MachLe

Summary

Lukas Schöpf

29. Januar 2025

1 Introduction

1.1 Data Types

- Numerical/Quantitive, discret: Countable

  Example: Number of rejected Loans, classes taken this semster

- Numerical/Quantitive, Continuos: Interval data

  Example: Distance, mg of drug taken, size of a house

- Categorical, ordinal: distinct and can be ordered

  Example: Credit score can be {low, medium, high}

- Categorical, nominal: categories cannot be orderd

  Example: gender, eye color

1.2 Machine Learning Paradigms

- Unsupervised Learning: Discover and explore structure from unlabelled data
- Supervised Learning: Learn to predict/forecast an output of interest, we know what we want to predict and labelled data is available

1.2.1 Unsupervised Learning

Tasks:

- Dimensionality reduction
- Feature Learning
- Matrix compilation
- Anomaly detection
- Generating data

1.2.2 Supervised Learning

Given a set of features/attributes for some objects and also the ouput/target value of what we want to predict. The Supervised ML task: Given a new object and its features what would be the output value:

- Regression: Ouput is a numeric value
- Classification: Ouput is a categorical value

1.3 Data Preparation/Preprocessing

Data will rarly be in the format and quality needed for analytics and model training and several of these operations will be needed:

- Data integration/consolidation: Collects and merges data from multiple sources into coherent data store
- Data cleaning: removing or modifying incorrect data, identify and reduce noise in data
- Data transformations: normalize, discretize or aggregate the data
- Data reduction: reduce data size by reducing the number of samples or reducing the number of attributes, balance skewed data

Low quality data will result in low quality results

2 Supervised ML, Similarities, kNN & Performance measurments

2.1 Supervised ML

- Goal: Find the best Hypothesis

$$H^* = \arg\min_{H \in \mathcal{H}} \sum_{i=1}^{n} loss(x_i, y_i)$$

- Loss:

$$loss(x_i, y_i) = (H(x_i) - y_i)^2$$

- Model: $H(x) = a + bx + cx^2 + \cdot$

Task of the Learning Algorithem to find best parameters a, b, c(those that minimize the loss)

2.1.1 Overfitting

Model learns traning data but doesnt generalize well.

2.1.2 Training and test error

Dataset gets split in Training set and Test set (80%/20%)
$error_{train}$: Error form trained model on train set
$error_{test}$: Estimate of the true error (generalization error). Error from trained model on test set.

2.2 kNN

Pros:

- Simple and intuitive
- Multiclass
- Interpretable

Cons:

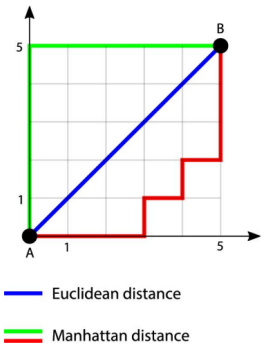
- Curse of Dimensionality
- Sensitive to noise
- Computationally expensive for large datasets

Given training set $X = (x_1, \text{class}_1)(x_2, \text{class}_2)$
Given a new instance $x_?$:

- Find the k-closest examples x_i to $x_?$ in the traning set
- Classify $x_?$ based on the majority vote of $\{x_{NN1}, x_{NN2}, \dots\}$

2.2.1 Distance and similarity measurments

Given 2 point $\mathbf{q} = (q_1, q_2, \dots, q_n)$ and $\mathbf{p} = (p_1, p_2, \dots, p_n)$. n is the number of dimensions.



Euclidean Distance

$$d(\mathbf{q},\mathbf{p}) = \sqrt{\sum_{i=1}^{n}(q_i - p_i)}$$

Manhatten Distance

$$d(\mathbf{q},\mathbf{p}) = ||\mathbf{q}-\mathbf{p}||_1 = \sum_{i=1}^{n}|q_i - p_i|$$

2.3 Finding k

$k_{opt} \in \{1, 2, \ldots, N\}$ N = # of training samples
Extreme cases:

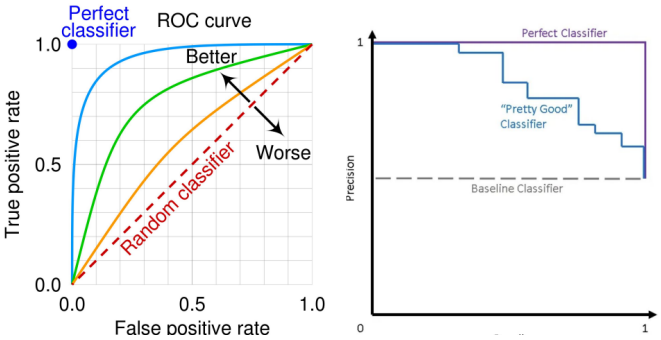
- $k = 1$: kNN = NN
- $k = N$: Majority class

2.4 Performance measures

- Accuracy: #corr / #all = (TP + TN) / (TP + FN + FP + TN)
- Error: #wrong / #all = 1 - Accuracy
- Recall,Sensitivity: TP /(TP + FN), How many relevant samples are correctly detected
- Specificity: TN/(TN + FP)
- Precision: TP/(TP + FP), How many detected samples are relevant
- F1 score: 2 · Precision · Recall / (Precision + Recall)

2.4.1 ROC (Receiver Operating Characteristic) curve

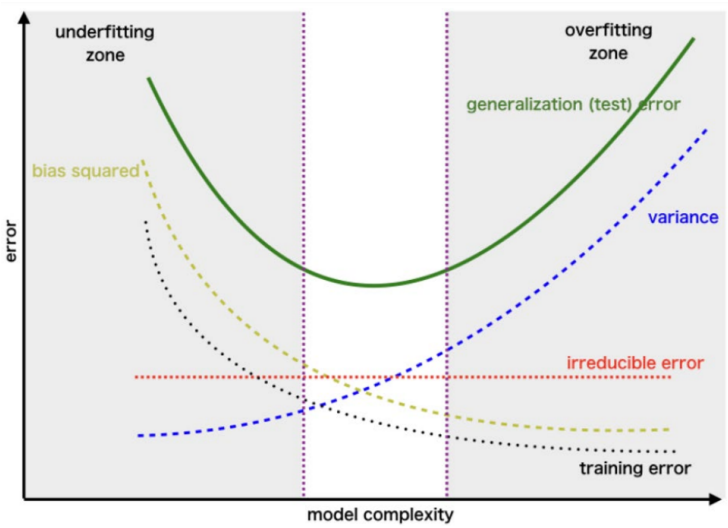
TPR, FPR, to find best threshold.



2.4.2 PRC(Precision-Recall Curve)

Precision, Recall, to find best threshold.

3 Bias-Varinace tradeoff



3.1 No free lunch Theorem

There is no universally best learner (across problems):

3.2 Ockham's Razor

Given 2 models with the same empirical (test) error, the simpler one should be preferred because simplicity is desirable in itself.

3.3 Error sources and bias-variance tradeoff

Different Error sources:

- The model: the best hypothesis is at distance to the true function
- The dataset: different datasets potentially provide different information
- Uncertainty in (X,Y) and its representation:

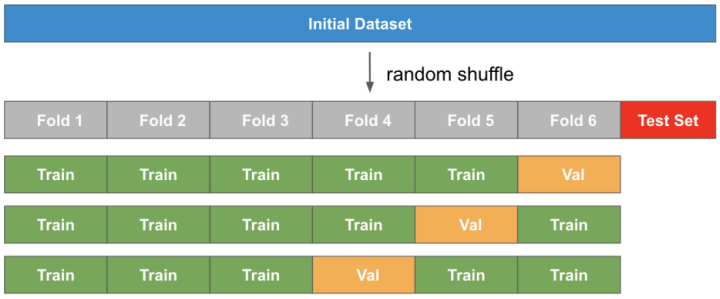
  Partial view of the task: have all relevant features been observed?

  Noisy data

Error Decomposit MSE:

$$E_{MSE} = bias + variance + Irreducible\,error$$

Bias(systematic error): average predictions deviation from the truth
Variance(dependence on specific sample): Sensitivity of prediction to specific training sample
Irreducible error(random nature of process): due to noise
Generally, for more complex/capable model: bias ↓,variance ↑.
Its a trade-off: Only way to redue both is to increase the size of the dataset.



$$\text{Error\_Estimate} = \frac{1}{K}\sum_1^K val\_error_{foldK}$$

3.4 Model selection: Validation score and CV score

k-Fold Corss Validation(CV): We can get a more realistic estimate of the test error using many validation sets (Typically K is 5 to 10)

3.4.1 Error Estimate Summary

In practice:

- Validation score(s) or CV score provide estimates of the test error
- The test error provides an estimate of the true error
- Never use any test data in the model training and model selection process

Which model to choose?

- The one with best validation or CV score
- Use student's t-test to check that an improvement is significant
- Ockham's razor: prefer simpler models in abscence of other evidence

Model selection is an empirical science.

3.5 Loss minimization (gradient descent)

Linear Model: $f(x) = w_0 + w_1 x = \hat{y}$

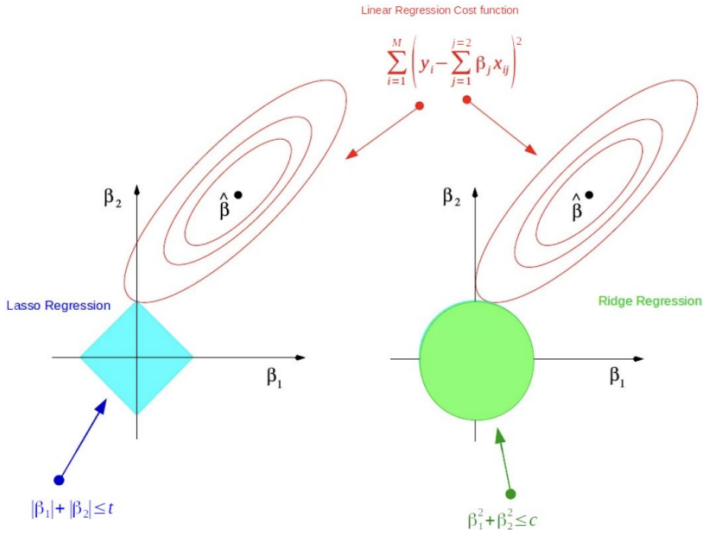
$$RSS = \sum_{i=1}^{N}(y_i - \hat{y_i})^2$$

Use gradient of RSS to find optimal weights for model.

$$\mathbf{w}_{new} = \mathbf{w}_{old} - stepsize \cdot \frac{df(\mathbf{w})}{d\mathbf{w}}$$

The effectiveness of gradient descent depends on the choice of the learning rate (step size):

- Too big, might not reach optimal value
- Too small, it will take a long time to converge



3.6 Regularization

Reduce overfitting of model by penalizing model complexity Tradeoff:

- increase bias
- decrease variance

$$H^* = \arg\min_{H\in\mathcal{H}} \sum_{i=1}^{n} \mathcal{L}_\theta(x_i, y_i) + \lambda R(H)$$

$R(H)$ is the models complexity. λ controlls the model complexity.

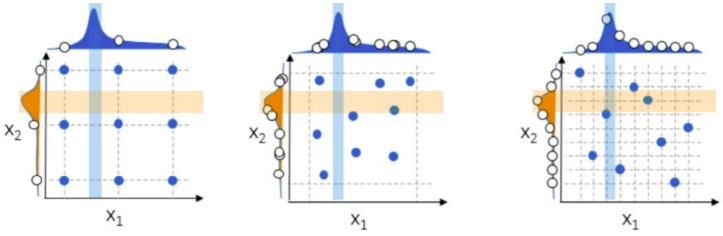
- Lasso:L_1 regularization

$$\arg\min \underbrace{||y - X\beta||_2^2}_{\text{Loss}} + \lambda \underbrace{||\beta||_2^2}_{\text{Penalty}}$$

- Ridge:L_2 regularization

$$\arg\min \underbrace{||y - X\beta||_2^2}_{\text{Loss}} + \lambda \underbrace{||\beta||_1}_{\text{Penalty}}$$

3.7 Hyperparameter tuning



- Grid-search
- Random-search
- {Optimization}

4 Decision Tree, Ensamble Methods & Random Forest

4.1 Decision Tree

- A flow-chart-like tree structure
- Internal node denotes a test on a attribute
- Branch represent an outcome of the test
- Leaf nodes represent class labels or class distribution

Gini Index:

$$I_G = 1 - \sum_{j=1}^{c} p_j^2$$

Entropy:

$$I_H = -\sum_{j=1}^{c} p_j \log(p_j)$$

p_j is the proportion of samples that belongs to calss j. Tree construction:

1. Initialization: whole region R_0(i.e., all given data)
2. Repeat:
    - For each region R_i, for each feature X_j, for each split $R_i = R_{i,l} \cup R_{i,r}$ with respect to feature x_j. Calculate change in impurity score (e.g., gini, entropy, error)
    - Choose best split,i.e., maximum decrease of the impurity score
    - Replace R_i with the two new split regions

Avoiding overfitting:

- Select a proper depth of the tree
- Select a proper minimum number of samples in a leaf to stop further splitting
- Tree pruning - remove split nodes bottom up or top down

All these performed using either a validation set or cross validation!

4.2 Pros/Cons

Pros:

- Easily visualized and interpreted
- No feature normalization or scaling needed
- Works well with mixed feature data types (categorical, continuous)

Cons:

- Easly ovefits
- Not robust, high variance

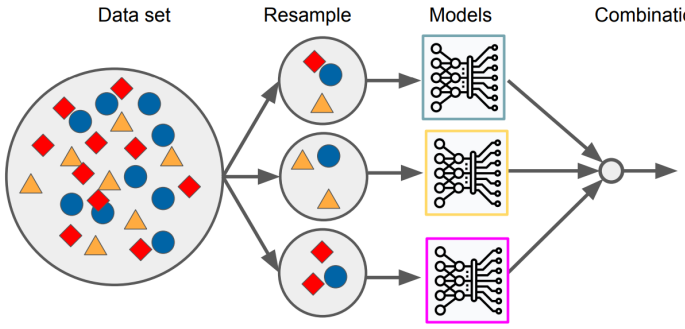
4.3 Ensamble methods

Appproach:

- Suppose you have n classifier
- Each classifier has error rate e
- Assume the classifier are independent
- Take Majority vote

The combined result is wrong if $n/2$ classifiers are wrong. According to central limit theorem variance reduces by factor n.

4.3.1 Bagging



To have independent classifier use many independent training sets S_i to train models.

- Variance reduces linearly (sub-linearly in practice beacuse S_i are correlated)
- Bias unchanged (increases slightly in practice)

4.3.2 Boosting

Weight samples. Samples where the model makes mistakes are weighted higher.
Algorithm:

1. Initialization: Train first model on data
2. Repeat:
    - Compute error of the model on each training sample
    - Give higher importance to samples where the model makes mistakes
    - Train next model using importance weighted training samples

In each iteration, introduce a weak model to compensate the shortcoming of the existing string (= combined) model.

Adaptive Boosting(AdaBoost)

Take a weak learning algorithm and turn it into a strong one by making it focus more on the accurate predictions of difficult cases.

$$F_T(x) = \sum_{t=1}^{T} f_t(x)$$

In each iteration t:

- A new weak classifier $h(x_i)$ with a coefficant α is added to the existing ones such that the error E_t of the ensemble at iteration t is minimized.

$$E_t = \sum_i [F_{t-1}(x_i) + \alpha_t h(x_i)]$$

- The new weak classifier is training using a weighted training set where the weight assigned to each sample is identical to the error of the current ensemble classifier on that sample $E(F_{t-1}(x_i))$.

4.3.3 Comparison

No Ensamble:

- complete training set, train one model

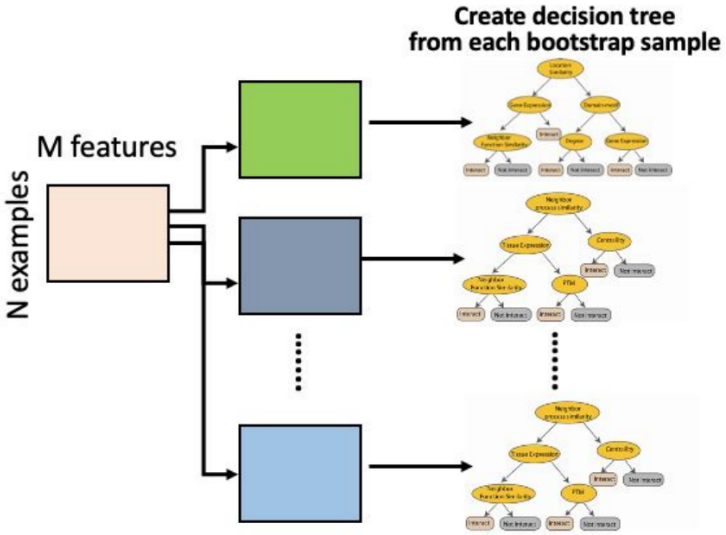
Bagging:

- randomly sample with replacement to obtain different training set
- minimizes variance (usually cannot reduce bias) – fights overfitting
- Computationally efficient (all models can be trained in parallel)

Boosting:

- randomly sample with replacement over weighted data to obtain different trainin sets
- Minimize bias by adding models to the ensemble – fight underfitting
- Address variance by using simple models with low variance

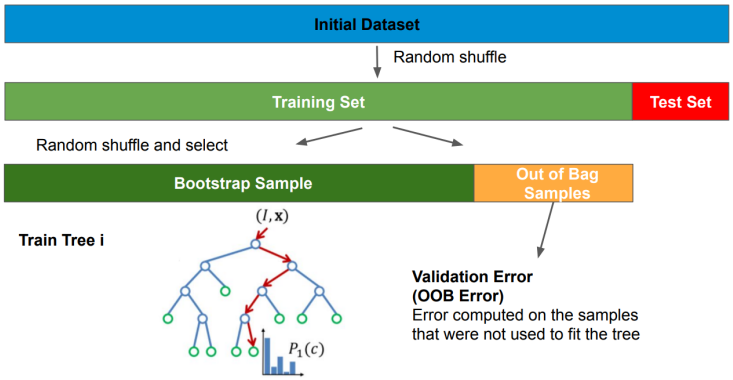
4.3.4 Random forest



Basic idea:

- Grow many trees in bootstrapped samples of training data
- Minimize bias by growing trees sufficiently deep (overfitting)
- Maximize variance reduction by minimizing correlation between trees by means of bootstrapping data for each tree and sampling variable set at each node
- Reduce variance of noisy but unbiased trees by averaging

Out of Bag Errors (OOB Error)



4.3.5 Summary Random Forrest

Pros:

- Simple - no assumption of the underlying distribution
- OOB error for free
- Many variables, even when they are not relevant for the task at hand or noisy
- Robust against outliers
- Multiclass
- Limit overfitting (trees have to be independent!)
- Unbalanced dataset (supsampling)

5 Probability Recap, Loss Functions, Logistic Regression, Neural Networks Intro

5.1 Probability Basics

Random Variable (RV) x denotes a quatity that is uncertain, discret or continuous. $p(x = \mathbb{X})$: the probability of variable x being in state $\mathbb{X}$.
Domain of RV denotes all the values it can take (states it can be in). $\text{dom}(coin) = \{\text{heads,tails}\}$
Joint distribution of two RVs x and y takes a particular combination of values and the joint probability density satisfies:

$$\int\int Pr(x, y) \cdot dxdy = 1$$

Marginal distributions $Pr(x)$ and $Pr(y)$ are obtained by:

$$\int Pr(x, y) \cdot dx = Pr(y)$$

$$\int Pr(x, y) \cdot dy = Pr(x)$$

Conditional probability $Pr(x|y)$ is the probability of a variable x taking a certain value assuming we know the value of y:

$$Pr(x|y) = \frac{Pr(x,y)}{Pr(y)}$$

5.1.1 probability rules

Sum rule:

$$p(X) = \sum_{Y} p(X,Y)$$

Product rule:

$$p(X,Y) = p(Y|X) \cdot p(X) = p(X|Y) \cdot p(Y)$$

Bayes Theorem:

$$p(Y|X) = \frac{p(X|Y) \cdot p(Y)}{p(X)} = \frac{p(X|Y) \cdot p(Y)}{\sum_{y \in Y} p(X,y)} = \frac{p(X|Y) \cdot p(Y)}{\sum_{y} p(X,y) \cdot p(y)}$$

Probability distributions and PDFs

$$\mathbb{E}[x] = \mathbb{E}_{x \sim p}[x] = \int x \cdot p(x)\, dx = \mu$$

$$\mathbb{E}[x^2] = \mathbb{E}_{x \sim p}[x^2] = \int x^2 \cdot p(x)\, dx = \mu^2 + \sigma^2$$

$$\mathbf{VAR}[x] = \mathbb{E}[x^2] - \mathbb{E}[x] = \sigma^2$$

5.2 Designing Loss Functions

Loss/cost function measures how bad the model is - the lower the value of the loss function the better the model maps inputs to output $\hat{\phi} = \arg\min [\mathrm{L}[\phi]]$.
Model training is finding parameter values that minimize the loss.
The negative log likelihood (to be minimized) gives us a loss function.

$$\hat{\phi} = \arg\min_{\phi} \left[-\sum_{i=1}^{I} \log [Pr(y_i|f(x_i,\phi))] \right]$$

5.3 Logistic Regression

Binary classification. Output is the probability that the input belongs to a class.

$$y = \text{logistic}(\mathbf{w}^T\mathbf{x}) = \frac{1}{1+e^{-\mathbf{w}^T\mathbf{x}}}$$

5.3.1 Loss function (Binary Cross Entropy Loss)

$$J(\theta) = -\frac{1}{N} \sum_{i=1}^{N} [y_i \log \hat{y}_i + (1-y_i) \log(1-\hat{y}_i)]$$

Leads to the following update rule:

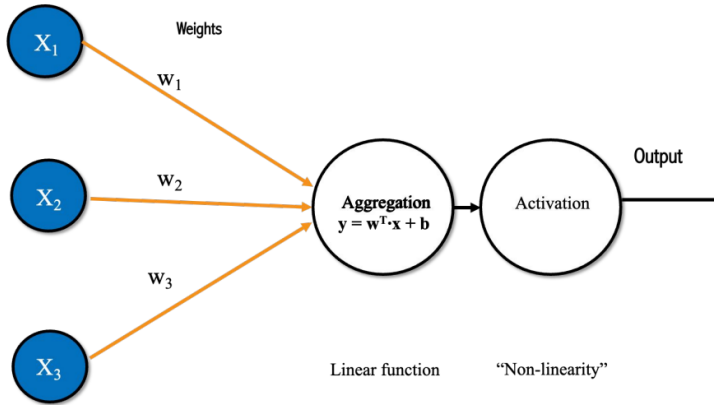
$$\theta_j = \theta_j - \alpha \frac{\partial J(\theta)}{\partial \theta_j}$$

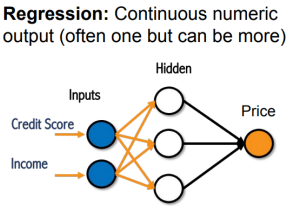
this is simplified to:

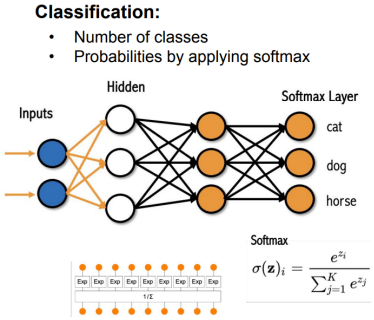
$$\theta_j = \theta_j - \alpha \frac{1}{N} \sum_{i=1}^{N} (\hat{y}_i - y_i) x_i^{(j)}$$

5.4 Neural Networks Basics

Neural Networks (NN) consist of Neurons. The value of a Neuron is defined by its connections to tha last layer, the weights(w_i) of the connection, the bias(b) and the activation funtion of the Neuron.







5.4.1 Number of parameters

Every connection(weights) + every Neuron(bias) = # of parameters

6 Neural Networks

Neural Networks are functions $y = f(x,\theta)$ with parameters θ that map mulitvariate inputs x to mulitvariate outputs y.
Unversal approximation theorem: A shallow neural network (MLP) using nonlinear activation function can approximate any given continuous function defined ib a compact subset of R^D to arbitrary precision given enough hidden units (finite number).

6.1 Loss function

Regression: Mean squared error loss

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

Classification: Cross entropy loss

$$Loss = -\sum_{i=1}^{\#\text{output classes}} y_i \cdot \log(\hat{y}_i)$$

6.2 Training

Repeat:

- Choose a training sample
- Forward pass: Compute the prediction
- Backward pass: If error > 0, update weights

Adjust weigths by Gradient descent

$$w_i = w_i - \alpha \frac{\partial J(w_i)}{\partial w_i}$$

Training Terms:

- Epoch = a forward pass and backward pass complete for all the training examples
- Batch size = the number of training samples in a Batch
- Iteration = forward and backward pass each using a Batch
- Iterations per Epoch = #training data /size of Batch

Avoide overfitting with:

- Regularization (Lasso, Ridge)
- Dropout, turn off some neurons during training
- Batch normalization
- Early stopping

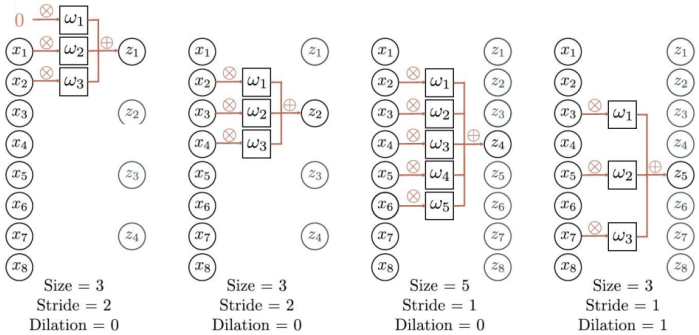
6.3 Convolutional Neural Networks (CNN)

Idea: nearby pixels in an image are correlated - using shared parameters across whole input. A convolution operation is determined:

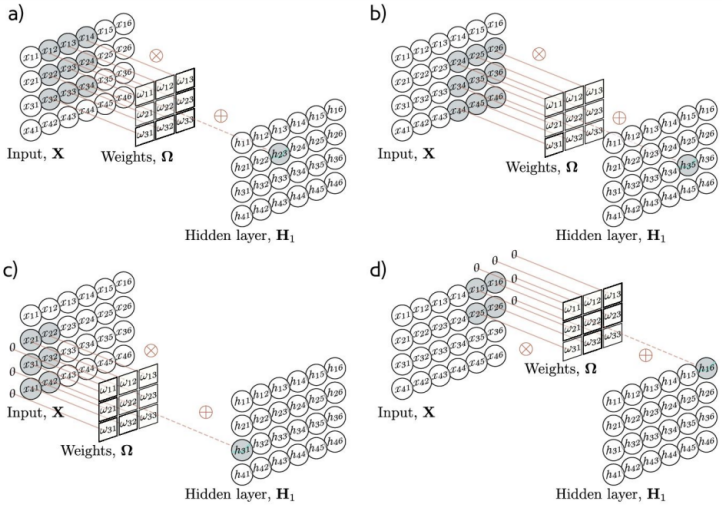
- stride (kernel shift)
- kernel size (typically odd)
- dilation (number of zero weights in kernel)

Convolutional Layer: apply convolution, add bias, apply activation function.
Channels(feature maps, activation maps): mulitple convolutions applied to the input in parallel.

6.3.1 CNN 1D



1D convolution is a weighted sum of nearby inputs.

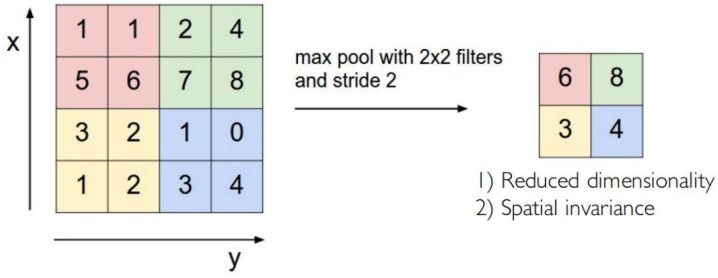


6.3.2 CNN 2D

Using 2D kernles, plus a bias and a activation function.

6.3.3 Pooling

Reduces Dimensions by min,max or average pooling a layer.



6.3.4 Transpose Convolution

Upsampling: Each input contributed multiple times to the output. Useful when output is an image.

7 Feature Engineering

7.1 Data Peparation and Exploration (EDA)

Though exploratory data analysis (EDA) we can often:

- discover important anomalies
- identify limitations in the collection process
- and better inform subsequent goal oriented analysis

Perpare, analyze and visualize data by unsing:

- Histogramm, QQ-Plot
- Scatter-Matrix
- countplots, contingency tables
- Heatmaps

7.2 Feature Preparation/generation

- Data **Cleaning**: Homogenize missing values and different types of the same feature, fix input errors, types, etc.
- **Aggregation|Pivoting**: Necessary when the entity to model is an aggregation from the provided data.
- **Imputation** of missing values. Strategies: mean, median, mode, using a model
- **Binarization**: Transform discrete or continuous numeric features into binary features
- **Binning** (fixed width, adaptive quantile binning): Split numerical values into bins and encode with a bin ID
- **Transformation**(eg. Log-Transform, BoxCox): Compress the range of large numbers or expand the range of small numbers.
- **Scaling** and **normalizing** (min/max,z-score): Scale numerical variables into a certain range.
- **Generate Interaction features**

7.3 Feature selection

There are general two reasons why feature selection is used:

1. Reducing the number of features, to reduce overfitting and improve the generalization of models
2. To gain a better understanding of the features and their relationship to the response variables

These two goals are often ar odds with each other.

7.3.1 Univariate Feature Selection

- Based on univariate statistical tests.Examined for each feature individually
- Good for gaining a better understanding
- For regression: f_regression, mutual_info_regression
- For classification: chi2, f_classif, mutual_info_classif

Pearson Correlation Coefficient:

$$\rho_{X_i,y} = \frac{\text{cov}[X_i, Y]}{\sigma_{X_i} \cdot \sigma_Y} = \frac{\mathbb{E}[X_i - \overline{X}_i] \cdot \mathbb{E}[Y - \overline{Y}]}{\sigma_{X_i} \cdot \sigma_Y}$$

F-Regression test:

$$\frac{\text{TSS} - \text{RSS}/p}{\text{RSS}/(n-p-1)} \sim F_{p,n-p-1}$$

Mutual Information coefficant:

$$I(X,Y) = \sum_{y \in Y} \sum_{x \in X} p(x,y) \cdot \log\left(\frac{p(x,y)}{p(x) \cdot p(y)}\right)$$

7.3.2 Feature selection using linear models & regularization

- Lasso feature selection
- Ridge feature selection
- Linear Models: Subset selection
- Tree-based methods

Ridge feature selection

Forces the coefficient values to be spread out more equally. More usefull for feature interpertation then Lasso feature selection.

Linear Models

Forward stepwise selection: Let $\mathcal{M}_0$ denote the null model with no predictors.

- For $k = 0, \ldots, p-1$:
  1. Consider all $(p-k)$ models that augment the perdictors in $\mathcal{M}_k$ with one additional predictor.
  2. Choose the best among these $(p-k)$ models and call it $\mathcal{M}_{k+1}$. Here best is defined as having smallest RSS or highest R^2.
- Select a single best model among $\{\mathcal{M}_0, \mathcal{M}_1, \ldots, \mathcal{M}_p\}$ using AIC or BIX or the adjusted R^2-score.

Metrics

RSS = Residual Sum of Squares

$$RSS = \sum_{i=1}^{n} (y_i - \hat{y}_i)^2$$

TSS = Total Sum of Squares

$$TSS = \sum_{i=1}^{n} (y_i - \overline{y})^2$$

R^2-score:(% explained variance)

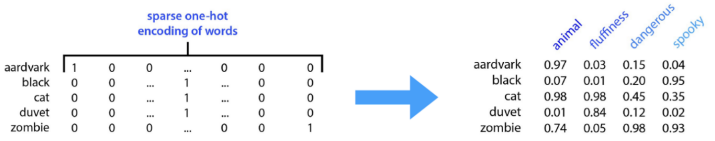
$$R^2 = 1 - \frac{RSS}{TSS}$$

Adjusted R^2-score

$$R^2_{adj} = 1 - \frac{\frac{RSS}{n-p-1}}{\frac{TSS}{n-1}}$$

AIC(Akaike) and BIC(Bayesian) Information Criteria, $\mathcal{L}$: Likelihood

$$AIC = -2\log(\mathcal{L}) + 2p \qquad BIC = -2\log(\mathcal{L}) + 2\log(n) \cdot p$$



7.4 Text data/Natural Lannguage Processing (NLP)

Text analysis is a major application field for ML algorithms. To feed text to a algorithms it first has to be transformd to a numerical vector by:

- **Tokenizing:** strings and give an integer ID to each possible token
- **Counting:** the occurences of token in each documnet
- **Normalizing:** and weighting with diminishing importance tokens that occur in the majority of samples/documents

7.4.1 Tokenization

1. All common seperators, operators, punctuations and non-printable characters are removed(using Regex).
2. Then, stop-words filtering that aims to filter-out the most frequent words performed
3. Finally, stemming and/or lemmatization is applied to obtain the stem of a word that is morphological root by removing the suffixes that present grammatical or lexical infromation about the word.

Porter Stemming: Simple replacement rules to create word roots.
Lemmatization: seeks to find actual roots for words.(e.g., to-me → book)

7.4.2 Word Embeddings

Idea: Map one-hot encoding to dense vectors

- Each word is represented using a low-dimensional, dense, real-valued vector that encodes the meaning of the word such that the words that are closer in the vector space are expected to be similar in meaning
- How to compute them? Can be pre-trained using a very general task on a very large corpus or learnt with the target task

This is called Word to vector(Word2Vec). There are 2 variants:

- continuous bad of words(CBOW)
- Skip-Gram

7.5 Audio data

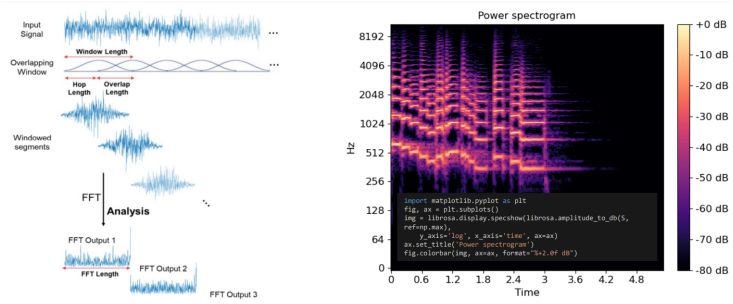
In the time domain the resulting amplitudes, time delays (echos) are captured. The fourier transform enables to decompose a signal into its individual frequencies an their amplitudes and phases.

7.5.1 Linear predictive coding coefficients (LPC)

LPC are parameters derived from audio signals that effectively model the spectral envelope of a sound wave. They capture the articulatory constraints of the vocal tract.

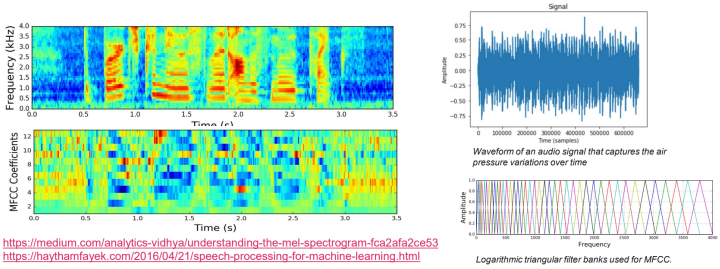
7.5.2 Spectogramm - Short Time Fourier Transform

In many cases we have non periodic signals: frequency content varies over time. Compute FFT on overlapping windowed segments of the signal to obtain the spectogramm.



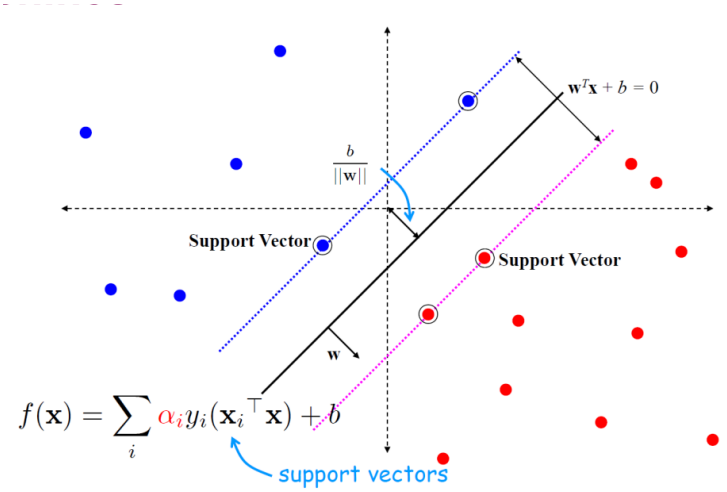
7.5.3 Mel Frequency Cepstral Coefficients (MFCC)

MFCC trys to mimic human hearing. It condeses audio information into fewer coefficients, simplifying data without losing critical information. They are noise robuster than raw spectral features.



https://medium.com/analytics-vidhya/understanding-the-mel-spectrogram-fca2afa2ce53
https://haythamfayek.com/2016/04/21/speech-processing-for-machine-learning.html

8 Support Vector Machines



8.1 Hyperplanes

Hyperplabes can be modified as:

$$H_{w,b} = \{\vec{x} | \vec{w} \cdot \vec{x} + b = 0\}$$

A projection of a point $x \in \mathbb{R}^p$ onto H_w, i.e., the closest point on H_w to x given by:

$$\pi_w(x) = x - \frac{\langle w, x \rangle + b}{\langle w, w \rangle} \cdot w$$

A hyperplane H_w is called separating, if

$$\hat{\gamma}_i = y_i \cdot h(x_i) = y_i \cdot (\langle w, x_i \rangle + b) > 0 \qquad (i = 1 \dots n)$$

8.2 maximal margin classifier

The data is called linear separable if there exists a separating hyperplane. In general, if there is one, there are many. We can define our classifier as:

$$f_{w,b}(x) = \text{sign}(\langle w, x \rangle + b)$$

Problem: Not invariant of rescaling. We need some sort of normalization.
We define the geometric margin γ_i of (w, b) with respect to a training sample (x_i, y_i) as:

$$\gamma_i = y_i \cdot \left(\frac{\langle w, x_i \rangle + b}{||w||} \right) = \frac{\hat{\gamma}_i}{||w||} = y_i d(x_i)$$

The geometric margin with respect of a training set $\mathcal{S}$ is smallest of these values in the training set:

$$\gamma = \min_{i=1 \dots n} \gamma_i$$

The geometric margin is invariant to rescaling of the parameters. We want to find a classifier that maximizes the geometric margin for linearly seperable dataset.
We will introduce the scaling constraint that the functional margin $\hat{y}$ of (w, b) with respect to the training set S must be 1: $\hat{\gamma} = 1$

$$\min_{w,b} \left\{ \frac{1}{2} ||w||_2^2 \right\}$$
$$\text{s.t.} \qquad y_i(\langle w, x_i \rangle + b) \geq 1$$

This is a convex optimization problem that could be solved using a Quadratic Programming (QP) code.

8.3 Support Vector Classifier SVC

The hyperplane is only dependent on the vectors closest to it. The closest vectors are called support vectors after their function.
SVM loss function in dual form:

$$-\frac{1}{2} \sum_{i=1}^{n} \sum_{j=1}^{n} \alpha_i \alpha_j y_i y_j \langle \phi(x_i), \phi(x_j) \rangle + \sum_{i=1}^{n} \alpha_i$$

The Representor Theorem states that the solution w^* of the optimization problem can always be written as a linear combination of the training (or ϕ-transformed training) data:

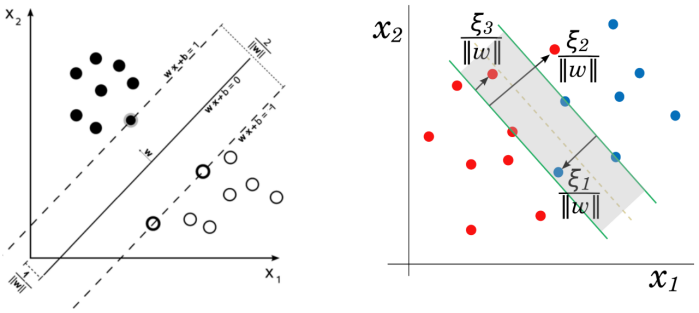
$$w^* = \sum_{i=1}^{N} \alpha_i y_i x_i \qquad w^* = \sum_{i=1}^{N} \alpha_i y_i \phi(x_i)$$

8.3.1 Handling data that is not linearly separable

1. Apply a feature transform $\phi(x)$
2. Introduce slack variables ξ(tolerate some missclassification)

By increasing the dimensionality of the feature space using mapping ϕ, we can find linearly separating hyperplanes.

8.3.2 Soft margin solution



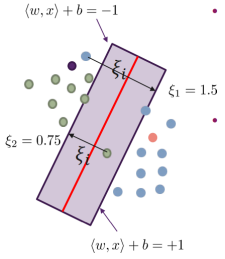
Mathematically, we formulate the trade-off by slack-variables $\xi_i > 0$.

$$\min_{w,b} \left\{ \frac{1}{2} ||w||_2^2 + C \sum_i \xi_i \right\}$$
$$\text{s.t.} \qquad y_i(\langle w, \phi(x_i) \rangle + 1) \geq 1 - \xi_i$$

We can fulfill every constraint by choosing ξ_i large enough. The larger ξ_i the larger the objective (that we try minimize). C is a regulariztion/trade-off parameter:

- small C: allow constraints to be easily ignored. Nearly no penalty for misclassification, large margin
- large C: makes constraints hard to ignore ! smaller margin, strong penalty for misclassification
- C = ∞: recovers a hard margin: all constraints must be fulfilled.

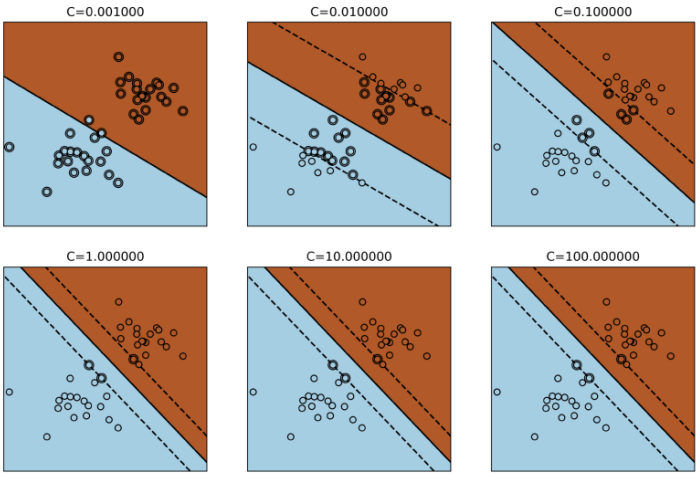
Non separable case example



$$y_1(\langle w, x_1 \rangle + b) \geq 1 - 1.5 = 0.5$$

$$y_2(\langle w, x_2 \rangle + b) \geq 1 - 0.75 = 0.25$$

$$\sum_i \xi_i = 1.5 + 0.75 = 2.25$$



Influence of the regulariztion parameter C

8.4 Support Vector Machine SVM

8.4.1 The Kernel Trick

Because of the Kernel Trick, a fir on kernel-transformed data can be done implicitly - that is, without ever building the full N-dimensional representation of the kernel projection. This kernel trick is built into the SVM, and is one of the reasons the method is so powerful.

8.4.2 Kernel Functions

Note that $\phi(x)$ only occures in pairs: $\langle \phi(x_i), \phi(x_j) \rangle = \phi^T(x_i) \cdot \phi(x_j)$
The complexity of learning depends on N(Data points), typically it is $O(N^3)$, not on D(Dimensions of Kernel space)

$$f(x) = \sum_{i=1}^{N} \alpha_i y_i K(x_i, x) + b$$

α: weights, might be zero
x_i: Support Vector(only few)
Kernel Function:

$$K_{ij} = K(x_i, x_j) = \langle \phi(x_i), \phi(x_j) \rangle = \langle z_i, z_j \rangle$$

K_{ij} is calculated fast, $z_i = \phi(x_i)$ is hard to calculate. Leads to the same optimization problem as the original dual optimization problem.

8.4.3 Primal Form of the SVM

$$\min_{\mathbf{w},b,\xi} \frac{1}{2}||w||_2^2 + C\sum_{i=1}^{n} \xi_i$$

$$y_i(\mathbf{w} \cdot x_i + b) \geq 1 - \xi_i, \quad \forall i = 1, 2, \ldots, n$$

$$\xi_i \geq 0, \quad \forall i = 1, 2, \cdot, n$$

The second term,$C\sum \xi_i$, incorporates the slack variable ξ_i and the regularization parameter C. C is a hyperparameter that controls the trade-off between maximizing the margin and minimizing the slack variables. A larger C encourages a narrower margin but penalizes misclassifications more severly.

8.4.4 Kernel Trick - Summary

- Classifiers can be learnt for high dimensional feature spaces, without actually having to map the points into high dimensional space
- Data may be linear separable in the high dimensional space, but not linearly separable in the original feature space
- Kernels can be used for an SVM because of the scalar product in the dual form, but can also be used elsewhere - they are not tied to the SVM formalism
- Kernels apply also to objects that are not vectors, e.g. to objects like histograms.

8.5 Most important information

- maximal margin classifier: Linearly seperable Data
- SVC: almost linearly seperable data, uses margins
- SVM: not linearly seperable data, uses Kernels to sperate data in higher dimensional space

Mercer's theorem

If $K(x, y)$ statisfies Mercer's conditions, it defines a inner product in a higher-dimensional space. This allows implicit mapping into higher dimensions without explicitly computing the transformation. K must be positive semi-definite(eingenvalus ≥ 0).

9 Gaussian Processes

9.1 The Bayes optimal classifier

Theoretically optimal (= most probable) classification. Combine predictions of all hypotheses, weighted by their posterior probabilities: (where y_i is a label from the set Y of classes)

$$P(Y = y_k | X_1, \ldots, X_n) = \frac{P(Y = y_k) \prod_i P(X_i | Y = y_k)}{\sum_j P(Y = y_j) \prod_i P(X_i | Y = y_j)}$$

Naive Bayes Classifier:

$$Y \leftarrow \arg\max_{y_k} P(Y = y_k) \prod_i P(X_i | Y = y_k)$$

9.2 Gaussian Distribution (1D)

Norma distribution with mean μ and variance σ^2:

$$p(x) = \mathcal{N}(x|\mu, \sigma^2) = \frac{1}{(2\pi\sigma^2)^{1/2}} \exp\left\{-\frac{1}{2\sigma^2}(x - \mu)^2\right\}$$

$$\mathbb{E}[x] = \mathbb{E}_{x\sim p}[x] = \int x \cdot p(x)\, dx = \mu$$

$$\mathbb{E}\left[x^2\right] = \mathbb{E}_{x\sim p}\left[x^2\right] = \int x^2 \cdot p(x)\, dx = \mu^2 + \sigma^2$$

$$\mathbf{VAR}[x] = \mathbb{E}\left[x^2\right] - \mathbb{E}[x]^2 = \sigma^2$$

9.2.1 Mulitvariante Gaussian Distribution

Gaussian defined over a vector x of continuous variables in a D-dimensional space with mean vector μ and covariance matrix Σ, where $|\Sigma|$ is the determinant of Σ.

$$p(\mathbf{x}) = \mathcal{N}(\mathbf{x}|\mu, \Sigma) = \frac{1}{(2\pi)^{D/2}|\Sigma|^{1/2}} \exp\left\{-\frac{1}{2}(\mathbf{x}-\mu)^T\Sigma^{-1}(\mathbf{x}-\mu)\right\}$$

The quadratic form in the argument of the exponential is called **Mahalanobis distance**:

$$\Delta = (\mathbf{x}-\mu)^T\Sigma^{-1}(\mathbf{x}-\mu)$$

9.3 Gaussian Processes

A Gaussian process $\mathcal{GP}$ is a stochastic process, i.e. a collection of random variables that are drawn from a infinite dimensional multivariate Gaussian distribution. A stochastic process is a collection of random variables, $\{h(x) : x \in \mathcal{X}\}$ indexed by elements from some set $\mathcal{X}$, known as the index set.

9.3.1 Definition of a Gaussian Process$\mathcal{GP}$

A Gaussian process is a stochastic process such that any finite subcollection of random variables has a multivariate Gaussian distribution(mean function $m(\cdot)$ and convariance function $k(\cdot,\cdot)$)

$$\begin{bmatrix} h(x_1) \\ \vdots \\ h(x_m) \end{bmatrix} \sim \mathcal{N}\left(\begin{bmatrix} m(x_1) \\ \vdots \\ m(x_m) \end{bmatrix}, \begin{bmatrix} k(x_1,x_1) & \cdots & k(x_1,x_m) \\ \vdots & \ddots & \vdots \\ k(x_m,x_1) & \cdots & k(x_m,x_m) \end{bmatrix}\right)$$

9.3.2 Example

One dimensional Gaussian process:

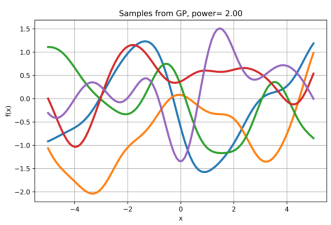
$$p(f(x)) \sim \mathcal{GP}\left\{m(x) = 0, k(x,x') = \exp\left(-\frac{1}{2}(x-x')^2\right)\right\}$$

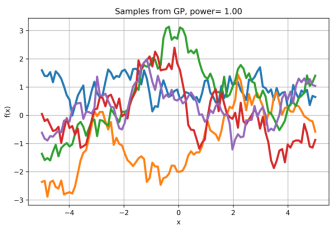
To get an indication of what this distribution over functions looks like, focus on a finite subset of function values $\mathbf{f} = (f(x_1), f(x_2), \ldots, f(x_n))$ for which:

$$\mathbf{f} \sim \mathcal{N}(0, \Sigma)$$

$$\Sigma_{ij} = k(x_i, x_j)$$

Then plot the coordinates of $\mathbf{f}$ as a function of the rorresponfing x values





9.4 Covariance Function$k(x,x')$ =Merver kernel

Gaussian processes are kernel-based probability distribution in the sense that any valid kernel function can be used as a covariance function. $k(x,x')$ is a kernel, if there is a feature mapping $\phi(x)$ into some (higher dimensional) space $\mathcal{H}$, where it can be represented by a scalar (dot) product.
Some possible Kernels:

- Radial Basis Function kernel (RBF)

$$k(x,x') = \sigma_0^2 \exp\left[-\frac{1}{2}(\frac{x-x'}{\lambda})^2\right]$$

- Rational-Quadratic kernel

$$k(x,x') = \left(1 + \frac{(x-x')^2}{2\alpha\lambda^2}\right)^{-\alpha}$$

- Exp-Sine-Squared kernel

$$k(x,x') = \exp\left[-\frac{2\cdot\sin^2(\pi/p\cdot|x-x'|)}{\lambda^2}\right]$$

- Dot-Product kernel

$$k(x,x') = \sigma_0^2 + x\cdot x'$$

9.5 Important information

- MAP assignment: The Most porbable A Posteriori (MAP) setting is thta which maximises the posteriori. It is the most probable set of parameters θ given the Data $\mathcal{D}$ and model class $\mathcal{M}$

$$\hat{\theta}_{\text{MAP}} = \arg\max_\theta\{p(\theta|\mathcal{D},\mathcal{M})\} = \arg\max_\theta\{p(\theta,\mathcal{D}|\mathcal{M})\}$$

- Max. Likelihood assignment: when $p(\theta|M)$ =const. It is the estimate for θ that maximises the probability of observing the data $\mathcal{D}$ given the model M.

$$\hat{\theta}_{\text{ML}} = \arg\max_\theta\{p(\mathcal{D}|\theta,\mathcal{M})\}$$

Matrix inverse:

$$A = \begin{bmatrix} a & b \\ c & d \end{bmatrix} \qquad A^{-1} = \frac{1}{\det(A)}\begin{bmatrix} d & -b \\ -c & a \end{bmatrix}$$

10 Dimensionality Reduction

10.1 The Curse of Dimensionality

One of the characteristic of high-dimensional data is that the number of dimensions is comparable or larger than the number of samples. High dimensional data is very sparse

10.2 The Manifold Hypothesis

A collection of methodologies for analyzing high dimensional data based on the hypothesis that data tend to lie near a low dimensional manifold is now called Mainfold learning.

Dim d	n for 10% coverage
1	10
2	100
3	1000
10	10^10
d	10^d

10.3 Principal Component Analysis (PCA)

The main linear technique for dimensionality reduction, mapping the data in such a way that the variance of the data in the low-dimensional representation is maximized.

10.3.1 Eigenvalues λ_k of C are the variance in the new space

We are given $i = 1 \dots n$ samples of dimension$(1 \times d)$ We constuct the data matrix $n \times d$: $X_{ik} = [x_{ik}]$
We diagonalize the covariance matrix C:

$$C = \frac{1}{n-1}(X - \overline{X})^T(X - \overline{X})$$

if $\overline{X} = 0$:

$$C = \frac{1}{n-1}X^TX$$

$$C = UDU^T = U\left[\text{diag}(\lambda_k)\right]U^T$$

10.3.2 Explained variance

No data is lost in transformation. The first component explains most of the variance, the second the second most,... All components explain the whole variance. Now take only a few components (the first k) and hope that the data is explained by them. How good is the approximation? A measure is the **explained variance ratio**:

$$P_k = \frac{\sum_{j=1}^{k}\mathbf{VAR}(Z:,j)}{\sum_{j=1}^{d}\mathbf{VAR}(Z:,j)} = \frac{\sum_{j=1}^{k}\lambda_j}{\sum_{j=1}^{d}\lambda_j}$$

10.3.3 Eigenvalue problem

$$\mathbf{CW} - \lambda\mathbf{w} = 0$$

$$(\mathbf{C} - \lambda \cdot \mathbf{I})\mathbf{w} = 0$$

$$\det(\mathbf{C} - \lambda\mathbf{I}) = 0$$

10.4 Mainfold Methods based on similarity

10.4.1 Example of metrics

L_p metric: $d_p(x,y) = ||x-y||_p = \sqrt[p]{\sum_i |x_i - y_i|^p}$
L_∞ metric: $||x||_\infty = \max_i \{|x_i|\}$
L_1 metric: $d_1(x,y) = ||x-y||_1 = \sum_i |x_i - y_i$ (manhatten distance)

10.4.2 MDS: Multidimensional scaling

MDS attemps to model similarity or dissimilarity data as distance in geometric spaces. In general, is a technique used for analyzing similarity or dissimilarity data. MDS attemps to find an embedding from the high dimensional objects in I into $y_i \in \mathbb{R}^d$ such that distances d_{ij} are preserved.

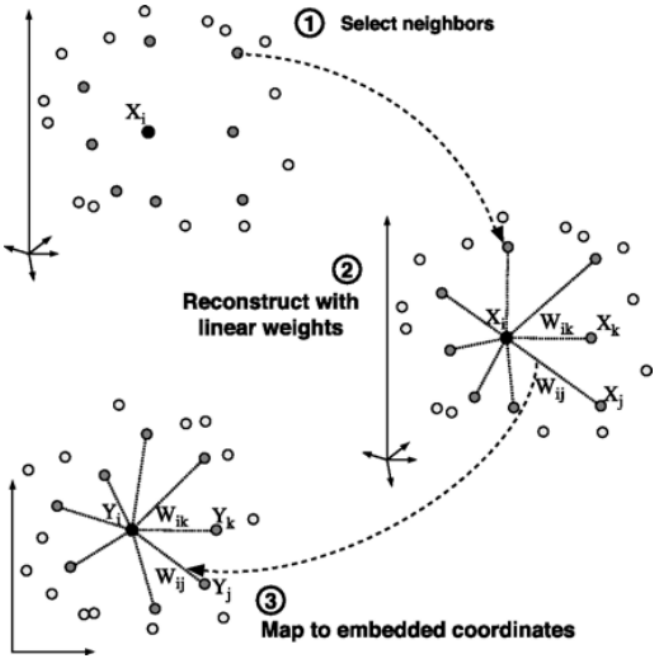
10.4.3 LLE: local linear embedding

LLE discribes the local properties of the manifold around a data point x_i by writing the data point as a linear combination (the so-called reconstruction weights w_{ij}) of its k nearest neighbor:

$$x_i \approx \sum_{j=1}^{k} w_{ij}x_j$$

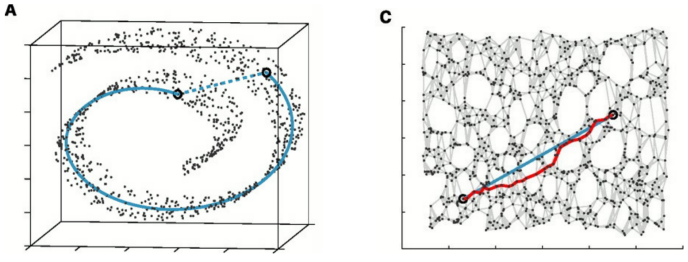
In the low dimension, LLE attemps to retain the reconstruction weights W as well as possible. Hence, LLE fits a hyperplane through the data point x_i and its nearest neighbors, thereby assuming that the mainfold is locally linear. Finding the low d-dimensional data representation Y amounts to minimizing the cost function $\mathcal{L}$:

$$\mathcal{L}(w,Y) = \sum_{i=1}^{N}||y_i - \sum_{j=1}^{k} w_{ij}y_j||^2$$



10.4.4 Isomap: Isometric mapping

A non-linear method for dimensionality reduction. Finds the map that preserves the global, non-linear geometry of the data by preserving the geodesic manifold interpoint distances. Geodesic: Shortest curve along the manifold connecting two points.



The basic steps are:

1. For each object, find a small set of neighboring objects and their distance.
2. Compute all-pairs shortest paths on the above neighborhood graph.
3. Run multidimensional scaling using the matrix of shortest-path distances.

Advantages:

- Nonlinear
- Non-iterative
- Globally optimal
- Parameters: k or η(chosen fixed radius)

Disadvantages:

- Graph disctreteness overestimates the geodesic distance
- k must be high to avoid linear shortcuts near regions of high surface curvature

10.4.5 t-SNE:t-distributed stochastic neighbor embedding

t-SNE is a manifold learning algorithem that constructs a probability distribution p over the dataset X, and then another probability distribution q in a lower dimensional data space Y, making both as close as possible
Baisc Steps:

1. In the high dimensional space X, create a probability distribution $p_{i|j}$ that dictates the relationships between various neighboring points x_i and x_j.
2. It then tries to recreate a probability distribution $q_{i|j}$ in a low dimensional space Y that follows that probability distribution $p_{i|j}$ as best as possible (KL-divergence as loss function).

Math

$$p_{i|j} = \frac{\exp\left(-\frac{||x_i - x_j||^2}{2\sigma_i^2}\right)}{\sum_{k \neq i} \exp\left(-\frac{||x_i - x_k||^2}{2\sigma_i^2}\right)}$$

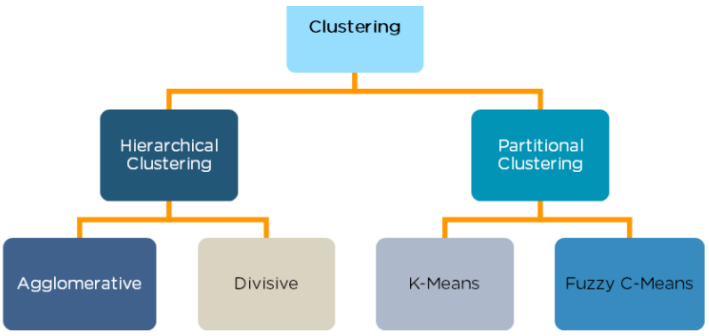
$$q_{i|j} = \frac{e^{-||y_i - y_j||^2}}{\sum_{k \neq i} e^{-||y_i - y_k||^2}}$$

The Kullback-Leibler (KL) divergence is a measure of how different two probability distributions are from one another (also called relative entropy):

$$KL(P|Q) = \sum_{i \neq q} p_{i|j} \ln\left(\frac{p_{i|j}}{q_{i|j}}\right)$$

11 Cluster Analysis

Clustering = Finding groups in data
Problem: given n data points, separate them into K clusters.



n	number of data points
K	number of clusters ($K << n$)
Δ	a partition, $\Delta = \{C_1, C_2, \ldots, C_K\}$
$\mathcal{L}(\Delta)$	loss of Δ to be minimized

11.1 Taxonomy of Clustering

Hard clustering: Each data point is assigned a unique cluster:Δ
Soft clustering: Each data point i is assigned a probability that it is in cluster k:
Parametric clustering: k known
Non-parametric: k determined by algorithm

11.2 Hierarchical Clustering

Hierarchical Clustering (HCA) seeks to build a hierarchy of clusters. Strategies for hierarchical clustering generally fall into two types:

- Agglomerative: This is a bottom-up approach: each observation starts in its own cluster, and pairs of clusters are merged as one moves up their hierarchy.
- Divisive: This is a top-down approach: all observations start in one cluster, and splits are performed recursively as one moves down the hierarchy.

We start with N datapoints that initially form N clusters. The two clusters with the smallest linkage are fused togther to form $N-1$ clusters. This is repeated until there is only one single cluster.

$$(i,j)_{fused} = \arg\min_{i \neq j}\{D_{link}(C_i, C_j)\}$$

11.2.1 Linkage criteria

The linkage citerion determines together with a metric $d(x,y)$ when two clusters A and B should be merged together in hierarchical clustering (fusion criterium)
where c_s and c_t are the centroids of clusters s and t, respectively.

11.2.2 Basic Algorithm

Input:

- Distance matrix D data points (size $n \times n$)

Maximum(complete)	$\max\{d(a,b) : a \in A, b \in B\}$
Minimum(single)	$\min\{d(a,b) : a \in A, b \in B\}$
Mean(average)	$\frac{1}{\lvert A\rvert \cdot \lvert B\rvert} \sum_{a \in A} \sum_{b \in B} d(a,b)$
Centroid	$\lvert\lvert c_s - c_t\rvert\rvert$

d_{ij}	c_1 o_1	c_2 o_2	c_3 o_3	c_4 o_4
o_1	0			
o_2	4	0		
o_3	7	5	0	
o_4	8	10	9	0

d_{ij}	c_1 o_1, o_2	c_2' o_3	c_3' o_4
o_{12}	0		
o_3	7 | 5	0	
o_4	8 | 10	9	0

Single-Linkage D_s

d_{ij}	c_1	c_2	c_3
o_{12}	0		
o_3	5	0	
o_4	8	9	0

Complete-Linkage D_c

d_{ij}	c_1	c_2	c_3
o_{12}	0		
o_3	7	0	
o_4	10	9	0

Average-Linkage D_{avg}

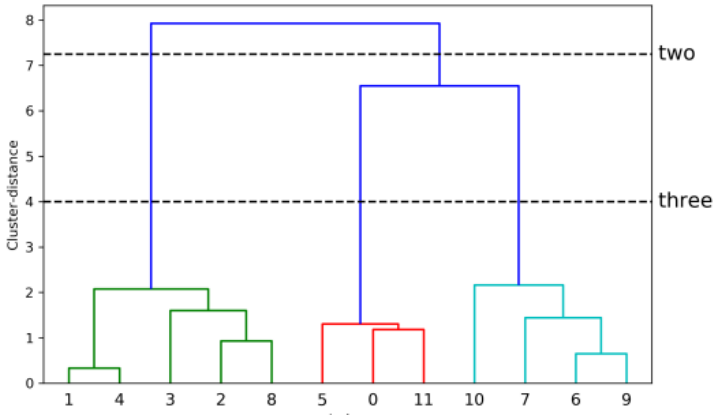
d_{ij}	c_1	c_2	c_3
o_{12}	0		
o_3	6	0	
o_4	9	9	0

- fucntion $d(a, b)$ to compute a distance between clusters (usually takes D as input)

Initialization: Clustering $\mathcal{C}^{(0)} = \{C_1^{(0)}, C_2^{(0)}, \ldots, C_n^{(0)}\} = \{i\}$
While the current number of clusters is > 1:

- find the two clusters which have the smallest linkage to each other
- merge them to one cluster

Ouput: Resulting Dendrogram: The Dendrogram is a tree that represent the hierarchical division of the data set O into ever smaller subsets.
Dendrograms cannot tell you how many clusters you should have: Interpretation of this kind is justified only when the ultrametric tree inequality holds, which is very rare.



11.3 K-means

The standard algorithem is non-probabilistic EM. The problem is that k-means is very sensitive to the choice of k, even with correct k it may converge to wrong local Minimum.

11.3.1 K-means algorithm

Input: Data $\mathcal{D} = \{x_i\}_{i=1:N}$, number of clusters K
Initialize: centers $\mu_1, \mu_2, \ldots, \mu_k \in \mathbb{R}^d$ at random
Iterate until convergence:

1. for $i = 1 : n$: $k(i) = \arg\min_k ||x_i - \mu_i||$(assign points to cluster $\rightarrow$ new clustering)
2. for $k = 1 : K$: $\mu_k = \frac{1}{|C_k|} \sum_{i \in C_k} x_i$ (recalcuate centers)

Convergence: if Δ does not change after iteration m, it will never change after that.
How to pick starting points?:

- Random initialization: Randomly choose some data points as starting centers.
- Forgy method: Chooses k observations from the dataset and uses these as the initial means.
- Random Partition: Randomly assignes a cluster to each observation an then proceeds to the update step, thus computing the initial mean to be the centroid of the clusters randomly assigned points.
- K-means++: The algorithm is guaranteed to find a solution that is $O(\log k)$ competitive to the optimal k-means solution.

11.3.2 K-means cost function

The distortion(least-squares) can also be expressed as sum of (squared) intracluster distances:

$$\mathcal{L}(\Delta) = \frac{1}{2} \sum_{k=1}^{K} \sum_{i \in C_k} ||x_i - x_j||^2 + \text{const}$$

11.3.3 Limitations of K-means: Non-globular shapes

1. Inertia W makes the assumption that clusters are convex and isotropic.
2. Inertia W is not a normalized metric.

11.3.4 More variants of K-means

- K-medians
- weighted K-means: weighted data points
- kernel-k-means
- soft K-means: soft assignments (in form of probability)

11.3.5 How shall we choose k

- Basic Elbow method
- Try a range of K values and plot average distance to centers
- Silhoutte
- Cross-Validation
- Information theoretic perspective: NMI,BIC and AIC

Silhoutte Score

A graphical method to select k. Given K and K clusters, given any data point i, let a_i be the average distance or dissimilarity of i with all other points in the same cluster. a_i measures how well i fits into its own cluster. b_i is the smallest average distance(Euclidean distance) of i to other clusters.
Silhouette score $s_i \in [-1, 1]$:

$$s_i = \frac{b_i - a_i}{\max(b_i, a_i)}$$

$s_i \approx 1$: point i is in a tight cluster and far away from other clusters
$s_i \approx -1$: point i is in a loose cluster and close to other clusters
Maximize $\frac{1}{n}\sum_{i=1}^{n} s_i$ over k.

12 Gaussian Mixture Models and EM

12.1 Metrics to evaluate Clustering

Cluster Inertia W

The within-cluster inertia W of the partitionnC_k is the sum of the inertia of the clusters and measures then the heterogeneity within the clusters.

$$W = \sum_{k=1}^{K} I(C_k) \quad I(C_k) = \sum_{i \in C_k} ||x_i - \mu_k||^2$$

NMI: Normalized Mutual Information

Mutual information is a function that measures the agreement of the two assignments, ignoring permutations.

$$\text{NMI} = \frac{2 \cdot I(Y|C)}{H[Y] + H[C]} \quad I(Y|C) = H[Y] - \sum_{k} H[Y|C]$$

$I[Y|C]$: mutual information between Y and C
$H[Y]$: Entropy of class labels Y
$H[C]$: Entropy of class labels C

BIC: Bayesian Information Criterium

Can measure the efficiency of the parameterized model in terms of prediciting data. It penalizes the complexity of the model$B$.

$$BIC = -2\ln(\mathcal{L}) + B \cdot \ln(n) = W + k \cdot (d+1)\ln(n)$$

$\mathcal{L}$: the maximized value of the likelihood function of the model $\mathcal{M}$, $\mathcal{L} = p(x|\hat{\theta}, \mathcal{M})$ where $\hat{\theta}$ are the ML estimates parameters.
x: observed data (dimensions d)
n: number of data points
B: number of parameters estimated by the model $\mathcal{M}$: $B = K \cdot (d+1)$
W: within-cluster inertia W = RSS(Residual Sum of Squares)

12.2 Density based Clustering (DBSCAN)

DBSCAN = density-based spatial clustering of applications with noise.
The idea if DBSCAN is that clusters from dense regions in that data and are seperated by relatively empty areas.
Advantages of DBSCAN:

- The user can not set the number of clusters a priori
- DBSCAN is able to capture clusters with complex shapes
- if identifies points that do not belong to any of the clusters

The DBSCAN procedure is slower than the agglomerative clustering and k-Means, but scales relatively well for large data sets.

12.2.1 How DBSCAN works

2 Parameters: min_samples and eps.
If at least min_samples data points are within the distance eps to a given point, this data point is classified as a core object. Core objects that are closer than eps to each other are assigned to the same cluster.
At the beginning, the algorithm selects any starting point. Then it finds all points at distance eps or closer to this point. If less than min_samples points are found within the distance eps to the starting point, this point will be classified as noise(it does not belong to any cluster).
If there is more as min_sampels points at a distance of eps, the point is used as core object and receives a new cluster designation.

12.2.2 Advantages of DBSCAN

- The user can not set the number of clusters a priori
- DBSCAN is able to capture clusters with complex shapes
- It identifies points that do not belong to any of the clusters

12.3 Gaussian Mixture Models (GMM)

Mixture Models approximate an arbitray distribution by a linear combination of a simpler, well-behaved distribution.
The Gaussian Mixture Models (GMM):

- Modeled by a weighte sum of N multivariante Gaussians
- The Gaussians parameters can be estimated effeciently using the EM algorithm

$$p(x|\pi_i, \mu_i, \Sigma_i) \sim \sum_{i=1}^{K} \pi_i \mathcal{N}(\mu_i, \Sigma_i)$$

The weights sum up to one (probability to belong to cluster k)

$$\sum_{k=1}^{K} = 1$$

GMM are soft clustering.

12.3.1 Expectation Maximization(EM)

1. Start with a random initial hypothesis. Pretend to knwo the parameters μ, σ^2 of the 2 Gaussians (Place 2 Gaussians somewhere)
2. E-Step: Estimate expected values of variables, assuming the current hypothesis holds. Compute probabilities γ_{ik}.(which data point belongs to wich Gaussian)

3. M-Step: Calculate new Maximum likelihood (ML) estimate of hypothesis, assuming the expected values from step 2 hold. Calculate the μ_k, σ_k^2, givem the currently assigned membership.(Update Gaussians according to data points of class)

4. Repeat with step 2 until convergence

$$\mu_k = \sum_{i=1}^{K} \gamma_{ik} \cdot x_i \quad \sigma_k^2 = \sum_{i=1}^{N} \gamma_{ik} \cdot (x_i - \mu_k)^2$$

12.3.2 Custering with Gaussians

K-means is equivalent to assuming that the data came from K spherically symmetric Gaussians. Instead of isotropic covarinaces $I \cdot \sigma_k^2$, we know use full covarinace matricies Σ_k to model elliptical clusters.

$$p(\mathbf{X}|\mathbf{Z}, \mu, \mathbf{\Sigma}) = \prod_{n=1}^{N} \prod_{k=1}^{K} \left[\mathcal{N}(\mathbf{x}_n|\mu_k, \mathbf{\Sigma}_k)\right]^{z_{nk}}$$

12.3.3 EM-Algorithm for GMM

Compute the cluster assignments(E-steps, Bayes rule with $\pi_k^{(t)}$):

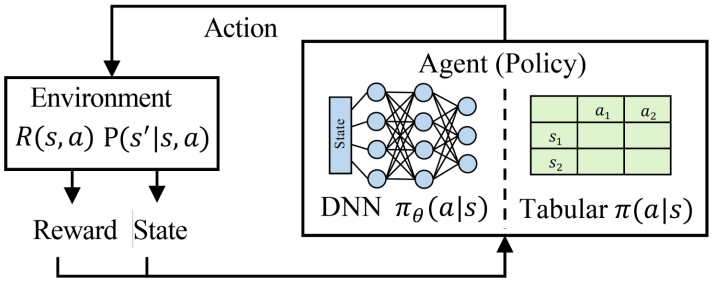
$$\gamma_{nk}^{(t)} = p\left(z_{nk} = k|\theta^{(t)}, \mathbf{x}_n\right) = \frac{\pi_k^{(t)} \mathcal{N}\left(\mathbf{x}_n|\mu_k^{(t)}, \mathbf{\Sigma}_k^{(t)}\right)}{\sum_{k=1}^{K} \pi_k^{(t)} \mathcal{N}\left(\mathbf{x}_n|\mu_k^{(t)}, \mathbf{\Sigma}_k^{(t)}\right)}$$

Update the cluster centers, covarinaces and probabilities(M-step, max.likelihood appr.):

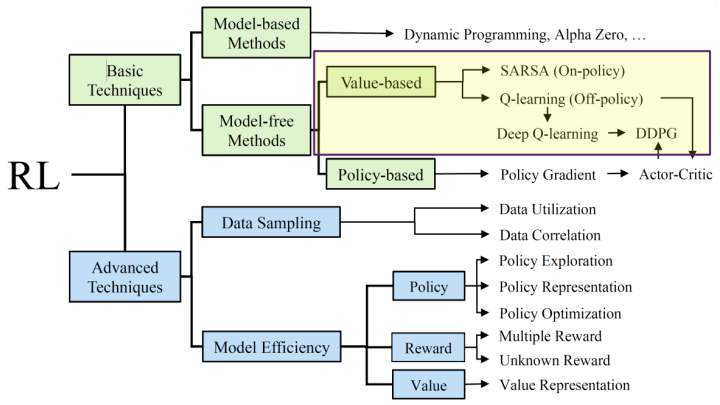
$$\mu_k^{(t+1)} = \frac{\sum_n \gamma_{nk}^{(t)} \mathbf{x}_n}{\sum_n \gamma_{nk}^{(t)}} = \frac{\sum_n \gamma_{nk}^{(t)} \mathbf{x}_n}{Z^{(t)}} \quad \pi_k^{(t+1)} = \frac{1}{N} \sum_n \gamma_{nk}^{(t)}$$

$$\mathbf{\Sigma}_k^{(t+1)} = \frac{1}{Z^{(t)}} \cdot \sum_N \gamma_{nk}^{(t)} \left(\mathbf{x}_n - \mu_k^{(t+1)}\right) \left(\mathbf{x}_n - \mu_k^{(t+1)}\right)^T$$

13 Reinforcement Learning



The agent and environment interact at each of a sequence of discrete time steps,$t = 0, 1, 2, \ldots$. At each time step t, the agent receives some representation of the environment's states,$s_i \in S$, where S is the set possible states. On that basis, the agent selects an action, $a \in A(s_t)$, where $A(s_t)$ is the set of actions available in state s_t. One time step later, in part as a consequence of its action, the agent receives a numerical reward,$R_{t+1} \in \mathbb{R}$, and finds itself in a new state, s_{t+1}.



13.0.1 Policy π

A policy π is a distribution over actions a given state s,i.e. it is the probability that the agent takes action $a_t \in A$ in state $s \in S$

$$\pi(a|s) = Pr\left[a_t = a|s_t = s\right]$$

13.0.2 Types of RL environments

- Deterministic environment
- Stochastic environment
- Fully observeable environment
- Partially observeable environment
- Discrete environment
- Continuous environment
- Episodic and non-episodic environment
- Single and multi-agent environment

13.0.3 Agent Categories

- Value Based: No policy, Value function
- Policy Based: Policy, No value function
- Actor Critic: Policy(for the actor), Value function (for the critic)
- Model Free: Policy and/or Value function, no (explicit) Model of the Environment, no explicit dynamics Model
- Model Based: Optionally Policy and/or Value function, (explicit) Model of the Environment, explicit dynamics Model

13.0.4 Markov Process

A Markov Process (or Markov Chain) is a tuple (S,P), where:

- S is a (finite) set of states
- P is a state transition probability matrix(Markov table): $P_{ss'} = Pr(s_{t+1}|s_t)$

The core concept is that the future only depends in the present and not on the past.

13.0.5 Markov Reward Process (MRP)

A Markov Reward Process is a tuple $(S, P, \mathcal{R}, \gamma)$:

- S is a (finite) set of states
- P is a state transition probability matrix:$P = P_{ss'} = Pr(s_{t+1}|s_t)$
- $\mathcal{R}$ is a reward function (expectation value of the next reward): $\mathcal{R}_s = \mathbb{E}[R_{t+1}|s_t]$
- γ is a discount factor: $\gamma \in [0, 1]$

13.1 Markov decision process (MDP = MRP + A)

A Markov Decision Process is a tuple $(S, A, P, \mathcal{R}, \gamma)$

- S is a (finite) set of states
- A is a (finite) set of actions
- P is a state transition probability matrix:$P = P_{ss'} = Pr(s_{t+1}|s_t)$
- $\mathcal{R}$ is a reward function (expectation value of the next reward): $\mathcal{R}_s = \mathbb{E}[R_{t+1}|s_t]$
- γ is a discount factor: $\gamma \in [0, 1]$

13.1.1 Return G_t

The Return G_t is the total discounted reward from time-step t on.

$$G_t = R_{t+1} + \gamma R_{t+2} + \cdots = \sum_{k=0}^{\infty} \gamma^k R_{t+k+1}$$

- γ close to 0 leads to myopic evaluation
- γ close to 1 leads to far-sighted evaluation

Recursion formula (basis for the Bellman equation):

$$G_t = R_{t+1} + \gamma G_{t+1}$$

13.1.2 Value Function $V(s)$ and Bellman Equation for MRP

The state-value-function $V(s)$ of an MRP is the expected cumulated return starting from state s;

$$V(s) = \mathbb{E}[G_t|s_t = s]$$

The state-value-function can be decomposed in two parts. This leads to the Bellman Equation for MRPs

$$V(s) = \mathcal{R}_s + \gamma \sum_{s' \in S} P_{ss'} V(s')$$

Bellman's principle of optimality: in many mathematical optimitzation problems, the optimum (global) solution is a sequence of optimum partial solutions.

13.1.3 State-action-value funtion $Q(s, a)$

The state-action-value function $Q(s, a)$ is a expected return starting from state s, taking action a, and then following policy π. It can be decomposed:

$$\begin{aligned} Q^\pi(s, a) &= \mathbb{E}_\pi[G_t|s, a] \\ &= \mathbb{E}_\pi[R_{t+1} + \gamma Q^\pi(s_{t+1}, a_{t+1})|s, a] \end{aligned}$$

The Bellmen Equation for $Q^\pi(s, a)$:

$$Q^\pi(a, s) = \mathcal{R}_s^a + \gamma \sum_{s' \in S} P_{ss'}^a \sum_{a' \in A} \pi(a'|s') \cdot Q^\pi(s', a')$$

13.1.4 Maximum value function: max.expected future reward

Goal of the MDP: the optimum policy

$$V^*(s) = \max_\pi \{V^\pi(s)\}$$

$$Q^*(s, a) = \max_\pi \{Q^\pi(s, a)\}$$

13.2 Temporal-Difference (TD) Learning

There are two concepts for solving Markov decision processes (MDPs).

- Monte Carlo(MC) techiques
- Dynamic Programming(DP)

Temporal-difference (TD) learning is a combination of these two approaches. It learns directly from experience by sampling, but also bootstraps. This represents a breakthrough in capability that allows agents to learn optimal strategies in any environment.

13.2.1 Updating the Value Function

Monte Carlo: The value function $V(s)$ is updated by averaging the returns observed from multiple episodes that pass through state s, where G_t is the return (cumulative discounted reward) following state s at time t and α is the learning rate.

$$V^{(t+1)}(s) = V^{(t)}(s) + \alpha \left[G_t - V^{(t)}(s) \right]$$

Bootstrapping methods, such as those used in Temporal-Difference (TD) learning, update the value function based on estimates rather than waiting for the final outcome. These methods can update the value function after each time step, leading to faster learning.
Bootstrapping TD-Learning:

$$V^{(t+1)}(s) = V^{(t)}(s) + \alpha \left[r + \gamma V^{(t)}(s') - V^{(t)}(s) \right]$$

Temporal-difference state-value function:

$$V_\pi(s) = \mathbb{E}[V_\pi(s) + \alpha(r + \gamma V_\pi(s') - V_p i(s))|s]$$

Online TD state-value estimate:

$$V_\pi(s) \leftarrow V_\pi(s) + \alpha[r + \gamma V_\pi(s')] = V_\pi(s) + \alpha[V_{target} - V_\pi(s)]$$