept with the y-axis and θ_1 represents the slope of a straight line.

Multivariate Linear Regression

Multivariate (Multiple) Linear Regression extends the simple model to handle multiple predictors:

$$\hat{y}^{(m)} = h_\theta(x^{(m)}) = \theta_0 + \theta_1 x_1^{(m)} + \theta_2 x_2^{(m)} + \ldots + \theta_N x_N^{(m)} = \theta^T X_{m,:}$$

This can be expressed compactly in matrix form:

$$y = X\theta + \varepsilon$$

where X is the design matrix, θ is the parameter vector, y is the output vector, and ε is the error term.

Cost Function for Linear Regression

We use the Residual Sum of Squares (RSS) to measure how well our model fits the data:

$$J(\theta_0, \theta_1) = \frac{1}{2M}\sum_{m=1}^{M}(y^{(m)} - \hat{y}^{(m)})^2$$

This is also called the Least Squares Approach.

Normal Equation

For linear regression, we can directly compute the optimal parameters using:

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

This gives the exact solution without requiring iterative approximation.

Linear Regression Models

Training Linear Regression Models

Training a Linear Regression Model

Steps for training linear regression models

1. Collect your training data consisting of feature vectors $x^{(m)}$ and target values $y^{(m)}$
2. Decide whether to use the normal equation or gradient descent:
   - Normal equation: Use for datasets with fewer than 20,000 features or samples
   - Gradient descent: Use for larger datasets
3. For normal equation:
   - Construct design matrix X with rows as samples, adding a column of 1s for the intercept term
   - Calculate $\theta = (X^T X)^{-1} X^T y$
4. For gradient descent:
   - Initialize parameters θ randomly
   - Update parameters iteratively using gradient descent until convergence
5. After training, predict using $\hat{y} = \theta^T x$

Linear Regression Example Consider predicting life satisfaction based on GDP:

- Input feature: GDP per capita (USD)
- Output: Life satisfaction score (1-10)
- Training the model gives us parameters $\theta_0 = 3.0$ and $\theta_1 = 8 \times 10^{-5}$
- For a country with GDP = 45,000 USD, we predict: $\hat{y} = 3.0 + (8 \times 10^{-5}) \times 45000 = 3.0 + 3.6 = 6.6$

Evaluating Regression Models

Evaluation Metrics for Regression

Common metrics for evaluating regression models include:
Mean Absolute Error:

$$MAE = \frac{1}{I} \sum_{i=1}^{I} |y^{(i)} - \hat{y}^{(i)}|$$

Mean Squared Error:

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

Root Mean Squared Deviation:

$$RMSD = \sqrt{MSE} = \sqrt{\frac{1}{I} \sum_{i=1}^{I} (y^{(i)} - \hat{y}^{(i)})^2}$$

Assumptions in Linear Regression

- **Linearity**:
  The relationship between X and y is linear
- **Independence**:
  The residuals are independent of each other
- **Normality**: The expected output values are normally distributed
- **Homoscedasticity**: The variance of the residual is the same for any value of X

These assumptions can be verified visually using residual plots.

Coefficient of Determination

The Coefficient of Determination R^2 measures the fraction of the variance explained by the model:

$$R^2 = 1 - \frac{SS_{res}}{SS_{tot}}$$

Residual sum of squares:

$$SS_{res} = \sum_i (y^{(i)} - \hat{y}^{(i)})^2$$

Total sum of squares:

$$SS_{tot} = \sum_i (y^{(i)} - \mu_y)^2$$

$R^2 = 1$ means a perfect fit, while $R^2 = 0$ means the model performs no better than predicting the mean.

Residual Analysis

Residual plots help evaluate model quality:

- **Random scatter**:
  Suggests good model fit
- **U-shaped pattern**:
  Suggests non-linear relationship
- **Funnel shape**:
  Suggests heteroscedasticity
- **Cyclic pattern**:
  Suggests seasonality or auto-correlation

Examining residuals can indicate model weaknesses and suggest improvements.

Interpreting Regression Results

Examine coefficient values

- Sign indicates direction of relationship (positive or negative)
- Magnitude indicates strength of relationship
- For standardized features, directly compare coefficient magnitudes

Assess model fit

- R^2 close to 1 indicates good fit
- Low MSE indicates accurate predictions
- Check residual plots for patterns

Check for violations of assumptions

- Non-linear patterns in residual plot suggest linearity violation
- Residuals changing with fitted values suggest heteroscedasticity
- QQ-plot deviating from straight line suggests non-normality

Consider feature importance

- Features with larger coefficients have greater impact
- Statistical significance (p-values) indicates confidence in relationship

Making Predictions with Linear Regression

Suppose we've trained a multivariate linear regression model to predict house prices based on size (sq ft), number of bedrooms, and age (years). Our trained model has parameters:

- $\theta_0 = 50,000$ (intercept)
- $\theta_1 = 100$ (coefficient for size)
- $\theta_2 = 5,000$ (coefficient for bedrooms)
- $\theta_3 = -200$ (coefficient for age)

For a house with 1,500 sq ft, 3 bedrooms, and 25 years old, we predict:

$$\begin{aligned}\hat{y} &= \theta_0 + \theta_1 \times \text{size} + \theta_2 \times \text{bedrooms} + \theta_3 \times \text{age} \\ &= 50,000 + 100 \times 1,500 + 5,000 \times 3 + (-200) \times 25 \\ &= 50,000 + 150,000 + 15,000 - 5,000 \\ &= 210,000\end{aligned}$$

So, the predicted price is $210,000.

Gradient Descent

Motivation and Basics

Gradient Descent

Gradient Descent is an optimization algorithm for finding the minimum of a function by iteratively moving in the direction of steepest descent. For a cost function $J(\theta)$, the update rule is:

$$\theta_j = \theta_j - \alpha \frac{\partial}{\partial \theta_j} J(\theta) \quad \forall j = 0, ...n$$

where α is the learning rate.

Why Gradient Descent?

The explicit solution for linear regression using the normal equation doesn't work well for large datasets because:

- Matrix operations have approximately cubic runtime complexity
- Normal equation becomes too time-consuming for more than 20,000 features or samples
- The normal equation might become numerically unstable if features are highly correlated

In these cases, gradient descent provides a more efficient approach.

Implementing Gradient Descent for Linear Regression

Initialize parameters

Start with random values for parameters $\theta_0, \theta_1, ..., \theta_n$

Repeat until convergence

For each parameter θ_j, update it simultaneously:

$$\theta_j = \theta_j - \alpha \frac{1}{M} \sum_{m=1}^{M} (h_\theta(x^{(m)}) - y^{(m)}) x_j^{(m)}$$

where:

- $h_\theta(x^{(m)})$ is the current prediction for input $x^{(m)}$
- $y^{(m)}$ is the expected output
- $x_j^{(m)}$ is the j-th feature of the m-th training example (with $x_0^{(m)} = 1$)

Key considerations

- Parameters must be updated simultaneously, not sequentially
- The learning rate α controls the step size
- A large learning rate might cause divergence
- A small learning rate leads to slow convergence

Gradient Descent Step-by-Step Consider a simple linear regression with two data points: (1,1) and (2,2).

- Start with parameters $\theta_0 = 0, \theta_1 = 0$
- Initial prediction: $\hat{y} = 0 + 0 \times x = 0$
- Error for first point: $1 - 0 = 1$
- Error for second point: $2 - 0 = 2$
- With learning rate $\alpha = 0.1$, after first iteration:
  - $\theta_0 = 0 - 0.1 \times \frac{1}{2} \times (1 + 2) = 0 - 0.15 = -0.15$
  - $\theta_1 = 0 - 0.1 \times \frac{1}{2} \times (1 \times 1 + 2 \times 2) = 0 - 0.25 = -0.25$
- After several iterations, parameters converge to $\theta_0 = 0, \theta_1 = 1$

Types of Gradient Descent

Batch Gradient Descent

Batch gradient descent uses all training examples in each iteration. For each parameter θ_j:

$$\theta_j = \theta_j - \alpha \frac{1}{M} \sum_{m=1}^{M} (h_\theta(x^{(m)}) - y^{(m)}) x_j^{(m)}$$

Advantages:

- Computes the true gradient of the cost function
- More stable convergence

Disadvantages:

- Slow for very large datasets
- Requires all data to be in memory

Stochastic Gradient Descent (SGD)

Stochastic gradient descent updates parameters using only one randomly selected training example in each iteration:

$$\theta_j = \theta_j - \alpha (h_\theta(x^{(m)}) - y^{(m)}) x_j^{(m)}$$

Advantages:

- Much faster for large datasets
- Can process data online (one example at a time)

Disadvantages:

- More erratic updates and convergence
- May require more iterations

Mini-Batch Gradient Descent

Mini-batch gradient descent is a compromise that updates parameters using a small batch of training examples (typically 10-1000) in each iteration:

$$\theta_j = \theta_j - \alpha \frac{1}{b} \sum_{m \in B} (h_\theta(x^{(m)}) - y^{(m)}) x_j^{(m)}$$

where B is the mini-batch and b is its size. Advantages:

- More efficient than batch gradient descent
- More stable than stochastic gradient descent
- Can leverage vectorized operations

Learning Rate Optimization

Learning Rate

The learning rate α in gradient descent controls the step size at each iteration. It affects convergence:

- Too small: algorithm converges very slowly
- Too large: algorithm might overshoot the minimum and diverge

Learning Rate Optimization

Several strategies can improve learning rate effectiveness:

- **Decay Rate**: Start with a larger learning rate and reduce it over time using $\alpha_t = \frac{1}{1 + decay\_rate \times t} \alpha_0$
- **Adaptive Methods**: Use algorithms like Adam or Adagrad that adjust learning rates based on the behavior of gradients

Choosing the Right Learning Rate

Start with a sensible default

Begin with a moderate learning rate (e.g., 0.01 or 0.001)

Learning rate search

- Try a range of learning rates (e.g., 1.0, 0.1, 0.01, 0.001, 0.0001)
- Plot the learning curves (cost vs. iterations)
- Too high: cost increases or oscillates wildly
- Too low: cost decreases very slowly
- Just right: cost decreases steadily and quickly

Adaptive learning rates

Consider using adaptive optimization algorithms:

- Adam: Adaptive Moment Estimation
- RMSprop: Root Mean Square Propagation
- Adagrad: Adaptive Gradient Algorithm

Learning rate schedules

Implement learning rate decay:

- Step decay: Reduce by a factor after fixed number of epochs
- Exponential decay: $\alpha_t = \alpha_0 \times e^{-kt}$
- 1/t decay: $\alpha_t = \alpha_0/(1 + kt)$

Learning Rate Impact

Consider training a linear regression model with different learning rates:

- Cost function: $J(\theta) = \frac{1}{2M} \sum_{m=1}^{M} (y^{(m)} - \theta^T x^{(m)})^2$
- Initial parameters: $\theta = [0, 0]^T$
- 100 training examples with true parameters $\theta^* = [2, 3]^T$

With learning rate $\alpha = 0.01$:

- Iteration 1: $J(\theta) = 6.5$
- Iteration 10: $J(\theta) = 2.1$
- Iteration 100: $J(\theta) = 0.2$
- Final parameters: $\theta \approx [1.9, 2.8]^T$ (close to true values)

With learning rate $\alpha = 1.0$:

- Iteration 1: $J(\theta) = 20.3$ (increasing!)
- Iteration 10: $J(\theta) = 156.7$ (diverging)
- Algorithm fails to converge

With learning rate $\alpha = 0.0001$:

- Iteration 1: $J(\theta) = 6.49$
- Iteration 10: $J(\theta) = 6.2$
- Iteration 100: $J(\theta) = 4.8$
- Very slow convergence

Convergence Criteria

Common stopping criteria for gradient descent:

- **Maximum iterations**: Stop after a fixed number of iterations
- **Cost threshold**: Stop when the cost is below a threshold
- **Small gradient**: Stop when the gradient magnitude is below a threshold
- **Parameter change**: Stop when parameters change very little between iterations
- **Validation performance**: Stop when performance on validation set stops improving

Polynomial Regression

Polynomial Models

Polynomial Regression

Polynomial regression extends linear regression to fit non-linear relationships by using polynomial terms. For example:

$$h_\theta(x) = \theta_0 + \theta_1 x + \theta_2 x^2 + \theta_3 x^3 + ...$$

This is accomplished by creating artificial variables $z_1 = x, z_2 = x^2, z_3 = x^3, ...$ and solving as a multivariate linear regression in the transformed space.

Implementing Polynomial Regression

Transform the features

Create new features by applying polynomial transformations to original features:

- For univariate case: Add powers of the feature (x^2, x^3, etc.)
- For multivariate case: Add powers and interaction terms (x_1^2, $x_1 x_2$, etc.)

Apply linear regression

Use standard linear regression methods (normal equation or gradient descent) on the expanded feature set:

- Create design matrix X with transformed features
- Train the model as in linear regression
- Apply regularization to prevent overfitting

Select polynomial degree

Use cross-validation to determine the optimal polynomial degree:

- Try different degrees and evaluate on validation set
- Choose degree that gives the best trade-off between bias and variance

Simple Polynomial Regression Consider fitting a quadratic model to the following data points: (1,1), (2,4), (3,9):

- We use the hypothesis: $h_\theta(x) = \theta_0 + \theta_1 x + \theta_2 x^2$
- Create artificial features: $z_1 = x, z_2 = x^2$
- Apply linear regression in the transformed space
- Result: $\theta_0 = 0, \theta_1 = 0, \theta_2 = 1$
- Final model: $h_\theta(x) = x^2$, which perfectly fits the data

Over- and Underfitting

Overfitting and Underfitting

- **Underfitting**: Model is too simple to capture the underlying pattern in the data, leading to high bias and low variance
- **Overfitting**: Model fits the training data too closely, including noise, leading to low bias but high variance. The model doesn't generalize well to new data

Bias-Variance Tradeoff

The bias-variance tradeoff is central to model selection:

- **Bias**: Error from erroneous assumptions in the model. High bias can cause underfitting.
- **Variance**: Error from sensitivity to small fluctuations in the training set. High variance can cause overfitting.
- **Tradeoff**: Increasing model complexity typically reduces bias but increases variance

The goal is to find the sweet spot that minimizes the total error.

Polynomial Degree Selection

Fitting polynomial models of different degrees to noisy data:

- True function: $f(x) = \sin(x)$ in range $[0, 2\pi]$
- Training data: 20 points with added Gaussian noise

Results with different polynomial degrees:

- Degree 1 (linear): High training error (5.2), high test error (5.1) - underfitting
- Degree 3: Moderate training error (2.1), moderate test error (2.3) - good fit
- Degree 9: Very low training error (0.3), high test error (8.7) - overfitting

The degree 3 polynomial provides the best balance between fitting the training data and generalizing to new data. The degree 9 polynomial follows the noise in the training data rather than the underlying pattern.

Regularization

Regularization

Regularization prevents overfitting by adding a penalty term to the cost function that discourages large parameter values:

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

where λ is the regularization parameter that controls the trade-off between fitting the data and keeping the model simple.

Ridge Regression

Ridge regression (also called L2 regularization) adds a penalty term proportional to the sum of squared coefficients:

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

The solution for ridge regression with the normal equation is:

$$\theta = (X^T X + \lambda I)^{-1} X^T y$$

where I is the identity matrix (with 0 in the position corresponding to the intercept term θ_0).

Effect of Regularization

Regularization has several effects:

- Shrinks coefficients toward zero (but typically not exactly to zero)
- Reduces model variance
- Works well when most features are useful
- Helps with multicollinearity (highly correlated features)

The regularization parameter λ controls the strength of regularization:

- $\lambda = 0$: No regularization (standard linear regression)
- $\lambda \to \infty$: All coefficients approach zero (except θ_0 if not regularized)

Implementing Regularized Polynomial Regression

Feature transformation

Create polynomial features as in standard polynomial regression

Feature scaling

Scale features to have similar ranges, which is important for regularization to work effectively:

- Use standardization: $x' = \frac{x-\mu}{\sigma}$
- Or min-max scaling: $x' = \frac{x-\min(x)}{\max(x)-\min(x)}$

Apply ridge regression

Incorporate regularization into the cost function:

- For normal equation: $\theta = (X^T X + \lambda I)^{-1} X^T y$
- For gradient descent: $\theta_j = \theta_j(1 - \alpha\frac{\lambda}{m}) - \alpha\frac{1}{m}\sum_{i=1}^{m}(h_\theta(x^{(i)}) - y^{(i)})x_j^{(i)}$ for $j \geq 1$

Select hyperparameters

Use cross-validation to select both polynomial degree and regularization parameter:

- Try combinations of degrees and λ values
- Choose combination with best validation performance

Hyperparameter Tuning

Hyperparameter Tuning

Hyperparameters like the regularization parameter λ and polynomial degree must be tuned. Approaches include:

- **Grid Search**: Try all combinations of hyperparameter values
- **Random Search**: Try random combinations from parameter distributions
- **Cross-validation**: Use validation set to evaluate different hyperparameter values

Cross-Validation for Hyperparameter Tuning

Split the data

Divide your data into three parts:

- Training set (e.g., 60%): Used to fit models
- Validation set (e.g., 20%): Used to select hyperparameters
- Test set (e.g., 20%): Used for final evaluation

Define parameter grid

Create a grid of hyperparameter values to explore:

- Polynomial degrees: e.g., [1, 2, 3, 4, 5]
- Regularization parameters: e.g., [0.001, 0.01, 0.1, 1, 10, 100]

Train and evaluate models

For each combination of hyperparameters:

- Train model on training set
- Evaluate on validation set
- Record validation error

Select best hyperparameters

Choose the combination with lowest validation error

Final evaluation

Train a model with the selected hyperparameters on combined training+validation data, and evaluate on test set

Cross-Validation Example Consider tuning a polynomial regression model:

- Dataset: 1000 samples split into 600 training, 200 validation, 200 test
- Grid: Degrees [1, 2, 3, 4] and λ values [0.1, 1, 10]
- Results:
  - Degree=1, λ=0.1: Validation MSE = 5.2
  - Degree=2, λ=1: Validation MSE = 2.1
  - Degree=3, λ=10: Validation MSE = 2.3
  - Degree=4, λ=1: Validation MSE = 2.4
- Best combination: Degree=2, λ=1
- Final test MSE = 2.0

Regularization Path

The regularization path shows how model coefficients change as the regularization parameter varies:

- As λ increases, coefficients generally shrink toward zero
- Some coefficients may shrink faster than others
- The path helps visualize the relative importance of features
- It can help identify a good range for λ

Classification with Logistic Regression

Binary Classification

Logistic Regression

Logistic regression is a supervised learning algorithm for binary classification. It models the probability that an input belongs to the positive class using the logistic function:

$$\hat{y} = h_\theta(x) = g(\theta^T x) = \frac{1}{1 + e^{-\theta^T x}}$$

where $g(z) = \frac{1}{1+e^{-z}}$ is the sigmoid (logistic) function that maps any real number to the range $(0, 1)$.

Decision Boundary

The decision boundary is the line (or hyperplane in higher dimensions) that separates the regions where the model predicts different classes. For logistic regression:

- Predict class 1 if $h_\theta(x) \geq 0.5$
- Predict class 0 if $h_\theta(x) < 0.5$
- This boundary occurs when $\theta^T x = 0$

Cost Function for Logistic Regression

The cost function for logistic regression is the Log Loss (also called Cross-Entropy Loss):

$$J(\theta) = \frac{1}{M} \sum_{m=1}^{M} [-y^{(m)} \log(h_\theta(x^{(m)})) - (1 - y^{(m)}) \log(1 - h_\theta(x^{(m)}))]$$

This function penalizes wrong predictions more heavily as they get more confident.

Training a Logistic Regression Model

Initialize parameters

Start with random values for parameters θ

Gradient descent

Apply gradient descent to minimize the cost function:

$$\theta_j = \theta_j - \alpha \frac{1}{M} \sum_{m=1}^{M} (h_\theta(x^{(m)}) - y^{(m)}) x_j^{(m)}$$

Make predictions

For new data point x:

- Calculate probability $h_\theta(x) = \frac{1}{1+e^{-\theta^T x}}$
- Predict class 1 if $h_\theta(x) \geq 0.5$, otherwise class 0

Evaluate model

Use metrics like accuracy, precision, recall, or F1-score to evaluate model performance on a test set.

Logistic Regression Application Suppose we want to predict whether a student will pass an exam based on hours studied and previous GPA.

- Features: Hours studied (x_1), Previous GPA (x_2)
- Output: Pass (1) or Fail (0)
- After training, we get: $\theta_0 = -6$, $\theta_1 = 0.05$, $\theta_2 = 1.5$
- Decision boundary: $-6 + 0.05 \times \text{hours} + 1.5 \times \text{GPA} = 0$
- For a student who studied 80 hours with GPA 3.2:
  - $z = -6 + 0.05 \times 80 + 1.5 \times 3.2 = -6 + 4 + 4.8 = 2.8$
  - $h_\theta(x) = \frac{1}{1+e^{-2.8}} \approx 0.94$
  - Prediction: Pass (94% confidence)

Differences from Linear Regression

Contrasting Logistic and Linear Regression

While both logistic and linear regression find a linear relationship between features and output, they differ in key ways:

- **Output range**: Linear regression predicts any real number, while logistic regression outputs probabilities between 0 and 1
- **Interpretation**: Linear regression predicts a quantity, while logistic regression predicts a probability
- **Cost function**: Linear regression uses squared error, while logistic regression uses log loss
- **Solution method**: Linear regression has a closed-form solution (normal equation), while logistic regression typically requires iterative optimization

When to Use Logistic vs. Linear Regression

- Use logistic regression when:
  - The target variable is categorical (especially binary)
  - You need probabilistic outputs
  - You want a decision boundary for classification
- Use linear regression when:
  - The target variable is continuous
  - You need to predict a quantity rather than a category
  - A linear relationship is appropriate for the data

Regularization in Logistic Regression

Regularized Logistic Regression

Regularization can be applied to logistic regression to prevent overfitting:

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

The regularization term penalizes large parameter values, encouraging a simpler model.

Implementing Regularized Logistic Regression

Modify the cost function

Include the regularization term in the cost function:

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

Update gradient descent

Modify the update rule for regularized parameters ($j \geq 1$):

$$\theta_j = \theta_j (1 - \alpha \frac{\lambda}{M}) - \alpha \frac{1}{M} \sum_{m=1}^{M} (h_\theta(x^{(m)}) - y^{(m)}) x_j^{(m)}$$

Note: The intercept term θ_0 is typically not regularized.

Select regularization parameter

Use cross-validation to choose an appropriate value for λ.

Multi-class Classification

One-vs-rest Classification

One-vs-rest (also called one-vs-all) is an approach to extend binary classification algorithms like logistic regression to multi-class problems:

- Train K separate binary classifiers, one for each class
- The k-th classifier distinguishes class k from all other classes
- For prediction, apply all classifiers and select the class with highest confidence

Implementing One-vs-rest Classification

Prepare the data

For each class k in $\{1, 2, ..., K\}$, create binary labels:

- $y_k^{(m)} = 1$ if $y^{(m)} = k$ (sample m belongs to class k)
- $y_k^{(m)} = 0$ otherwise (sample m belongs to any other class)

Train multiple classifiers

For each class k:

- Train a binary logistic regression classifier to predict $y_k^{(m)}$
- This gives parameters $\theta^{(k)}$ for class k

Make predictions

For a new data point x:

- Calculate $h_{\theta^{(k)}}(x)$ for each k
- Predict the class with highest probability: $\hat{y} = \text{argmax}_k h_{\theta^{(k)}}(x)$

One-vs-rest Classification

Consider classifying iris flowers into three classes: Setosa, Versicolor, and Virginica.

- Features: Petal length and width
- Three binary classifiers:
  - Classifier 1: Setosa (1) vs. rest (0)
  - Classifier 2: Versicolor (1) vs. rest (0)
  - Classifier 3: Virginica (1) vs. rest (0)

For a new flower with petal length 4.5 cm and width 1.3 cm:

- Classifier 1 probability: 0.02 (Setosa)
- Classifier 2 probability: 0.87 (Versicolor)
- Classifier 3 probability: 0.15 (Virginica)

Since Classifier 2 gives the highest probability, we predict this flower is a Versicolor.

Evaluating Classification Models

Classification Metrics

Common metrics for evaluating binary classification models:

- **Accuracy**: Proportion of correct predictions
- **Precision**: Proportion of positive predictions that are correct
- **Recall**: Proportion of actual positives that are correctly identified
- **F1-score**: Harmonic mean of precision and recall
- **ROC curve**: Plot of true positive rate vs. false positive rate at different thresholds
- **AUC**: Area under the ROC curve

Choosing the Right Evaluation Metric

Consider class balance

- For balanced classes: Accuracy is often sufficient
- For imbalanced classes: Consider precision, recall, F1-score

Consider business context

- False positives more costly: Focus on precision
- False negatives more costly: Focus on recall
- Need to balance both: Use F1-score

Consider probability calibration

- If probabilities are important (not just class predictions): Use log loss or AUC
- For comparing different types of models: AUC is often useful

Use threshold-independent metrics

If the classification threshold might change:

- Use ROC curve to visualize performance across thresholds
- Use AUC to get a single performance number

Confusion Matrix Analysis Consider a medical test for a disease with 1000 test cases:

- True Positives (TP): 150 (correctly identified disease cases)
- False Positives (FP): 50 (incorrectly flagged as disease)
- True Negatives (TN): 700 (correctly identified healthy cases)
- False Negatives (FN): 100 (missed disease cases)

Evaluation metrics:

- Accuracy $= \frac{TP+TN}{TP+FP+TN+FN} = \frac{150+700}{1000} = 0.85$ (85%)
- Precision $= \frac{TP}{TP+FP} = \frac{150}{150+50} = 0.75$ (75%)
- Recall $= \frac{TP}{TP+FN} = \frac{150}{150+100} = 0.60$ (60%)
- F1-score $= \frac{2 \times Precision \times Recall}{Precision+Recall} = \frac{2 \times 0.75 \times 0.60}{0.75+0.60} = 0.67$

Neural Networks

Feed-Forward Neural Networks

Neural Network

A neural network is a computational model inspired by the human brain. The basic building block is the neuron, which computes a weighted sum of its inputs, adds a bias term, and applies an activation function:

$$a = \zeta(w^T x + b)$$

where:

- a is the neuron's output (activation)
- x is the input vector
- w is the weight vector
- b is the bias
- ζ is the activation function

Feed-forward Neural Network

Feed-forward neural networks consist of neurons organized in layers:

- Input layer: Passes input features to the network
- Hidden layers: Process information through weighted connections
- Output layer: Produces the final prediction

Information flows in one direction, from input to output, with no cycles/loops.

Softmax Regression

Softmax regression generalizes logistic regression to multi-class classification:

- For each class, it computes a separate linear combination of the inputs
- It applies the softmax function to get probabilities for each class: $softmax(z)_k = \frac{\exp(z_k)}{\sum_{i=1}^{K} \exp(z_i)}$
- The predicted class is the one with the highest probability

Activation Functions

Common activation functions include:

- Sigmoid: $\zeta(z) = \frac{1}{1+e^{-z}}$, outputs in range (0,1)
- Tanh: $\zeta(z) = \frac{e^z - e^{-z}}{e^z + e^{-z}}$, outputs in range (-1,1)
- ReLU: $\zeta(z) = \max(0, z)$, simple and computationally efficient
- Softmax: $\zeta(z_i) = \frac{e^{z_i}}{\sum_j e^{z_j}}$, used for multi-class classification

Universality Theorem

The universality theorem (Hornik, 1991) states that a neural network with a single hidden layer using a nonlinear activation function can approximate any continuous function to any desired level of accuracy, given enough hidden units.

This means neural networks are very flexible and can learn complex relationships in data, though deeper networks with multiple hidden layers often learn more efficiently.

Training Neural Networks

Cost Functions for Neural Networks

Common cost functions include:

- For regression: Mean Squared Error (MSE)
- For binary classification: Binary Cross-Entropy
- For multi-class classification: Categorical Cross-Entropy

Backpropagation

Backpropagation is the algorithm used to train neural networks by:

1. Performing a forward pass to compute predictions
2. Computing the error/loss
3. Propagating the error backwards through the network to compute gradients
4. Updating weights and biases using gradient descent

This process leverages the chain rule of calculus to efficiently compute gradients.

Training a Neural Network

Initialize the network

- Define network architecture (number of layers, neurons per layer)
- Initialize weights and biases randomly
- Select activation functions, cost function, and optimizer

Forward propagation

For each training sample:

- Input features to the first layer
- Compute activations through each layer
- Obtain prediction from output layer

Compute loss

Calculate the difference between predictions and actual values using the cost function

Backpropagation

- Compute gradients of the cost function with respect to weights and biases
- Use chain rule to propagate gradients backward through the network

Update weights

- Use gradient descent or an advanced optimizer (Adam, RMSprop)
- Update weights and biases: $w = w - \alpha \frac{\partial L}{\partial w}$

Repeat

Iterate steps 2-5 until convergence or for a fixed number of epochs

Backpropagation Example

Consider a simple neural network with one input, one hidden layer with two neurons, and one output neuron.

- Input: $x = 0.5$
- Hidden layer weights: $w_{11}^{(1)} = 0.15, w_{21}^{(1)} = 0.25$
- Hidden layer biases: $b_1^{(1)} = 0.35, b_2^{(1)} = 0.45$
- Output layer weights: $w_{11}^{(2)} = 0.40, w_{12}^{(2)} = 0.50$
- Output layer bias: $b_1^{(2)} = 0.60$
- Activation function: Sigmoid
- True output: $y = 0.75$

Forward pass:

$$
\begin{aligned}
z_1^{(1)} &= w_{11}^{(1)} \cdot x + b_1^{(1)} = 0.15 \cdot 0.5 + 0.35 = 0.425 \\
a_1^{(1)} &= \sigma(z_1^{(1)}) = \sigma(0.425) = 0.605 \\
z_2^{(1)} &= w_{21}^{(1)} \cdot x + b_2^{(1)} = 0.25 \cdot 0.5 + 0.45 = 0.575 \\
a_2^{(1)} &= \sigma(z_2^{(1)}) = \sigma(0.575) = 0.640 \\
z_1^{(2)} &= w_{11}^{(2)} \cdot a_1^{(1)} + w_{12}^{(2)} \cdot a_2^{(1)} + b_1^{(2)} \\
&= 0.40 \cdot 0.605 + 0.50 \cdot 0.640 + 0.60 = 0.842 + 0.60 = 1.442 \\
\hat{y} = a_1^{(2)} &= \sigma(z_1^{(2)}) = \sigma(1.442) = 0.809
\end{aligned}
$$

Backpropagation:

$$
\begin{aligned}
\frac{\partial L}{\partial a_1^{(2)}} &= 2(a_1^{(2)} - y) = 2(0.809 - 0.75) = 0.118 \\
\frac{\partial a_1^{(2)}}{\partial z_1^{(2)}} &= a_1^{(2)}(1 - a_1^{(2)}) = 0.809 \cdot 0.191 = 0.154 \\
\frac{\partial L}{\partial z_1^{(2)}} &= \frac{\partial L}{\partial a_1^{(2)}} \cdot \frac{\partial a_1^{(2)}}{\partial z_1^{(2)}} = 0.118 \cdot 0.154 = 0.018
\end{aligned}
$$

Continuing with more gradients and weight updates...

Neural Network Architectures

Common Neural Network Architectures

- **Fully Connected (Dense) Network**: Each neuron is connected to all neurons in adjacent layers
- **Convolutional Neural Network (CNN)**: Uses convolutional layers to process grid-like data (e.g., images)
- **Recurrent Neural Network (RNN)**: Contains loops to persist information, suitable for sequential data
- **Long Short-Term Memory (LSTM)**: A special type of RNN that can learn long-term dependencies
- **Transformer**: Uses attention mechanisms, effective for sequential data like text

Preventing Overfitting

Dealing with Overfitting in Neural Networks

Techniques to prevent overfitting include:

- **Dropout**: Randomly deactivate neurons during training
- **Early Stopping**: Stop training when performance on validation set starts to degrade
- **Data Augmentation**: Create new training samples by applying transformations
- **Regularization**: Add penalty terms for large weights

Dropout

Dropout is a regularization technique that prevents overfitting in neural networks:

- During training, randomly deactivate a fraction of neurons in each layer
- Each forward and backward pass uses a different randomly dropped network
- Forces the network to learn redundant representations
- During inference, all neurons are used (with scaled activations)

Early Stopping

Early stopping prevents overfitting by monitoring performance on a validation set:

- Train the model and evaluate on validation set periodically
- Stop training when validation error starts to increase
- Use the model parameters from the point with best validation performance

Data Augmentation

Data augmentation increases the effective size of the training dataset:

- For images: rotations, flips, crops, color adjustments
- For text: synonym replacement, back-translation
- For audio: time stretching, pitch shifting, masking

Advanced Topics

TODO: Add advanced topics like transfer learning, generative adversarial networks (GANs), and reinforcement learning.

Neural Networks

Neural networks are a powerful class of models inspired by the human brain, capable of learning complex patterns from data. This section covers their structure, training methods, and advanced applications.

Feed-Forward Neural Networks

Feed-forward neural networks are the foundation of deep learning, consisting of layers of interconnected neurons that process information in one direction.

Neural Network

A neural network is a computational model inspired by the human brain. The basic building block is the neuron, which computes a weighted sum of its inputs, adds a bias term, and applies an activation function:

$$a = \zeta(w^T x + b)$$

where:

- a is the neuron's output (activation)
- x is the input vector
- w is the weight vector
- b is the bias
- ζ is the activation function

Feed-forward Neural Network

Feed-forward neural networks consist of neurons organized in layers:

- Input layer: Passes input features to the network
- Hidden layers: Process information through weighted connections
- Output layer: Produces the final prediction

Information flows in one direction, from input to output, with no cycles/loops.

Softmax Regression

Softmax regression generalizes logistic regression to multi-class classification:

- For each class, it computes a separate linear combination of the inputs
- It applies the softmax function to get probabilities for each class: $softmax(z)_k = \frac{\exp(z_k)}{\sum_{i=1}^{K} \exp(z_i)}$
- The predicted class is the one with the highest probability

Activation Functions

Common activation functions include:

- Sigmoid: $\zeta(z) = \frac{1}{1+e^{-z}}$, outputs in range (0,1)
- Tanh: $\zeta(z) = \frac{e^z - e^{-z}}{e^z + e^{-z}}$, outputs in range (-1,1)
- ReLU: $\zeta(z) = \max(0, z)$, simple and computationally efficient
- Softmax: $\zeta(z_i) = \frac{e^{z_i}}{\sum_j e^{z_j}}$, used for multi-class classification

Universality Theorem

The universality theorem (Hornik, 1991) states that a neural network with a single hidden layer using a nonlinear activation function can approximate any continuous function to any desired level of accuracy, given enough hidden units.

This means neural networks are very flexible and can learn complex relationships in data, though deeper networks with multiple hidden layers often learn more efficiently.

Training Neural Networks

Training neural networks involves finding optimal weights and biases that minimize the difference between predicted and actual outputs.

Cost Functions for Neural Networks

Common cost functions include:

- For regression: Mean Squared Error (MSE)
- For binary classification: Binary Cross-Entropy
- For multi-class classification: Categorical Cross-Entropy

Backpropagation

Backpropagation is the algorithm used to train neural networks by:

1. Performing a forward pass to compute predictions
2. Computing the error/loss
3. Propagating the error backwards through the network to compute gradients
4. Updating weights and biases using gradient descent

This process leverages the chain rule of calculus to efficiently compute gradients.

Training a Neural Network

Initialize the network

- Define network architecture (number of layers, neurons per layer)
- Initialize weights and biases randomly
- Select activation functions, cost function, and optimizer

Forward propagation

For each training sample:

- Input features to the first layer
- Compute activations through each layer
- Obtain prediction from output layer

Compute loss

Calculate the difference between predictions and actual values using the cost function

Backpropagation

- Compute gradients of the cost function with respect to weights and biases
- Use chain rule to propagate gradients backward through the network

Update weights

- Use gradient descent or an advanced optimizer (Adam, RMSprop)
- Update weights and biases: $w = w - \alpha \frac{\partial L}{\partial w}$

Repeat

Iterate steps 2-5 until convergence or for a fixed number of epochs

Backpropagation Example

Consider a simple neural network with one input, one hidden layer with two neurons, and one output neuron.

- Input: $x = 0.5$
- Hidden layer weights: $w_{11}^{(1)} = 0.15, w_{21}^{(1)} = 0.25$
- Hidden layer biases: $b_1^{(1)} = 0.35, b_2^{(1)} = 0.45$
- Output layer weights: $w_{11}^{(2)} = 0.40, w_{12}^{(2)} = 0.50$
- Output layer bias: $b_1^{(2)} = 0.60$
- Activation function: Sigmoid
- True output: $y = 0.75$

Forward pass:

$$
\begin{aligned}
z_1^{(1)} &= w_{11}^{(1)} \cdot x + b_1^{(1)} = 0.15 \cdot 0.5 + 0.35 = 0.425 \\
a_1^{(1)} &= \sigma(z_1^{(1)}) = \sigma(0.425) = 0.605 \\
z_2^{(1)} &= w_{21}^{(1)} \cdot x + b_2^{(1)} = 0.25 \cdot 0.5 + 0.45 = 0.575 \\
a_2^{(1)} &= \sigma(z_2^{(1)}) = \sigma(0.575) = 0.640 \\
z_1^{(2)} &= w_{11}^{(2)} \cdot a_1^{(1)} + w_{12}^{(2)} \cdot a_2^{(1)} + b_1^{(2)} \\
&= 0.40 \cdot 0.605 + 0.50 \cdot 0.640 + 0.60 = 0.842 + 0.60 = 1.442 \\
\hat{y} = a_1^{(2)} &= \sigma(z_1^{(2)}) = \sigma(1.442) = 0.809
\end{aligned}
$$

Backpropagation:

$$
\begin{aligned}
\frac{\partial L}{\partial a_1^{(2)}} &= 2(a_1^{(2)} - y) = 2(0.809 - 0.75) = 0.118 \\
\frac{\partial a_1^{(2)}}{\partial z_1^{(2)}} &= a_1^{(2)}(1 - a_1^{(2)}) = 0.809 \cdot 0.191 = 0.154 \\
\frac{\partial L}{\partial z_1^{(2)}} &= \frac{\partial L}{\partial a_1^{(2)}} \cdot \frac{\partial a_1^{(2)}}{\partial z_1^{(2)}} = 0.118 \cdot 0.154 = 0.018
\end{aligned}
$$

This gradient is then used to update the weights in the network.

Neural Network Architectures

Neural networks come in various architectures, each designed for specific types of data or tasks.

Common Neural Network Architectures

- **Fully Connected (Dense) Network**: Each neuron is connected to all neurons in adjacent layers
- **Convolutional Neural Network (CNN)**: Uses convolutional layers to process grid-like data (e.g., images)
- **Recurrent Neural Network (RNN)**: Contains loops to persist information, suitable for sequential data
- **Long Short-Term Memory (LSTM)**: A special type of RNN that can learn long-term dependencies
- **Transformer**: Uses attention mechanisms, effective for sequential data like text

Convolutional Neural Networks

CNNs are specialized for processing grid-like data such as images:

- **Convolutional layers**: Apply filters to detect local patterns
- **Pooling layers**: Reduce spatial dimensions while retaining important features
- **Parameter sharing**: Same filter applied across the entire input
- **Local connectivity**: Each neuron connects to a small region of the input

These properties make CNNs highly effective for tasks like image classification, object detection, and image segmentation.

Recurrent Neural Networks

RNNs are designed for sequential data:

- Each neuron receives input from the current sample and its own previous output
- This creates a form of memory that captures dependencies across time steps
- Useful for tasks like time series prediction, speech recognition, and language modeling
- Variants like LSTM and GRU address the vanishing gradient problem in standard RNNs

Preventing Overfitting

Overfitting occurs when a model learns the training data too well, including its noise, and fails to generalize to new data. Several techniques can prevent this issue in neural networks.

Dealing with Overfitting in Neural Networks

Techniques to prevent overfitting include:

- **Dropout**: Randomly deactivate neurons during training
- **Early Stopping**: Stop training when performance on validation set starts to degrade
- **Data Augmentation**: Create new training samples by applying transformations
- **Regularization**: Add penalty terms for large weights

Dropout

Dropout is a regularization technique that prevents overfitting in neural networks:

- During training, randomly deactivate a fraction of neurons in each layer
- Each forward and backward pass uses a different randomly dropped network
- Forces the network to learn redundant representations
- During inference, all neurons are used (with scaled activations)

Early Stopping

Early stopping prevents overfitting by monitoring performance on a validation set:

- Train the model and evaluate on validation set periodically
- Stop training when validation error starts to increase
- Use the model parameters from the point with best validation performance

Data Augmentation

Data augmentation increases the effective size of the training dataset:

- For images: rotations, flips, crops, color adjustments
- For text: synonym replacement, back-translation
- For audio: time stretching, pitch shifting, masking

Implementing Dropout

During training

- For each training batch, generate a binary mask for each layer with dropout
- Set a dropout rate (typically 0.2-0.5) representing the fraction of neurons to drop
- Multiply activations by the mask to zero out dropped neurons
- Scale remaining activations by $\frac{1}{1-\text{dropout\_rate}}$ to maintain same expected value

During testing/inference

- Use all neurons without dropout
- No scaling needed if scaling was done during training

Implementation considerations

- Apply dropout after activation functions
- Use different dropout rates for different layers (typically higher for layers with more neurons)
- Don't apply dropout to the output layer

Vanishing and Exploding Gradient Problems

In deep networks, gradients can become very small (vanishing) or very large (exploding) as they propagate through the network:

- **Vanishing gradients**: Parameters in early layers change very slowly or not at all
- **Exploding gradients**: Parameters in early layers change dramatically, causing instability
- **Causes**: Repeated multiplication of values less than 1 (vanishing) or greater than 1 (exploding)

Solutions include:

- Better activation functions (ReLU, Leaky ReLU)
- Weight initialization techniques (Xavier, He initialization)
- Batch normalization
- Residual connections (skip connections)

Hyperparameter Tuning

Hyperparameters are settings that control the training process and model architecture. Choosing optimal values is crucial for model performance.

Choosing Hyperparameters for Neural Networks

Architecture design

- Number of layers: Start small and increase gradually
- Neurons per layer: Powers of 2 (64, 128, 256) work well; larger for complex tasks
- Activation functions: ReLU for hidden layers, softmax for multi-class output

Training parameters

- Learning rate: Start with 0.01 or 0.001 and tune using learning rate finder
- Batch size: 32, 64, or 128 for most tasks; larger if memory allows
- Optimizer: Adam is a good default choice
- Epochs: Use early stopping with patience

Regularization

- Dropout rate: 0.2-0.5 depending on layer size
- Weight decay (L2 regularization): 1e-4 is a reasonable starting point
- Early stopping patience: 10-20 epochs

Advanced techniques

- Learning rate schedules: Reduce on plateau or cosine annealing
- Batch normalization: Apply before activation functions
- Skip connections: Use for networks deeper than 10 layers

Cross-Validation for Neural Networks

Cross-validation helps find optimal hyperparameters:

- Split data into training, validation, and test sets
- Train models with different hyperparameter combinations
- Evaluate each model on validation set
- Select hyperparameters that yield best validation performance
- Final evaluation on test set

Unlike traditional k-fold cross-validation, neural networks often use a single validation split due to computational constraints.

Digit Recognition with Neural Networks Consider the MNIST dataset of handwritten digits:

- Input: 28x28 pixel grayscale images (784 features)
- Output: 10 classes (digits 0-9)
- Network architecture:
  - Input layer: 784 neurons
  - Hidden layer 1: 128 neurons with ReLU activation
  - Hidden layer 2: 64 neurons with ReLU activation
  - Output layer: 10 neurons with softmax activation
- Training:
  - Loss function: Categorical cross-entropy
  - Optimizer: Adam with learning rate 0.001
  - Batch size: 64
  - Epochs: 20 with early stopping
- Results:
  - Training accuracy: 99.2%
  - Validation accuracy: 98.1%

Advanced Topics

Neural networks continue to evolve with advanced techniques that improve performance, efficiency, and applicability to various domains.

Transfer Learning

Transfer learning leverages knowledge from pre-trained models:

- Start with a model trained on a large dataset
- Remove the final layer(s) and replace with new layers
- Either freeze pre-trained layers or fine-tune them with a small learning rate
- Requires much less data than training from scratch
- Common in computer vision (ResNet, VGG) and NLP (BERT, GPT)

Batch Normalization

Batch normalization normalizes the inputs of each layer:

- Normalizes activations to have mean 0 and variance 1
- Reduces internal covariate shift
- Allows higher learning rates
- Acts as a regularizer
- Applied before or after activation functions

Advanced Optimizers

Beyond standard gradient descent, advanced optimizers improve training:

- **Momentum**: Adds a fraction of previous update to current update
- **RMSprop**: Adapts learning rates based on recent gradients
- **Adam**: Combines momentum and RMSprop
- **AdamW**: Adam with decoupled weight decay

Implementing Transfer Learning

Select pre-trained model

- Choose model trained on a relevant domain
- Consider model size, accuracy, and compatibility
- Common options: ResNet, VGG, EfficientNet for images; BERT, GPT for text

Prepare the model

- Load pre-trained weights
- Remove the output layer (and possibly some layers before it)
- Add new layers specific to your task

Training approach

- Feature extraction: Freeze pre-trained layers, train only new layers
- Fine-tuning: Train the entire network with a small learning rate
- Progressive fine-tuning: Gradually unfreeze more layers

Adapt to your data

- Ensure input preprocessing matches pre-trained model
- Consider domain adaptation techniques if domains differ significantly
- Use appropriate regularization for new layers

Practical Neural Network Project

Consider an image classification task to identify plant diseases from leaf images:

- Dataset: 5,000 images across 10 disease categories
- Limited computational resources

Solution approach using transfer learning:

- Base model: Pre-trained ResNet50 (trained on ImageNet)
- Remove top classification layer
- Add custom layers:
  - Global average pooling
  - Dropout layer (rate=0.3)
  - Dense layer with 256 neurons and ReLU activation
  - Final layer with 10 neurons and softmax activation
- Training strategy:
  - First stage: Freeze ResNet layers, train only custom layers (5 epochs)
  - Second stage: Unfreeze last 30 ResNet layers, train entire model with small learning rate (0.0001) for 15 epochs
- Data augmentation:
  - Random rotations ($\pm 20°$)
  - Horizontal and vertical flips
  - Small zoom and brightness variations
- Results: 96.5% accuracy on test set, compared to 85% when training from scratch

Model Interpretability

Techniques to understand neural network decisions:

- **Feature importance**: Quantifying the impact of each input feature
- **Activation visualization**: Examining neuron activations for different inputs
- **Saliency maps**: Highlighting regions of input that most influence the output
- **LIME (Local Interpretable Model-agnostic Explanations)**: Approximating the model locally with an interpretable model
- **SHAP (SHapley Additive exPlanations)**: Assigning each feature an importance value

Interpretability helps build trust, debug models, and identify potential biases.

Debugging Neural Networks

Diagnose learning problems

- Monitor training and validation losses
- If validation loss increases while training loss decreases: Overfitting
- If both losses plateau at high values: Underfitting
- If both losses are unstable: Learning rate may be too high

Check gradients

- Inspect gradient magnitudes; very large or very small values indicate problems
- Test with gradient clipping to prevent exploding gradients
- Verify with numerical gradient checking in critical cases

Analyze model predictions

- Examine confusion matrix to identify patterns in misclassifications
- Look at the most confidently wrong predictions
- Test with simpler inputs to verify basic functionality

Simplify and rebuild

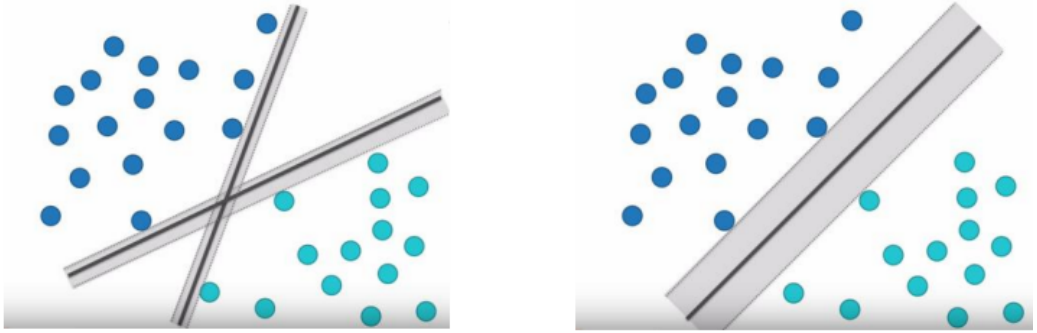
- Start with a minimal working model
- Add complexity incrementally
- Establish baseline performance with simple models
- Verify each component individually

Support Vector Machines

Hyperplanes and Margin Classification

Support Vector Machine (SVM)

Goal: Find the optimal hyperplane that separates data points of different classes in a high-dimensional space. (maximize the margin between classes)



Support Vector Machine (SVM)

Support Vector Machines are supervised learning models used for classification and regression. In classification, SVMs find the hyperplane that best separates classes with the maximum margin.

Hyperplane

In N-dimensional space, a hyperplane is a flat affine subspace of dimensions $N-1$. In two dimensions, a hyperplane is a line defined by:

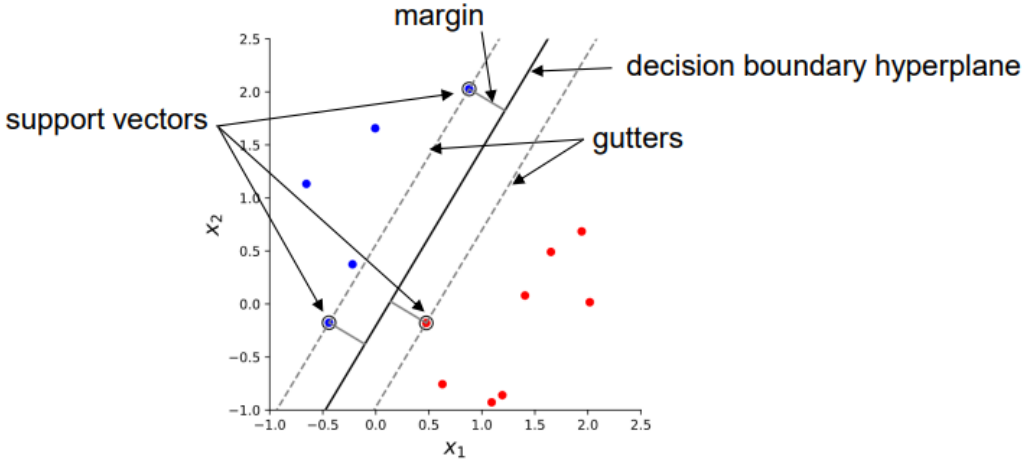
$$b + w_1 x_1 + w_2 x_2 = 0$$

In the general N-dimensional case:

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

Support Vectors

Support vectors are the data points that are closest to the hyperplane and influence its position. They are critical for defining the optimal hyperplane.



Optimisation Task for the Maximal Margin Classifier

The optimization task for the maximal margin classifier is to minimize the following objective function:

$$\min_{\mathbf{w},b} \frac{1}{2}||\mathbf{w}||^2 \quad \text{'Primal Form'}$$

subject to the constraints:

$$y^{(m)}(b + \mathbf{w}^T \mathbf{x}^{(m)}) \geq 1, \quad \text{for } m = 1, ..., M$$

where $\mathbf{w}$ is the weight vector, b is the bias term, and $y^{(m)}$ is the class label of the m-th data point.

Maximal Margin Classifier Primal Form

A maximal margin classifier finds the hyperplane that maximizes the distance to the nearest data point from any class. This leads to the optimization problem:

$$\min_{b,w} \frac{1}{2}||w||^2$$

subject to: $y^{(m)}(b + w^T x^{(m)}) \geq 1 \quad \forall m = 1, ..., M$

The data points closest to the hyperplane are called support vectors.

Maximal Margin Classifier Dual Form

The optimization problem for SVMs can be reformulated in its dual form:

$$\max_{\alpha} \mathcal{L}(\alpha) = -\frac{1}{2} \sum_{i=1}^{M} \sum_{j=1}^{M} \alpha_i \alpha_j y^{(i)} y^{(j)} x^{(i)} x^{(j)} + \sum_{m=1}^{M} \alpha_m$$

subject to: $\sum_{m=1}^{M} \alpha_m y^{(m)} = 0$ and $0 \leq \alpha_m \leq C$ for $m = 1, ..., M$

This formulation allows the application of the kernel trick by replacing the dot product $x^{(i)} x^{(j)}$ with a kernel function $\mathcal{K}(x^{(i)}, x^{(j)})$.

Predictions from the Optimised Lagrangian

With the optimal $\hat{\alpha}_m$ we can compute the optimal $\widehat{\mathbf{w}}$ from the equation in the previous slide

$$\widehat{\mathbf{w}} = \sum_{m=1}^{M} \hat{\alpha}_m y^{(m)} \mathbf{x}^{(m)}$$

$$\text{and } \hat{b} = \frac{1}{M_s} \sum_{m=1}^{M_s} \left(y^{(m)} - \widehat{\mathbf{w}}^T \mathbf{x}^{(m)} \right) \text{ for the support vectors}$$

where M_s is the number of support vectors and $\hat{\alpha}_m > 0$

Then, a new sample $\mathbf{x}^{(*)}$ can be classified using $y^{(*)} = \text{sign}(\widehat{\mathbf{w}}^T \mathbf{x}^{(*)} + \hat{b})$ The distance from the hyperplane can be interpreted as a confidence for the prediction.

Kernel Trick and Soft Margin

Hard Margin vs. Soft Margin

- **Hard Margin Classifier**: Requires all points to be correctly classified and outside the margin
- **Soft Margin Classifier**: Allows some points to violate the margin or be misclassified

The hard margin classifier works only when data is linearly separable and is sensitive to outliers.

Soft Margin Classifier

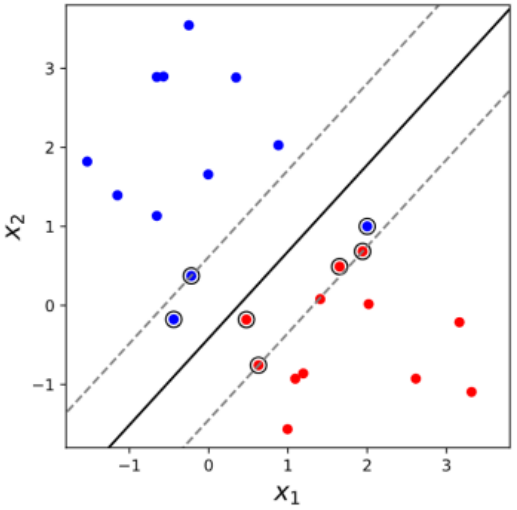
Cost Function in Soft-Margin SVM

The soft margin classifier allows some data points to violate the margin constraints, adding **slack variables** $\varepsilon_m \geq 0$:

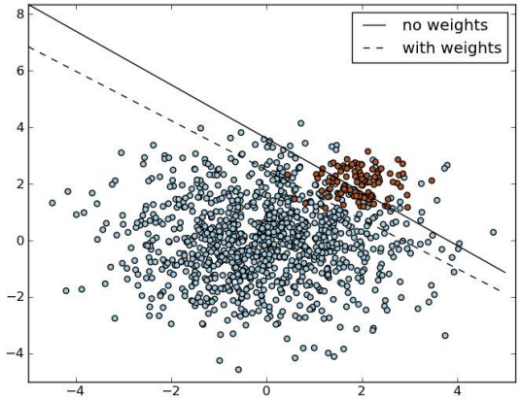
$$\min_{b,w,\varepsilon} \frac{1}{2}||w||^2 + C\sum_{m=1}^{M} \varepsilon^{(m)}$$

subject to: $y^{(m)}(b + \mathbf{w}^T\mathbf{x}^{(m)}) \geq 1 - \varepsilon^{(m)}$ and $\varepsilon^{(m)} \geq 0$ for $m = 1, ..., M$

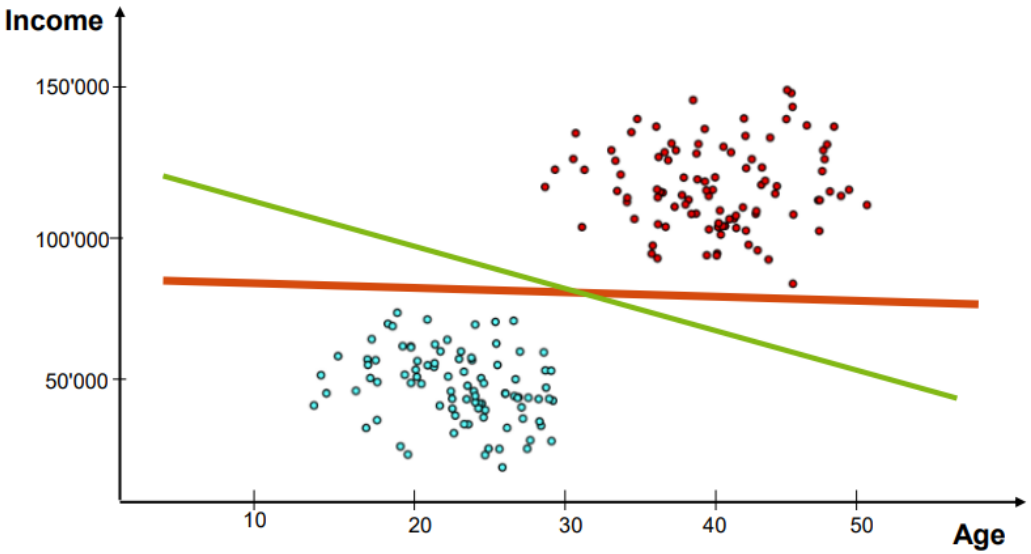
The parameter C controls the trade-off between maximizing the margin and minimizing constraint violations.



Unbalanced data:

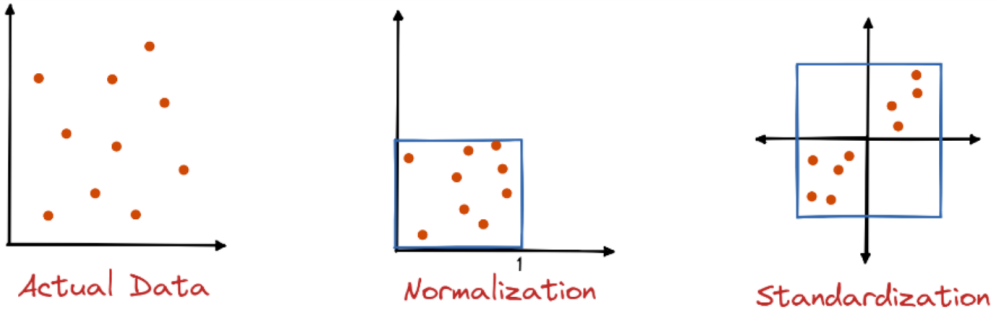


'Wrong' Hyperplane



The **RED** line is the optimal hyperplane for the hard margin classifier. The **GREEN** line is NOT optimal!

Standardization and Normalization



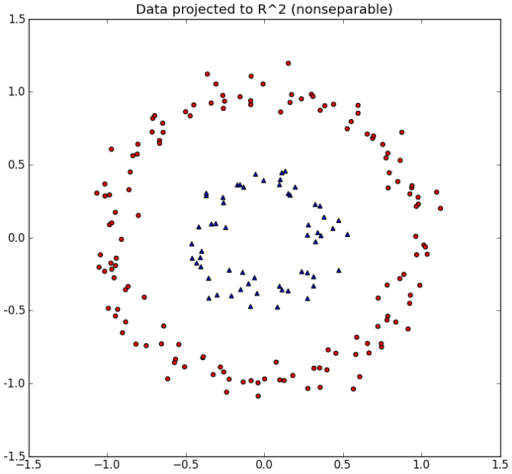
$$x_{\text{norm}} = \frac{x - \min(x)}{\max(x) - \min(x)}$$

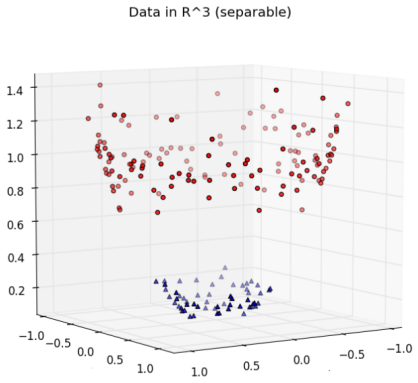
$$x_{\text{stand}} = \frac{x - \text{mean}(x)}{\text{standard deviation }(x)}$$

Normalization: e.g. for Neural Networks
Standardization: e.g. for SVMs, Linear and Logistic Regression

Kernel Trick for non-linearly seperable data

Lifting Trick Projection to Higher Dimensions





Transformation: $(x_1, x_2) \rightarrow (x_1, x_2, x_1^2 + x_2^2)$

The lifting trick allows us to transform the data into a higher-dimensional space where it may become linearly separable.

Kernel Trick

The kernel trick allows SVMs to efficiently operate in high-dimensional spaces without explicitly computing the transformation. Common kernels include:

- Linear: $\mathcal{K}(x^{(m)}, x^{(m')}) = \sum_{n=1}^{N} x_n^{(m)} x_n^{(m')}$
- Polynomial: $\mathcal{K}(x^{(m)}, x^{(m')}) = (1 + \sum_{n=1}^{N} x_n^{(m)} x_n^{(m')})^d$
- RBF: $\mathcal{K}(x^{(m)}, x^{(m')}) = \exp(-\gamma ||x^{(m)} - x^{(m')}||^2)$

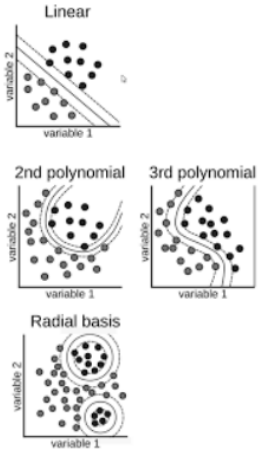
Maximal Margin Classifier Dual Form with Kernel Trick

$$\max_{\alpha} \mathcal{L}(\alpha) = -\frac{1}{2} \sum_{i=1}^{M} \sum_{j=1}^{M} \alpha_i \alpha_j y^{(i)} y^{(j)} \mathcal{K}(x^{(i)}, x^{(j)}) + \sum_{m=1}^{M} \alpha_m$$

subject to: $\sum_{m=1}^{M} \alpha_m y^{(m)} = 0$ and $0 \leq \alpha_m \leq C$ for $m = 1, ..., M$

This formulation allows the application of the kernel trick by replacing the dot product $x^{(i)} x^{(j)}$ with a kernel function $\mathcal{K}(x^{(i)}, x^{(j)})$.

Common Kernel Functions



Linear: $K(a, b) = a^\top \cdot b$

Polynomial: $K(a, b) = \left(a^\top \cdot b + t\right)^r$ with parameters $\mathbf{t}, \mathbf{r}$

RBF = Radial Basis Function:

$$\mathrm{K(a,\ b)} = e^{-\gamma \|a-b\|^2}$$

with parameter γ

Selecting the Right Kernel and Parameters

When to use different kernels

- Linear kernel:
  - When data is linearly separable
  - When feature space is large compared to sample size
  - For text classification problems
- Polynomial kernel:
  - When decision boundary is a curved line or surface
  - For image processing problems
  - Typical degrees: 2 or 3
- RBF kernel:
  - When classes form complex shapes
  - For most general-purpose classification
  - When you have sufficient data

Parameter tuning

- Use grid search or random search with cross-validation
- C parameter search range: $[0.1, 1, 10, 100, 1000]$
- For RBF kernel, γ search range: $[0.01, 0.1, 1, 10]$
- For polynomial kernel, degree values: $[2, 3, 4]$

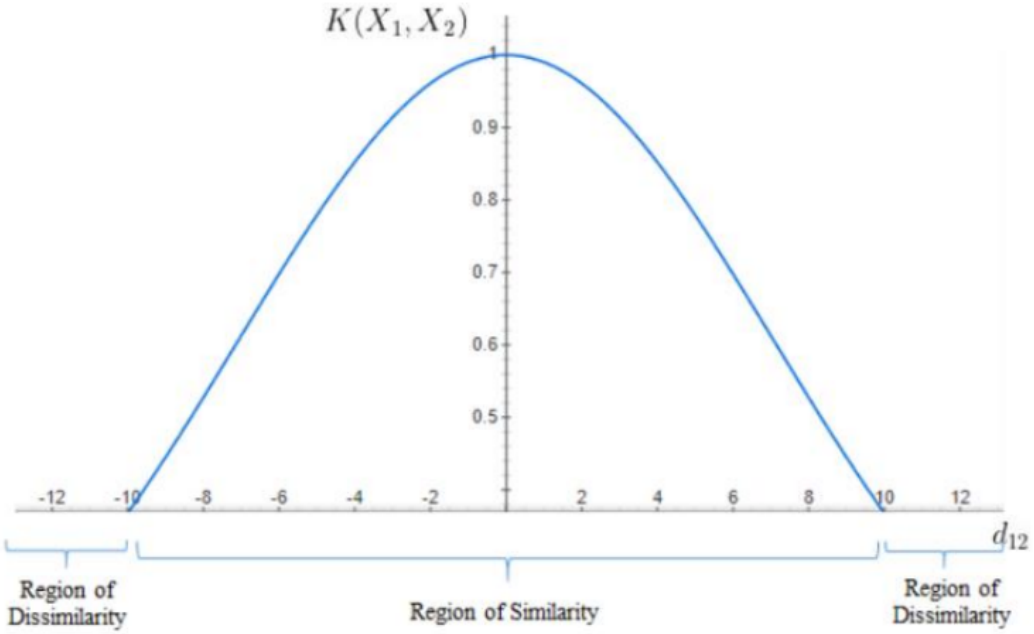
Optimize for the metric that matters

- Accuracy for balanced problems
- F1-score for imbalanced problems
- Precision or recall when false positives or false negatives are more critical

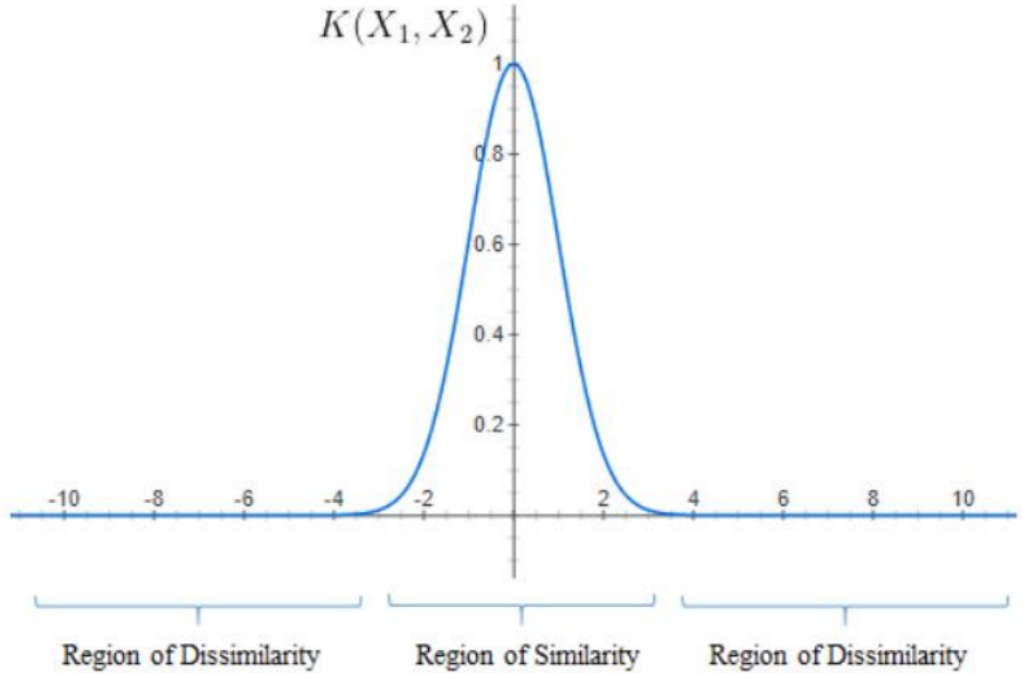
RBF Kernel

$$\mathrm{K(a,\ b)} = e^{-\gamma\|a-b\|^2} = e^{-\frac{||a-b||^2}{2\sigma^2}}$$

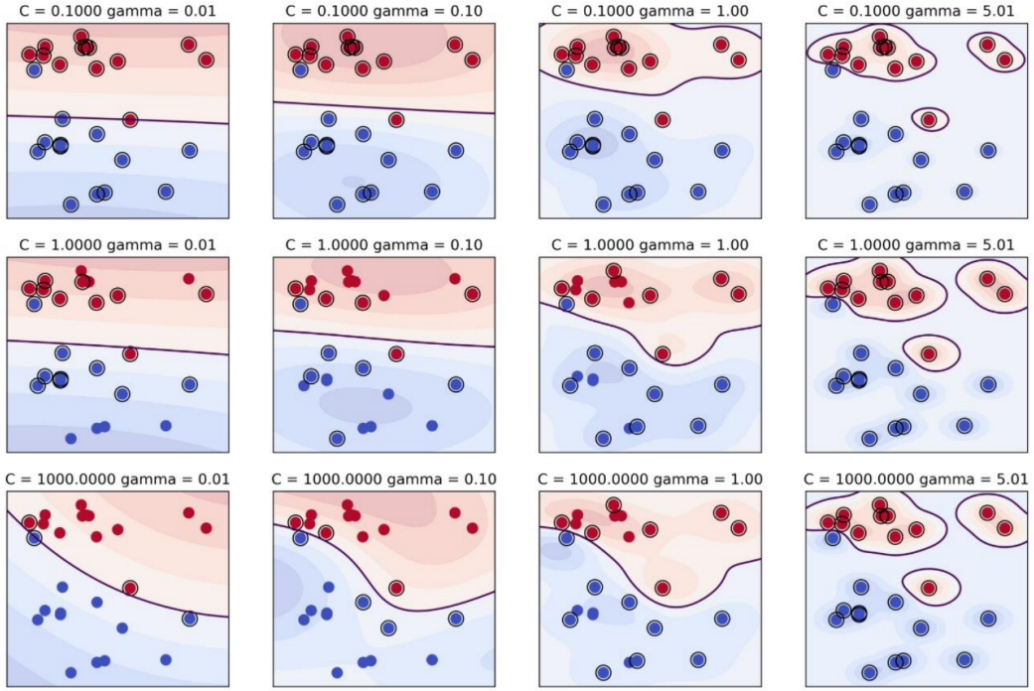
Small $\gamma \rightarrow$ large width σ (Gaussian kernel)



Large $\gamma \rightarrow$ small width σ (Gaussian kernel)



RBF Kernel



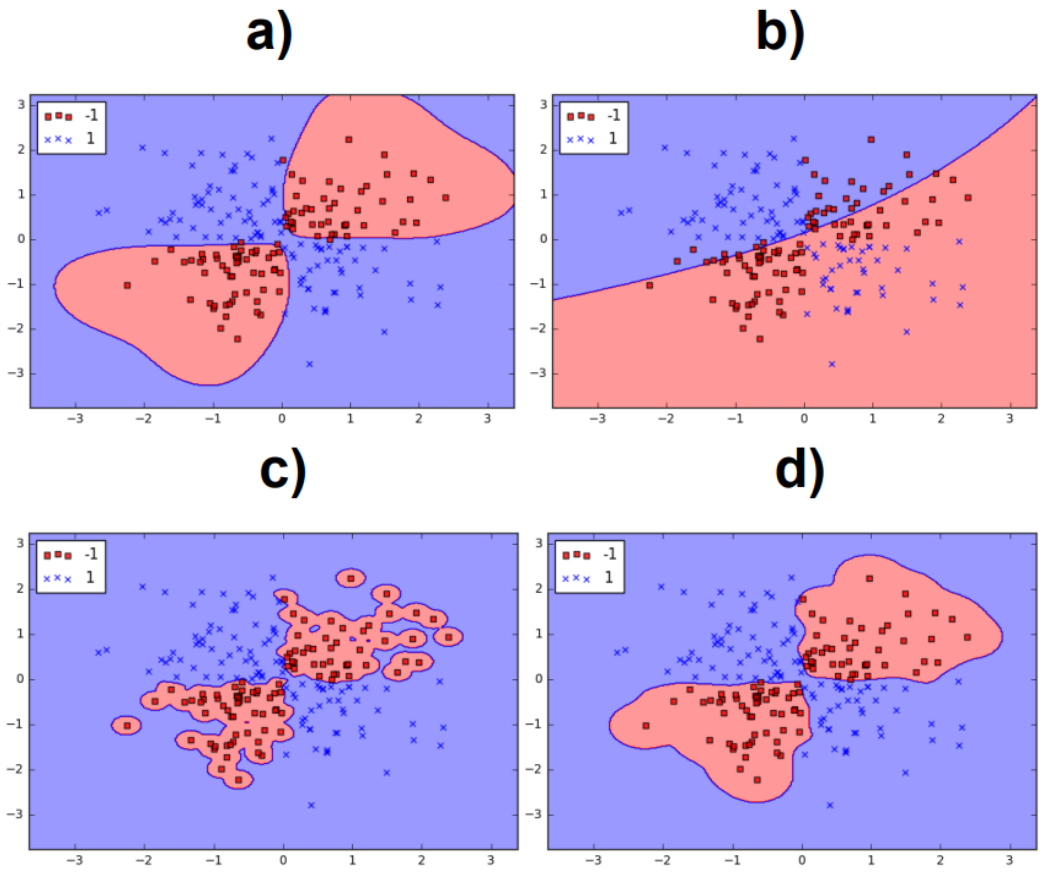
Interdependence of SVM Hyperparameters

C and γ both influence the shape of the decision boundary and need to be tuned together using hyperparameter optimization (e.g. via cross validation)

ConceptTest: RBF Kernel

In the SVM below we are using $C = 1$. Which image has the highest value for hyperparameter γ ?

$$K\left(\mathbf{x}, \mathbf{x}^{(m)}\right) = \exp\left(-\gamma \left\|\mathbf{x}^{(m)} - \mathbf{x}\right\|^2\right)$$



SVM Application Linear Kernel

Consider classifying flowers based on petal length and width:

- Features: Petal length (x_1) and petal width (x_2)
- Classes: Setosa (-1) and Versicolor (+1)
- After training with a linear kernel, we get $b = -0.5$ and $w = [0.2, 0.8]$
- For a new flower with petal length = 4 cm and width = 1.3 cm:
  - $f(x) = -0.5 + 0.2 \times 4 + 0.8 \times 1.3 = -0.5 + 0.8 + 1.04 = 1.34 > 0$
  - Prediction: Versicolor (class +1)

Training a Support Vector Machine

Select kernel

Choose an appropriate kernel function:

- Linear kernel for linearly separable data
- Polynomial kernel for more complex, non-linear boundaries
- RBF (Gaussian) kernel for highly non-linear data

Select hyperparameters

- C: Regularization parameter (controls trade-off between margin width and misclassification)
- Kernel parameters: degree d for polynomial kernel, γ for RBF kernel

Train the model

- Set up the optimization problem to maximize the margin
- Solve the quadratic programming problem to find support vectors

Make predictions

For a new data point x:

- Compute $f(x) = b + \sum_{i \in SV} \alpha_i y_i K(x_i, x)$
- Predict class 1 if $f(x) > 0$, otherwise class -1

Multi-class Classification with SVMs

Multi-class SVM

SVMs are inherently binary classifiers. For multi-class problems, approaches include:

- **One-vs-All**: Train K SVMs, each separating one class from all others
- **One-vs-One**: Train $\binom{K}{2}$ SVMs, each separating one class from another

One-vs-All (One-vs-Rest) Approach

In the one-vs-all approach:

- Train K binary classifiers, one for each class
- For each classifier, positive examples are from one class, negative examples from all other classes
- For prediction, apply all classifiers and select the class with highest confidence (furthest from the decision boundary)

One-vs-One Approach

In the one-vs-one approach:

- Train $\binom{K}{2} = \frac{K(K-1)}{2}$ binary classifiers, one for each pair of classes
- For prediction, each classifier votes for one class
- The class with the most votes wins

Multi-class Classification

Consider a flower classification problem with three classes: Setosa, Versicolor, and Virginica.

- Features: Petal length and petal width
- One-vs-All approach:
  - Classifier 1: Setosa vs. non-Setosa
  - Classifier 2: Versicolor vs. non-Versicolor
  - Classifier 3: Virginica vs. non-Virginica
- One-vs-One approach:
  - Classifier 1: Setosa vs. Versicolor
  - Classifier 2: Setosa vs. Virginica
  - Classifier 3: Versicolor vs. Virginica

For a new flower with petal length $= 5$ cm and width $= 1.8$ cm:
One-vs-All approach:

- Classifier 1 (Setosa vs. rest): $f_1(x) = -3.5 < 0$ (not Setosa)
- Classifier 2 (Versicolor vs. rest): $f_2(x) = -0.8 < 0$ (not Versicolor)
- Classifier 3 (Virginica vs. rest): $f_3(x) = 2.3 > 0$ (Virginica)

Prediction: Virginica
One-vs-One approach:

- Classifier 1 (Setosa vs. Versicolor): Votes for Versicolor
- Classifier 2 (Setosa vs. Virginica): Votes for Virginica
- Classifier 3 (Versicolor vs. Virginica): Votes for Virginica

Virginica gets 2 votes, Versicolor gets 1 vote, Setosa gets 0 votes. Prediction: Virginica

SVM for Regression

Support Vector Regression (SVR)

SVR applies the same principles as SVM to regression tasks by:

- Fitting as many instances as possible on a ßtreet"with width controlled by parameter ε
- Allowing some points to be off the street (margin violations)
- Using kernels to handle non-linear regression tasks

Key Differences Between SVR and Linear Regression

- Linear regression minimizes the squared error for all points
- SVR is only concerned with errors larger than ε (the tube width)
- SVR is more robust to outliers
- SVR produces a sparse solution (depends only on support vectors)

Implementing Support Vector Regression

Define the problem

- Determine if a non-linear relationship exists in the data
- Decide if robustness to outliers is important

Select parameters

- ε: Controls the width of the insensitive tube (larger ε means fewer support vectors)
- C: Regularization parameter (smaller C means more regularization)
- Kernel and its parameters (same as for SVM classification)

Train and evaluate

- Train using appropriate solver
- Evaluate using metrics like MSE, MAE, or R^2
- Tune parameters using cross-validation

SVR Example Consider predicting house prices based on size (in square feet):

- Training data: 100 houses with sizes ranging from 1000 to 3000 sq ft and prices from $200,000 to $600,000
- We use SVR with an RBF kernel, $C = 1000$, $\varepsilon = 10000$, and $\gamma = 0.0001$
- For a new house with 2500 sq ft:
  - Predicted price: $475,000
  - Only 15 support vectors were used for this prediction (out of 100 training examples)

Decision Trees and Random Forests

Predictions from Decision Trees

Decision Tree

A decision tree is a tree-like model of decisions where:

- Each internal node represents a test on a feature
- Each branch represents an outcome of the test
- Each leaf node represents a prediction (class label or value)

Decision trees can be used for both classification and regression tasks.

Gini Impurity

Gini impurity measures how often a randomly chosen element would be incorrectly labeled if labeled randomly according to the distribution of labels in the subset:

$$G(Q_i) = 1 - \sum_{k=1}^{K} p_{i,k}^2$$

where $p_{i,k}$ is the proportion of class k observations in node i.

Entropy

Entropy is another measure of impurity:

$$H(Q_i) = -\sum_{k=1}^{K} p_{i,k} \log_2 p_{i,k}$$

Both Gini impurity and entropy are used in building decision trees.

Node Purity and Splitting Criteria

When building a decision tree, nodes are split to maximize "purity:

- A pure node contains only samples of a single class
- Splits are chosen to maximize information gain
- Information gain is the reduction in impurity (Gini or entropy) after a split
- The cost of a split is the weighted average of child node impurities:

$$J(Q_i, \theta) = \frac{m_i^{left}}{m_i} G(Q_i^{left}(\theta)) + \frac{m_i^{right}}{m_i} G(Q_i^{right}(\theta))$$

Making Predictions with Decision Trees

For classification

Follow the tree from root to leaf:

- Start at the root node
- At each internal node, evaluate the feature test
- Follow the branch corresponding to the test outcome
- Continue until reaching a leaf node
- Return the majority class in the leaf as the prediction

For regression

Follow the same process, but:

- Return the average target value of training samples in the leaf
- Trees approximate the target function as a piecewise constant function

Prediction confidence

For classification:

- Confidence can be estimated from class proportions in the leaf
- For example, if a leaf has 80 samples of class A and 20 of class B:
  - Predict class A with 80% confidence
  - Alternative: predict probabilities [0.8, 0.2]

Decision Tree Classification Consider classifying iris flowers based on petal length and width:

- Root node: Ïs petal length < 2.45 cm?"
- If yes: Predict Setosa (all samples with petal length < 2.45 cm are Setosa)
- If no: Ïs petal width < 1.75 cm?"
- If yes: Predict Versicolor
- If no: Predict Virginica

Training Decision Trees

Training a Decision Tree with CART Algorithm

Start with root node

Begin with all training samples in the root node

Find best split

For each feature and potential threshold:

- Calculate the cost function (e.g., Gini impurity or entropy for classification, MSE for regression)
- Choose the feature and threshold that minimizes the cost

Split the node

Divide the data based on the best split found

Recursively build sub-trees

For each child node, repeat steps 2-3 until stopping criteria are met:

- Maximum depth reached
- Minimum samples per node threshold met
- Node becomes pure (all samples belong to same class)

Make predictions

For classification: Predict the majority class in the leaf node For regression: Predict the average value of samples in the leaf node

Building Binary Trees

The CART algorithm builds binary trees:

- Each split creates exactly two child nodes (binary split)
- For categorical features with p possible values, there are $2^{p-1} - 1$ possible binary partitions
- In practice, often use one-hot encoding for categorical features

Training a Decision Tree

Consider a dataset with two features (age and income) and a binary target (buy vs. don't buy):

- 10 samples: 6 "buyänd 4 "don't buy"
- Root node Gini impurity: $1 - (\frac{6}{10})^2 - (\frac{4}{10})^2 = 1 - 0.36 - 0.16 = 0.48$

Evaluate split on age $= 30$:

- Left node (age < 30): 5 samples, 4 "buyänd 1 "don't buy"
- Left node Gini: $1 - (\frac{4}{5})^2 - (\frac{1}{5})^2 = 1 - 0.64 - 0.04 = 0.32$
- Right node (age ≤ 30): 5 samples, 2 "buyänd 3 "don't buy"
- Right node Gini: $1 - (\frac{2}{5})^2 - (\frac{3}{5})^2 = 1 - 0.16 - 0.36 = 0.48$
- Weighted Gini after split: $\frac{5}{10} \times 0.32 + \frac{5}{10} \times 0.48 = 0.16 + 0.24 = 0.40$
- Information gain: $0.48 - 0.40 = 0.08$

Evaluate split on income $= 50$K:

- Left node (income < 50K): 6 samples, 2 "buyänd 4 "don't buy"
- Left node Gini: $1 - (\frac{2}{6})^2 - (\frac{4}{6})^2 = 1 - 0.11 - 0.44 = 0.45$
- Right node (income leq 50K): 4 samples, 4 "buyänd 0 "don't buy"
- Right node Gini: $1 - (\frac{4}{4})^2 - (\frac{0}{4})^2 = 1 - 1 - 0 = 0$
- Weighted Gini after split: $\frac{6}{10} \times 0.45 + \frac{4}{10} \times 0 = 0.27 + 0 = 0.27$
- Information gain: $0.48 - 0.27 = 0.21$

The split on income $= 50$K has higher information gain, so it's selected for the root node.

Regularization in Decision Trees

Regularization in Decision Trees

Decision trees tend to overfit without regularization. Common regularization techniques include:

- **Max depth**: Limit the maximum depth of the tree
- **Min samples split**: Require minimum number of samples to split a node
- **Min samples leaf**: Require minimum number of samples in leaf nodes
- **Max features**: Limit number of features considered for splitting
- **Pruning**: Remove branches that don't improve generalization

Pruning Decision Trees

Pre-pruning (early stopping)

Stop growing the tree early by setting constraints:

- Maximum depth
- Minimum samples required for splitting
- Minimum samples required in leaf nodes
- Minimum impurity decrease required for splitting

Post-pruning

Grow a full tree, then prune back branches:

- Grow a full, deep tree
- Evaluate performance on validation set
- Iteratively remove branches that don't improve validation performance
- Continue until pruning no longer improves performance

Cost-complexity pruning

A systematic approach to post-pruning:

- Define a cost-complexity measure that balances tree complexity and error
- Generate a sequence of trees with decreasing complexity
- Select the tree with best validation performance

White Box vs. Black Box Models

Decision trees are considered "white box"models because:

- The decision-making process is transparent and interpretable
- You can see the exact rules and thresholds used for classification
- Each prediction can be explained by tracing the path through the tree

This contrasts with "black box"models like neural networks, where the decision-making process is opaque.

Random Forests

Random Forest

Random Forest is an ensemble learning method that combines multiple decision trees to improve prediction accuracy and control overfitting. Key aspects include:

- Each tree is trained on a bootstrap sample of the training data (random sampling with replacement)
- When building trees, only a random subset of features is considered for splitting at each node
- Final prediction is made by averaging predictions (regression) or taking majority vote (classification) from all trees

Bagging (Bootstrap Aggregating)

Bagging is a technique to improve model stability and accuracy:

- Generate multiple training sets by sampling with replacement from the original training data
- Train a separate model on each bootstrap sample
- Combine predictions by averaging (regression) or voting (classification)
- Reduces variance without increasing bias

Out-of-Bag Evaluation

In Random Forests, each tree is trained on a bootstrap sample, leaving some samples out (out-of-bag or OOB samples):

- OOB samples act as a validation set for each tree
- For each sample, predictions are made only by trees that didn't use it for training
- OOB error provides an unbiased estimate of the generalization error

Training a Random Forest

Initialize the forest

- Define number of trees n_trees
- Set parameters: max_features, max_depth, min_samples_leaf, etc.

Build individual trees

For each tree:

- Create bootstrap sample from training data
- Build decision tree considering only random subset of features at each split

Make predictions

For a new data point:

- Get prediction from each tree
- Aggregate predictions (average for regression, majority vote for classification)

Feature Importance in Random Forests

Random forests provide a natural way to measure feature importance:

- For each feature, measure the increase in prediction error after permuting its values
- Features with large increases in error are more important
- Alternatively, measure the average decrease in impurity across all trees when a feature is used for splitting

This helps identify which features have the most predictive power.

Random Forest Application Consider a credit scoring application:

- Task: Predict if a customer will default on a loan
- Features: Income, age, employment history, credit history, loan amount
- Model: Random forest with 100 trees
- Each tree uses a bootstrap sample (about 63% of training data)
- At each node, only $\sqrt{5} \approx 2$ features are considered for splitting
- Results:
  - Individual tree accuracy: 78-85%
  - Random forest accuracy: 92%
  - Feature importance: Credit history (40%), income (25%), loan amount (20%), employment history (10%), age (5%)

Tuning Random Forest Hyperparameters

Number of trees (n_estimators)

- Start with a large number (100-500)
- More trees generally improves performance but increases computation
- Performance often plateaus after a certain number

Maximum features (max_features)

- For classification: Default is $\sqrt{n\_features}$
- For regression: Default is $n\_features/3$
- Try different values and use cross-validation

Tree constraints

- max_depth: Controls maximum depth of trees
- min_samples_split: Minimum samples required to split
- min_samples_leaf: Minimum samples required in leaf nodes

Bootstrap options

- bootstrap: Whether to use bootstrap samples (True by default)
- max_samples: Size of bootstrap samples

Other Tree-Based Ensemble Methods

Beyond random forests, other tree-based ensemble methods include:

- **Extra Trees (Extremely Randomized Trees)**: Similar to random forests but uses random thresholds for splitting
- **Gradient Boosting**: Builds trees sequentially, each correcting errors of previous trees
- **AdaBoost (Adaptive Boosting)**: Focuses subsequent trees on examples previous trees misclassified
- **XGBoost, LightGBM, CatBoost**: Optimized implementations of gradient boosting

Generative Models and Naive Bayes

Probability Fundamentals

Probability Basics

Key probability concepts include:

- **Sample Space** (S): Set of all possible outcomes
- **Event**: Subset of the sample space
- **Probability of an Event** ($P(A)$): Measure of how likely event A is to occur
- **Axioms**: Non-negativity, certainty ($P(S) = 1$), and additivity

Joint and Conditional Probability

- **Joint Probability**: $P(A \cap B)$ is the probability of events A and B occurring together
- **Conditional Probability**: $P(B|A) = \frac{P(A \cap B)}{P(A)}$ is the probability of event B given that event A has occurred
- **Independence**: Events A and B are independent if $P(A \cap B) = P(A) \cdot P(B)$

Bayes' Theorem

Bayes' theorem describes how to update probabilities based on new evidence:

$$P(A|B) = \frac{P(B|A) \cdot P(A)}{P(B)}$$

where:

- $P(A|B)$ is the posterior probability
- $P(B|A)$ is the likelihood
- $P(A)$ is the prior probability
- $P(B)$ is the evidence

Law of Total Probability

The law of total probability relates the probability of an event to conditional probabilities:

$$P(A) = \sum_{i=1}^{n} P(A|B_i)P(B_i)$$

where $B_1, B_2, ..., B_n$ form a partition of the sample space (mutually exclusive and collectively exhaustive).

Parameter Estimation

Parameter Estimation

Two common approaches for parameter estimation:

- **Maximum Likelihood Estimation (MLE)**: Finds parameter values that maximize the likelihood of observing the training data
- **Maximum A Posteriori (MAP)**: Incorporates prior beliefs about parameters, finding values that maximize the posterior probability

Likelihood Function

The likelihood function measures how well a statistical model (with given parameters) explains observed data:

$$L(\theta; X) = P(X|\theta)$$

For independent observations, it's the product of individual probabilities:

$$L(\theta; X) = \prod_{i=1}^{n} P(x^{(i)}|\theta)$$

Maximum Likelihood Estimation

Define the likelihood function

Express the probability of observing the data given the parameters:

$$L(\theta; X) = P(X|\theta) = \prod_{i=1}^{n} P(x^{(i)}|\theta)$$

Take the logarithm

Convert to log-likelihood for easier computation:

$$\log L(\theta; X) = \sum_{i=1}^{n} \log P(x^{(i)}|\theta)$$

Find the maximum

Set the derivative to zero and solve for parameters:

$$\frac{\partial}{\partial \theta} \log L(\theta; X) = 0$$

Check for maximum

Verify second derivative is negative to ensure maximum

MLE for Bernoulli Distribution Consider estimating the probability of heads for a biased coin:

- Data: 7 heads and 3 tails in 10 flips
- Model: Bernoulli distribution with parameter p (probability of heads)
- Likelihood: $L(p; X) = p^7(1-p)^3$
- Log-likelihood: $\log L(p; X) = 7 \log(p) + 3 \log(1-p)$
- Derivative: $\frac{\partial}{\partial p} \log L(p; X) = \frac{7}{p} - \frac{3}{1-p}$
- Setting to zero: $\frac{7}{p} - \frac{3}{1-p} = 0$
- Solving: $p_{MLE} = \frac{7}{10} = 0.7$

Generative vs. Discriminative Models

Generative Models

Generative models learn the joint probability distribution $P(X, Y)$ of inputs X and labels Y. They model how the data is generated and can generate new samples:

- Learn $P(X|Y)$ (likelihood) and $P(Y)$ (prior)
- Use Bayes' theorem to calculate $P(Y|X)$ for predictions
- Examples include Naive Bayes and Hidden Markov Models

Discriminative Models

Discriminative models directly learn the conditional probability $P(Y|X)$ or the mapping from inputs to outputs:

- Focus on decision boundaries between classes
- Don't model the data generation process
- Examples include Logistic Regression, SVM, and Neural Networks

Comparing Generative and Discriminative Models

- **Data efficiency**: Generative models often perform better with less data
- **Expressiveness**: Generative models capture the full distribution
- **Accuracy**: Discriminative models typically achieve better classification accuracy
- **Outlier detection**: Generative models can identify outliers as low-probability instances
- **Missing features**: Generative models can handle missing features naturally

Generative vs. Discriminative Approach

Consider a simple email classification task:

- Task: Classify emails as spam or not spam
- Features: Presence of words like "money", "free", "meeting"

Discriminative approach (Logistic Regression):

- Directly models: $P(\text{spam}|\text{words})$
- Learns weights for each word that predict spam probability
- Example function: $P(\text{spam}|\text{words}) = \sigma(0.8 \times \text{money} + 0.9 \times \text{free} - 0.7 \times \text{meeting})$

Generative approach (Naive Bayes):

- Models: $P(\text{words}|\text{spam})$ and $P(\text{spam})$
- Learns how likely each word appears in spam and non-spam
- For example: $P(\text{money}|\text{spam}) = 0.3$, $P(\text{money}|\text{not spam}) = 0.05$
- Uses Bayes' theorem: $P(\text{spam}|\text{words}) = \frac{P(\text{words}|\text{spam})P(\text{spam})}{P(\text{words})}$

Naive Bayes Classifier

Naive Bayes

Naive Bayes is a probabilistic classifier based on Bayes' theorem with a "naiveïndependence assumption between features:

$$P(Y|X) = \frac{P(X|Y) \cdot P(Y)}{P(X)}$$

The naive assumption allows us to write:

$$P(X|Y) = \prod_{i=1}^{n} P(X_i|Y)$$

This simplifies calculations significantly.

Implementing Naive Bayes

Calculate prior probabilities

Compute $P(Y = y_k)$ for each class y_k based on their frequency in the training data

Calculate likelihoods

For each feature X_i and class y_k, compute the likelihood $P(X_i|Y = y_k)$:

- For discrete features (Multinomial NB): Use frequency counts
- For binary features (Bernoulli NB): Use presence/absence frequencies
- For continuous features (Gaussian NB): Estimate using Gaussian distribution

Apply Bayes' theorem for prediction

For a new instance x, calculate for each class y_k:

$$P(Y = y_k|X = x) \propto P(Y = y_k) \prod_{i=1}^{n} P(X_i = x_i|Y = y_k)$$

Predict the class with highest posterior probability.

Types of Naive Bayes

Different implementations of Naive Bayes exist for different types of data:

- **Bernoulli Naive Bayes**: For binary features (presence/absence)
- **Multinomial Naive Bayes**: For discrete features (e.g., word counts)
- **Gaussian Naive Bayes**: For continuous features, assuming normal distribution

Naive Bayes for Text Classification Spam detection using Naive Bayes:

- Classes: Spam (1) and Not Spam (0)
- Features: Presence of words like öfferänd "free"
- Prior probabilities: $P(Spam) = 0.3$, $P(NotSpam) = 0.7$
- Likelihoods:
  - $P(\ddot{o}fferSpam) = 0.6$, $P(\ddot{o}fferNotSpam) = 0.1$
  - $P(\text{"}freeSpam) = 0.8$, $P(\text{"}freeNotSpam) = 0.2$
- For an email containing both öfferänd "free":
  - $P(Spam|\ddot{o}ffer\text{"}, \text{"}free\text{"}) \propto 0.3 \times 0.6 \times 0.8 = 0.144$
  - $P(NotSpam|\ddot{o}ffer\text{"}, \text{"}free\text{"}) \propto 0.7 \times 0.1 \times 0.2 = 0.014$
  - Since $0.144 > 0.014$, predict Spam

Laplace Smoothing

Laplace smoothing (or additive smoothing) addresses the zero-probability problem:

- Problem: If a feature value never appears in a class during training, its likelihood is zero
- Solution: Add a small count (typically 1) to all feature values
- Formula: $P(X_i = x_i|Y = y_k) = \frac{count(X_i=x_i, Y=y_k)+\alpha}{count(Y=y_k)+\alpha \cdot |V|}$
- Where $|V|$ is the number of possible values for feature X_i and α is the smoothing parameter

Text Classification with Naive Bayes

Preprocess text

- Tokenize text into words
- Remove stop words (optional)
- Apply stemming or lemmatization (optional)
- Create feature vectors (e.g., bag of words, TF-IDF)

Train Multinomial Naive Bayes

- Calculate class priors $P(C_k)$
- Calculate word likelihoods $P(w_i|C_k)$ for each word and class
- Apply Laplace smoothing to handle unseen words

Classify new documents

- Tokenize and preprocess
- Apply Bayes' theorem using log probabilities to avoid underflow
- Assign document to class with highest probability

Advantages and Disadvantages of Naive Bayes

Advantages:

- Simple and fast to train
- Works well with high-dimensional data
- Performs well with small training sets
- Handles missing values naturally

Disadvantages:

- Independence assumption is often violated in real data
- Not suitable for regression problems
- May be outperformed by more sophisticated models for complex tasks
- Probability estimates may be unreliable

Clustering

Unsupervised Learning

Unsupervised Learning

In unsupervised learning, the training data does not contain any output (target) values. The goal is to model the underlying distribution of the data to describe its structure, discover hidden patterns, and gain insights.

Introduction to Clustering

Clustering

Clustering is the task of grouping a set of objects in such a way that objects in the same group (called a cluster) are more similar to each other than to objects in other groups. Given a set of M data points $X = \{x^{(1)}, x^{(2)}, \ldots, x^{(M)}\}$, where each data point consists of N features $x^{(i)} = (x_1^{(i)}, \ldots, x_N^{(i)}) \in \mathbb{R}^N$, and a distance measure d, a clustering algorithm separates the data into K clusters.

Types of Clustering

There are two main types of clustering:

- **Hard Clustering**: Each data point is assigned to exactly one cluster
- **Soft Clustering**: Each data point is assigned a probability or membership degree for each cluster

Clustering Application Customer segmentation for targeted marketing:

- Data: Customer purchase history, demographics, website behavior
- Clustering reveals natural customer segments:
  - Cluster 1: Young, high-income professionals who buy luxury items
  - Cluster 2: Middle-aged parents who focus on household essentials
  - Cluster 3: Budget-conscious shoppers who primarily buy discounted items
- Marketing strategies can be tailored to each segment

K-Means Algorithm

K-Means Algorithm

K-means is a clustering algorithm that aims to partition M data points into K clusters, where each data point belongs to the cluster with the nearest mean (centroid). The algorithm minimizes the within-cluster sum of squares (inertia):

$$\sum_{k=1}^{K} \sum_{x^{(i)} \in C_k} ||x^{(i)} - \mu_k||^2$$

where C_k is the set of points in cluster k and μ_k is the centroid of cluster k.

Implementing K-Means

Initialize centroids

Choose K initial centroids:

- Random selection: Randomly select K data points
- K-means++: Select centroids that are farther apart

Assign points to closest centroid

For each data point $x^{(i)}$:

- Calculate distance to each centroid
- Assign the point to the closest centroid's cluster

Update centroids

For each cluster k:

- Calculate the mean of all points in the cluster
- Set the new centroid to this mean: $\mu_k = \frac{1}{|C_k|} \sum_{x^{(i)} \in C_k} x^{(i)}$

Repeat until convergence

Repeat steps 2 and 3 until:

- Centroids no longer change significantly
- Maximum number of iterations is reached
- Assignment of points to clusters stabilizes

K-Means Step-by-Step

Consider a dataset with six 2D points: (1,1), (2,1), (4,3), (5,4), (1,2), (2,2)

- Initialize $K = 2$ centroids: $\mu_1 = (1, 1)$ and $\mu_2 = (5, 4)$

Iteration 1:

- Assign points to clusters:
  - Cluster 1: (1,1), (2,1), (1,2), (2,2) [closer to μ_1]
  - Cluster 2: (4,3), (5,4) [closer to μ_2]
- Update centroids:
  - $\mu_1 = \frac{(1,1)+(2,1)+(1,2)+(2,2)}{4} = (1.5, 1.5)$
  - $\mu_2 = \frac{(4,3)+(5,4)}{2} = (4.5, 3.5)$

Iteration 2:

- Assign points to clusters:
  - Cluster 1: (1,1), (2,1), (1,2), (2,2) [closer to μ_1]
  - Cluster 2: (4,3), (5,4) [closer to μ_2]
- The assignments haven't changed, so the algorithm has converged

Final clusters:

- Cluster 1: (1,1), (2,1), (1,2), (2,2) with centroid (1.5, 1.5)
- Cluster 2: (4,3), (5,4) with centroid (4.5, 3.5)

K-means++

K-means++ is an initialization method for K-means that selects initial centroids that are far away from each other:

1. Choose the first centroid randomly from the data points
2. For each subsequent centroid, select a data point with probability proportional to the squared distance to the nearest existing centroid
3. Repeat until K centroids are selected

This approach typically leads to better and more consistent results than random initialization.

Evaluating Clustering Quality

Inertia

Inertia (within-cluster sum of squares) measures how internally coherent clusters are:

$$\text{Inertia} = \sum_{k=1}^{K} \sum_{x^{(i)} \in C_k} ||x^{(i)} - \mu_k||^2$$

Lower inertia indicates better clustering, but it always decreases with more clusters.

Silhouette Score

The silhouette score measures how similar objects are to their own cluster compared to other clusters:

$$s(i) = \frac{b(i) - a(i)}{\max(a(i), b(i))}$$

where:

- $a(i)$ is the average distance of point i to other points in the same cluster
- $b(i)$ is the minimum average distance of point i to points in a different cluster

The silhouette score ranges from -1 to 1, with higher values indicating better clustering.

Elbow Method

The elbow method helps find the optimal number of clusters:

- Run K-means with different values of K
- Plot inertia vs. number of clusters
- Look for the ëlbow"point where adding more clusters gives diminishing returns

Choosing the Optimal Number of Clusters

Elbow method

- Run K-means for a range of K values (e.g., 1-10)
- Plot inertia (within-cluster sum of squares) vs. K
- Identify the ëlbow"point where the rate of decrease sharply changes

Silhouette method

- Run K-means for a range of K values
- Calculate the average silhouette score for each K
- Choose K that maximizes the average silhouette score

Gap statistic

- Compare the within-cluster dispersion to that expected under a null reference distribution
- Choose K that maximizes the gap statistic

Domain knowledge

- Consider business requirements or prior knowledge
- Sometimes the number of clusters has a natural interpretation in the domain

DBSCAN Algorithm

DBSCAN

Density-Based Spatial Clustering of Applications with Noise (DBSCAN) is a clustering algorithm that groups together points that are closely packed together (points with many nearby neighbors):

- Does not require specifying the number of clusters in advance
- Can find arbitrarily shaped clusters
- Has a notion of noise (points that don't belong to any cluster)
- Based on two parameters: ϵ (maximum distance between points) and minPts (minimum number of points in a neighborhood)

Types of Points in DBSCAN

DBSCAN classifies points into three types:

- **Core Point**: Has at least minPts points within distance ϵ
- **Border Point**: Has at least one core point within distance ϵ but fewer than minPts points within distance ϵ
- **Noise Point (Outlier)**: Neither a core point nor a border point

Implementing DBSCAN

Set parameters

- ϵ: Maximum distance between points in the same neighborhood
- minPts: Minimum number of points required to form a dense region

Classify points

For each unvisited point P:

- Mark P as visited
- Find all points within distance ϵ of P (its neighborhood)
- If neighborhood has fewer than minPts, mark P as noise (can be changed later)
- If neighborhood has at least minPts, start a new cluster with P as a core point

Expand clusters

For each core point P in a cluster:

- Find all points in the ϵ-neighborhood of P
- For each point Q in the neighborhood:
  - If Q is unvisited or marked as noise, add it to the cluster
  - If Q is a core point, recursively expand the cluster from Q

Advantages of DBSCAN

DBSCAN has several advantages over other clustering algorithms:

- Doesn't require specifying the number of clusters in advance
- Can find arbitrarily shaped clusters, not just spherical ones
- Robust to outliers (identifies them as noise points)
- Only needs two parameters (ϵ and minPts)
- Can handle clusters of different sizes and densities (to some extent)

Limitations of DBSCAN

DBSCAN also has some limitations:

- Struggles with varying density clusters
- Sensitive to parameter choices
- Doesn't work well with high-dimensional data due to the "curse of dimensionality"
- Not deterministic when points can be reached through multiple paths
- Computationally expensive for large datasets (though optimized implementations exist)

DBSCAN Application Consider identifying geographical regions in a city based on crime density:

- Data: Latitude and longitude coordinates of crime incidents
- Parameters: $\epsilon = 0.5$ km, minPts $= 10$
- Results:
  - Cluster 1: Downtown area with high crime density
  - Cluster 2: Entertainment district with moderate crime
  - Cluster 3: Shopping mall area with focused retail theft
  - Noise points: Isolated incidents throughout the city
- Advantages: Naturally identifies crime hotspots without predefined number of clusters

Distance Metrics

Common Distance Metrics

The choice of distance metric affects clustering results:

- **Euclidean Distance**: $d(p,q) = \sqrt{\sum_{i=1}^{N}(q_i - p_i)^2}$ (standard straight-line distance)
- **Manhattan Distance**: $d(p,q) = \sum_{i=1}^{N}|q_i - p_i|$ (sum of absolute differences)
- **Maximum Distance**: $d(p,q) = \max_i |q_i - p_i|$ (largest difference along any dimension)
- **Cosine Similarity**: $d_{cos}(p,q) = \frac{\sum_{i=1}^{N} p_i q_i}{\sqrt{\sum_{i=1}^{N} p_i^2}\sqrt{\sum_{i=1}^{N} q_i^2}}$ (angle between vectors)

Choosing the Right Distance Metric

Understand the data

- Consider the nature of features (categorical vs. numerical)
- Consider the relative importance of features
- Consider the scale of different features

Consider the application domain

- Euclidean: Good for continuous data in low dimensions
- Manhattan: Good when movement is restricted along axes (e.g., city blocks)
- Cosine: Good for text documents or high-dimensional data where magnitude is less important
- Correlation-based: Good when patterns matter more than absolute values

Preprocessing matters

- Standardize or normalize features before clustering
- Consider dimensionality reduction for high-dimensional data
- Handle categorical variables appropriately (one-hot encoding, etc.)

Other Clustering Algorithms

Hierarchical Clustering

Hierarchical clustering builds a tree of clusters (dendrogram):

- **Agglomerative (bottom-up)**: Start with each point as its own cluster, then merge closest clusters
- **Divisive (top-down)**: Start with all points in one cluster, then recursively divide
- Doesn't require specifying the number of clusters in advance
- Can visualize cluster structure at different levels
- Linkage methods (single, complete, average, Ward) determine how cluster distances are measured

Gaussian Mixture Models (GMM)

GMMs are a probabilistic clustering approach:

- Model the data as a mixture of several Gaussian distributions
- Each Gaussian component represents a cluster
- Provides soft cluster assignments (probabilities)
- Parameters estimated using Expectation-Maximization (EM) algorithm
- More flexible than K-means but more computationally intensive

Comparison of Clustering Algorithms

Consider clustering customer purchase behavior:

- Features: Average purchase amount, purchase frequency, time since last purchase
- Dataset: 1000 customers

K-means:

- Fast computation (convergence in 15 iterations)
- Creates spherical clusters of similar sizes
- Some customers don't fit well in any cluster
- Needs careful initialization

DBSCAN:

- Identifies high-spending and frequent shopper clusters effectively
- Marks outliers (abnormal purchase patterns) as noise
- Doesn't force customers into clusters artificially
- More computationally expensive than K-means

Hierarchical Clustering:

- Provides insight into relationships between clusters
- Shows that the high-spending cluster has two distinct sub-groups
- Most computationally expensive of the three
- Results sensitive to the linkage method chosen

Implementing a Clustering Project

Define the problem

- Clarify business objectives
- Understand what insights are needed
- Define what makes a "good"cluster in your context

Prepare the data

- Clean and preprocess data (handle missing values, outliers)
- Select relevant features
- Normalize or standardize features
- Consider dimensionality reduction if necessary

Choose and apply clustering algorithms

- Select appropriate algorithms based on data characteristics
- Try multiple algorithms and parameter settings
- Evaluate using internal validation metrics

Interpret and validate results

- Profile clusters to understand their characteristics
- Visualize clusters using dimensionality reduction (PCA, t-SNE)
- Validate with domain experts
- Test stability by rerunning with different samples

Implement findings

- Create actionable insights from clusters
- Develop a strategy to use clusters in business processes
- Set up a system to assign new data points to clusters