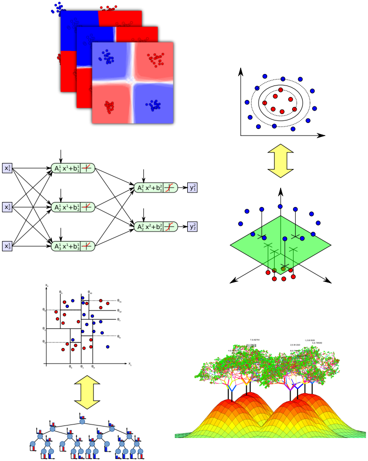
MVA - Discrete Inference and Learning
Lecture 6
-
Classical Learning

Yuliya Tarabalka
Inria Sophia Antipolis-Méditerranée - TITANE team
Université Côte d'Azur - France

Overview

1. Classification based on Features
   - Decision Boundary
   - Linear Decision Boundary
   - Non-linear Decision Boundary
2. Feature extraction
3. Machine Learning Methods
   - Support Vector Machine (SVM)
   - Multi-Layer Perceptron (MLP)
   - Random Forest (RF)

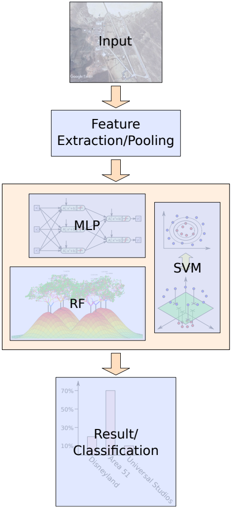


Overview

1. Classification based on Features
   - Decision Boundary
   - Linear Decision Boundary
   - Non-linear Decision Boundary
2. Feature extraction
3. Machine Learning Methods
   - Support Vector Machine (SVM)
   - Multi-Layer Perceptron (MLP)
   - Random Forest (RF)

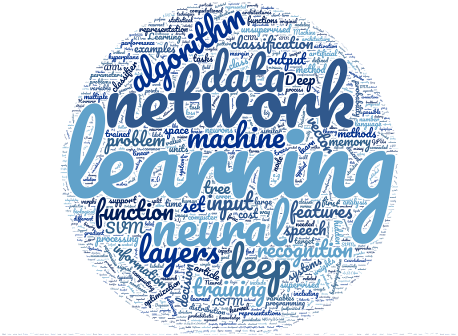
Classical Learning

- Features extract very basic, low level information
- We want very high level information (e.g. class of objects)
- Classical Learning: Learn the mapping between low level features and high level information



Classical Learning

- Machine Learning is a huge (growing) field
- Many different approaches for modeling/parametrizing this mapping!



Methods

- Choice of method not always rational
- Different pros/cons
- Speed, memory, scalability of training data, ease of implementation, ease of hyper parameter tuning, ...
- First intuitive understanding of the problems, then identifying methods

Decision based on features

Toy example

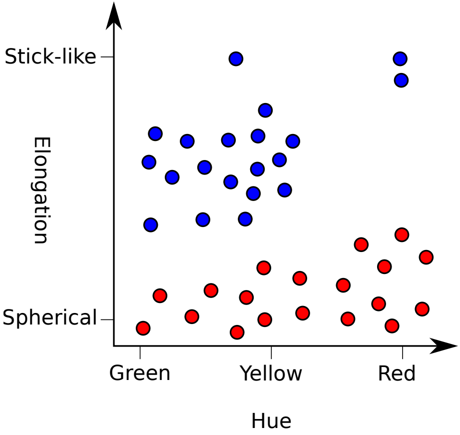
Task: Classify fruits into either bananas or apples

Extracted Feature Vector

- Hue (yellow to red)
- Elongation (max extend over min extend)

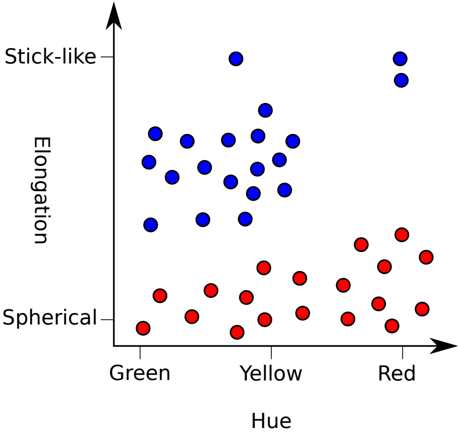
Some training data

- Feature space is just 2D
- Datapoints can be plotted as a scatter plot



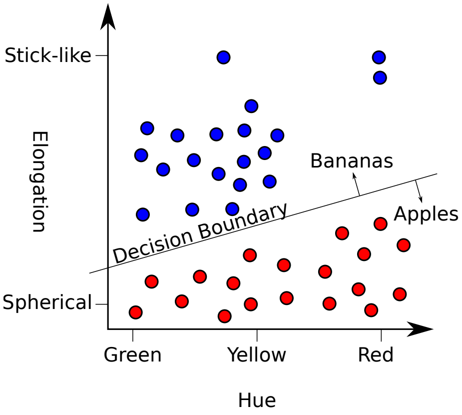
Some training data

- Feature space is just 2D
- Datapoints can be plotted as a scatter plot
- Can we "learn", which part of the feature space is bananas/apples?



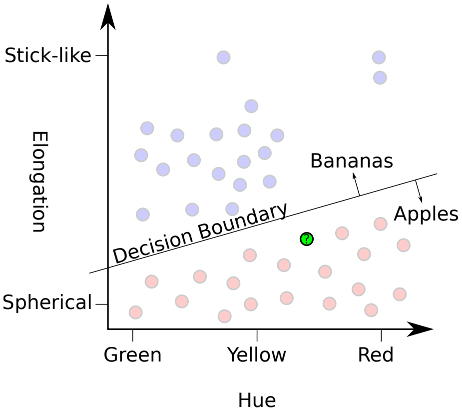
Decision boundary

- (Very) simple idea: Split the feature space into two half spaces



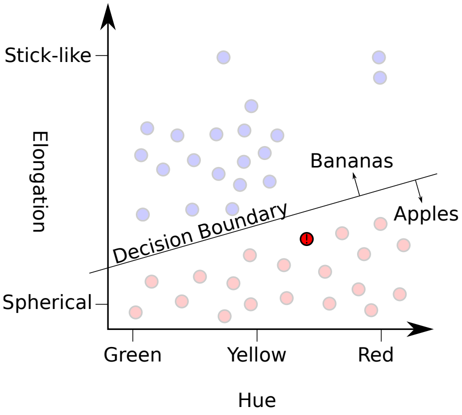
Decision boundary

- (Very) simple idea: Split the feature space into two half spaces
- During application, classify data based on this decision boundary



Decision boundary

- (Very) simple idea: Split the feature space into two half spaces
- During application, classify data based on this decision boundary

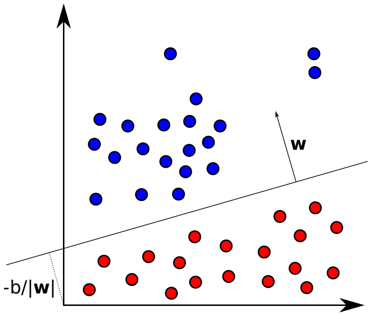


Perceptron

Perceptron

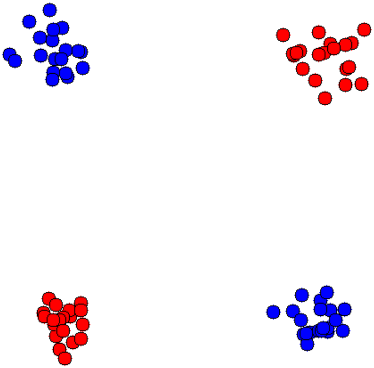
$$y = \text{sign}(\mathbf{w}^T \mathbf{x} + b) \qquad (1)$$

- $y \in \{-1, 1\}$: Predicted class
- $\mathbf{x} \in \mathbb{R}^2$: Feature vector
- $\mathbf{w} \in \mathbb{R}^2$: "Weight vector" (needs to be learned)
- $b \in \mathbb{R}$: "Bias" (needs to be learned)



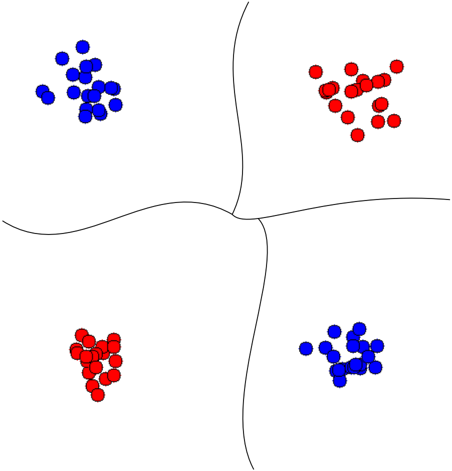
Linear Separability

- What if no such line exists?
- Quite often, problem not linearly separable
- Needs non-linear decision boundary



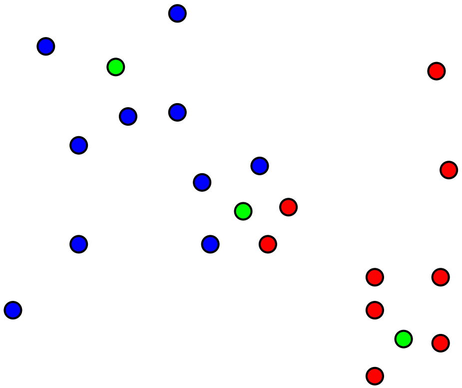
Non-linear Decision Boundary

- Decision boundaries of more complex ML techniques usually non-linear
- Regions need not be connected



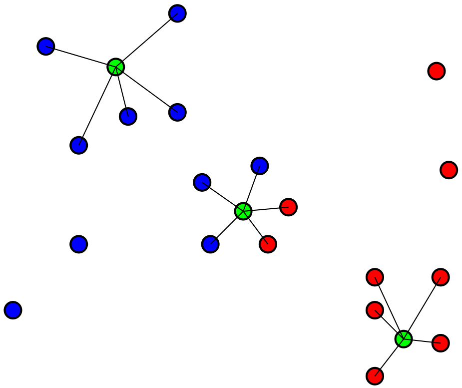
kNN

- Very simple idea:
  k-Nearest-Neighbors for
  classification



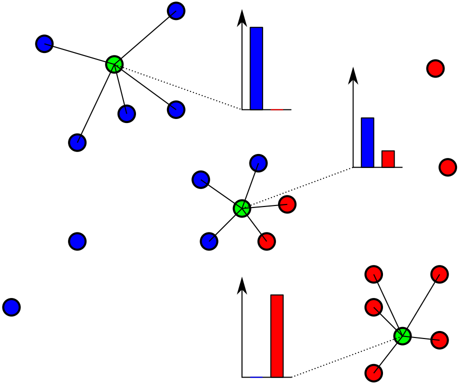
kNN

- Very simple idea:
  k-Nearest-Neighbors for classification
- For a sample find the k (e.g. 5) closest data points in the training dataset



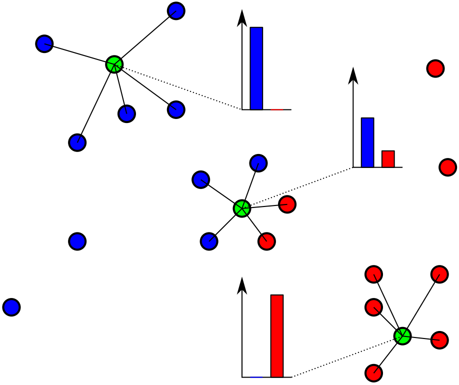
kNN

- Very simple idea:
  k-Nearest-Neighbors for classification
- For a sample find the k (e.g. 5) closest data points in the training dataset
- Look at the labels of those neighbors

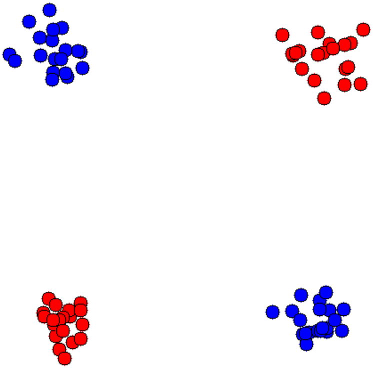


kNN

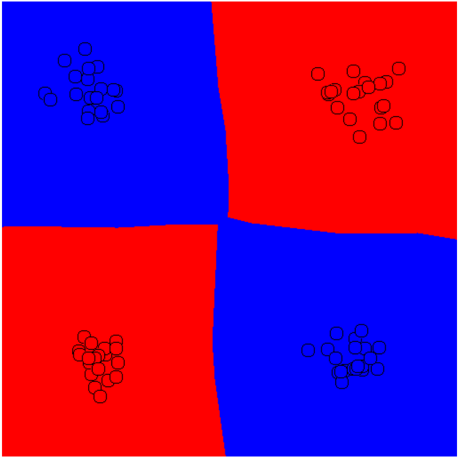
- Very simple idea: k-Nearest-Neighbors for classification
- For a sample find the k (e.g. 5) closest data points in the training dataset
- Look at the labels of those neighbors
- Fast lookup through trees/approximate methods
- Needs to keep all training data around



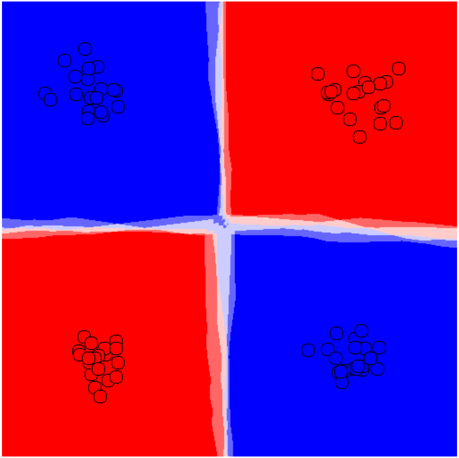
kNN Example - Simple



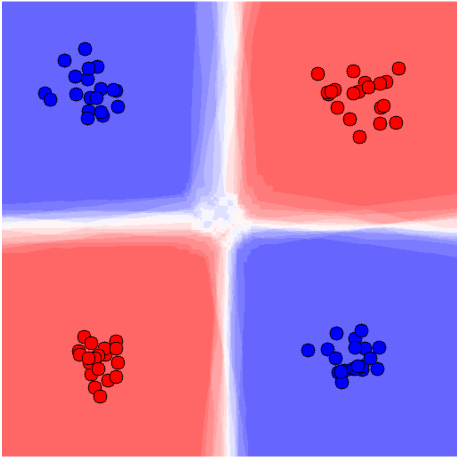
kNN Example - Simple - kNN K=1



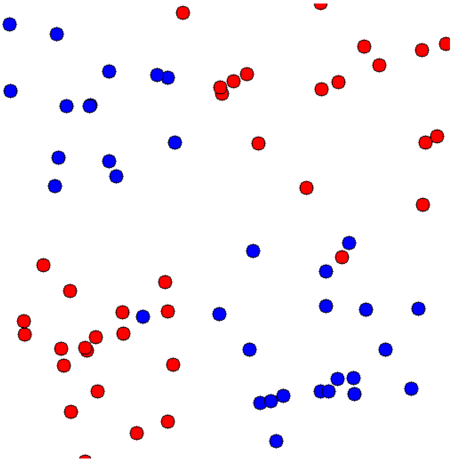
kNN Example - Simple - kNN K=5



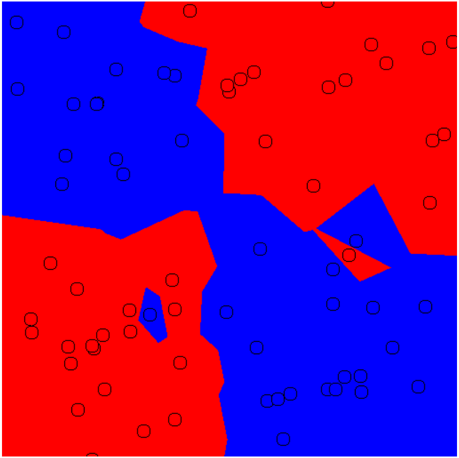
kNN Example - Simple - kNN K=25



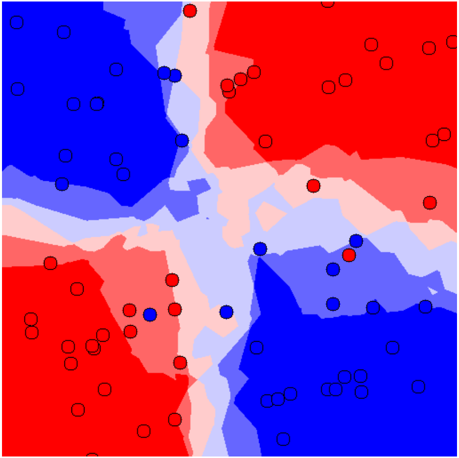
kNN Example - Hard



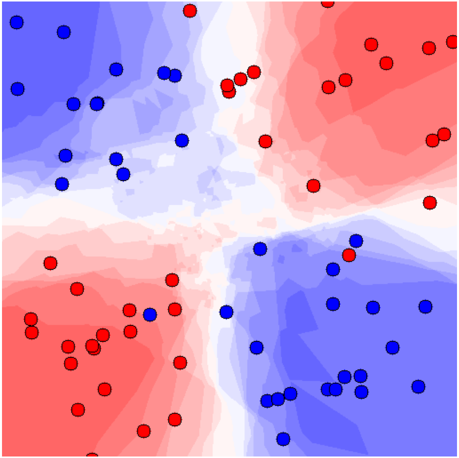
kNN Example - Hard - kNN K=1



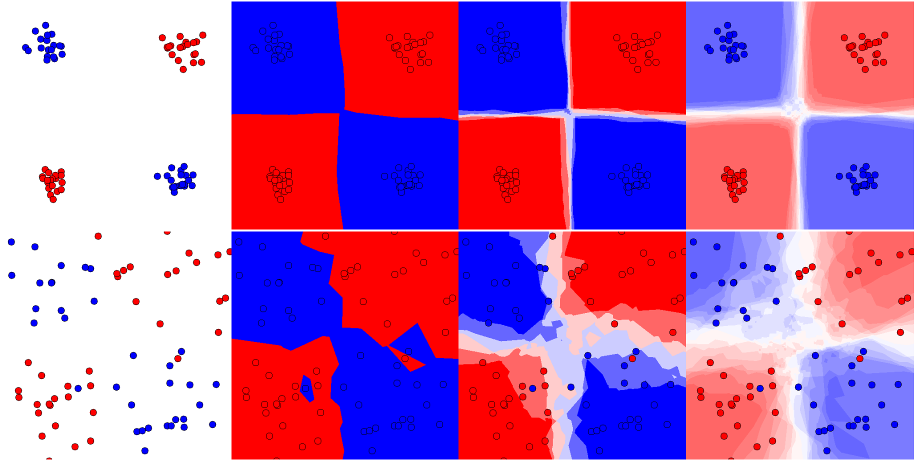
kNN Example - Hard - kNN K=5



kNN Example - Hard - kNN K=25

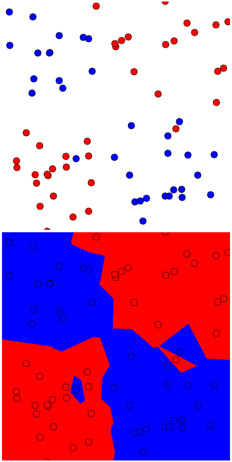


kNN Example

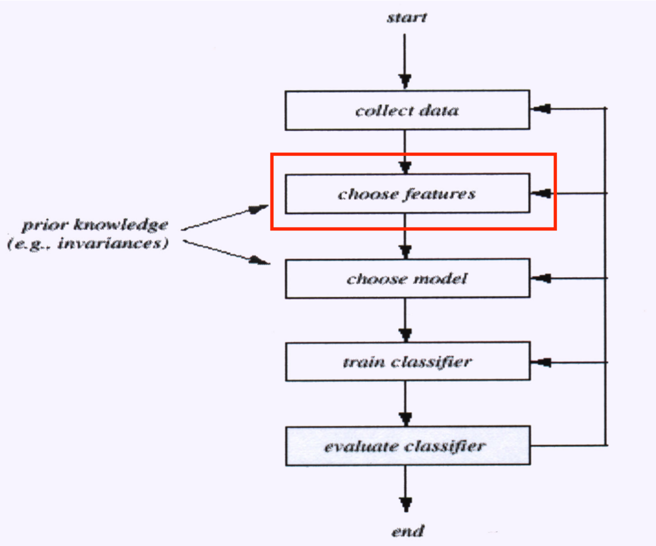


Model Complexity vs Overfitting

- With sufficient model complexity, it is often easy to get ZERO training error
- Generalization is what matters!
- **Test on data not used during training**
  - Disjoint train and test set
  - Non-overlapping samples if spatial features are used
  - Semi-manual parameter tuning (grid-search, etc.) needs third independent data set



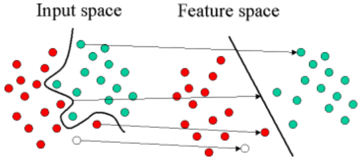
Why do we need feature extraction?



Motivation

- **Main motivation:** get out most of the data

- For classification task: find a space where samples from different classes are well separable

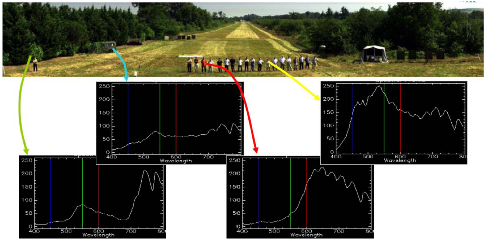


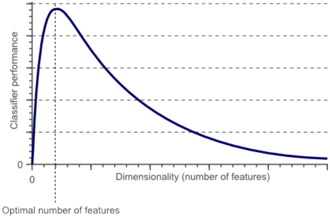
Objectives:

- Reduce computational load of the classifier
- Increase data consistency
- Incorporate different sources of information into a feature vector: spectral, spatial, multisource, ...

Motivation - Curse of dimensionality

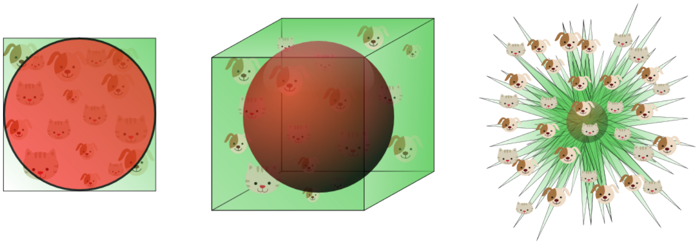
- Too few features do not allow to discriminate between classes
    - In the color image, both trees and a truck are green
- As the dimensionality of the feature space increases, the classifier's performance increases until the optimal number of features is reached
- Further increasing the dimensionality without increasing the number of training samples yields a performance decrease





Motivation - Curse of dimensionality

- As the dimensionality increases:
  - The volume of the hypersphere tends to zero
  - A larger percentage of the training data resides in the corners of the feature space
  - Distance measures start losing their effectiveness
  - Gaussian likelihoods become flat and heavy tailed distributions



How to reduce data dimensions?

Principal component analysis

Convert a set of observations of possibly correlated variables into a set of values of linearly uncorrelated variables, called **principal components**

Discriminant analysis

Find the best set of vectors which best separates the patterns

Principal component analysis

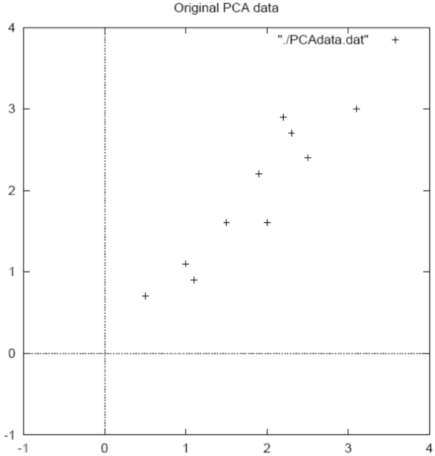
- **Goal**: represent data is a space that best describes the variation in a sum-squared error sense
- Projection onto eigenvectors that correspond to the first few largest eigenvalues of the covariance matrix
  - d-dimensional data are represented in a lower-dimensional space
  - Reduces the space and time complexities
- Intuitive introduction: `http://www.youtube.com/watch?v=BfTMmoDFXyE&feature=related`

Principal component analysis

- **Step 1**: Get some data

Data =

x	y
2.5	2.4
0.5	0.7
2.2	2.9
1.9	2.2
3.1	3.0
2.3	2.7
2	1.6
1	1.1
1.5	1.6
1.1	0.9



Principal component analysis

- **Step 2**: Subtract the mean
  - From each of the data dimensions (from x- and y-dimension)

Data =

x	y
2.5	2.4
0.5	0.7
2.2	2.9
1.9	2.2
3.1	3.0
2.3	2.7
2	1.6
1	1.1
1.5	1.6
1.1	0.9

DataAdjust =

x	y
.69	.49
-1.31	-1.21
.39	.99
.09	.29
1.29	1.09
.49	.79
.19	-.31
-.81	-.81
-.31	-.31
-.71	-1.01

Principal component analysis

- **Step 3**: Calculate the covariance matrix

Data =

x	y
2.5	2.4
0.5	0.7
2.2	2.9
1.9	2.2
3.1	3.0
2.3	2.7
2	1.6
1	1.1
1.5	1.6
1.1	0.9

$$\Rightarrow$$

$$cov = \begin{pmatrix} .616555556 & .615444444 \\ .615444444 & .716555556 \end{pmatrix}$$

Principal component analysis

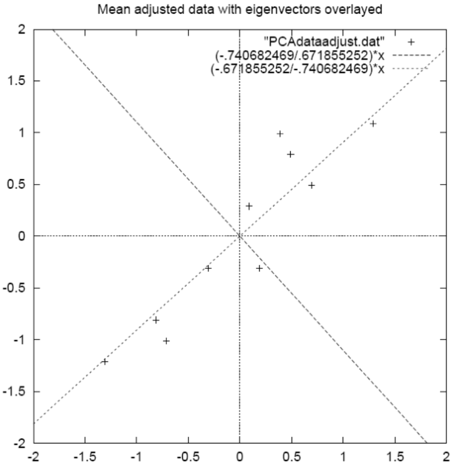
- **Step 4**: Calculate the unit eigenvectors and eigenvalues of the covariance matrix

$$cov = \begin{pmatrix} .616555556 & .615444444 \\ .615444444 & .716555556 \end{pmatrix}$$

$$\Downarrow$$

$$eigenvalues = \begin{pmatrix} .0490833989 \\ 1.28402771 \end{pmatrix}$$

$$eigenvectors = \begin{pmatrix} -.735178656 & -.677873399 \\ .677873399 & -.735178656 \end{pmatrix}$$



Principal component analysis

- The 1st eigenvector (principle component) shows how data in two dimensions are related along the eigenvector line
- The 2nd eigenvector shows that all the points are off to the side of the main line by some amount
- Eigenvectors are lines that characterize the data
- The next steps: transforming the data so that it is expressed in terms of these lines

Principal component analysis

- **Step 5**: Choose components and form a feature vector
  - Order eigenvectors by eigenvalues
    - This gives the components in order of significance
    - You can decide to ignore the components of lesser significance $\Rightarrow$ final data will have less dimensions ($p < d$)
  - Form a feature vector (matrix of vectors):

$$FeatureVector = (eig_1 \;\; eig_2 \;\; eig_3)$$

- For our example, two feature vectors are possible:

$$eigenvalues = \begin{pmatrix} .0490833989 \\ 1.28402771 \end{pmatrix}$$

$$eigenvectors = \begin{pmatrix} -.735178656 & -.677873399 \\ .677873399 & -.735178656 \end{pmatrix}$$

$$\begin{pmatrix} -.677873399 & -.735178656 \\ -.735178656 & .677873399 \end{pmatrix}$$

or

$$\begin{pmatrix} -.677873399 \\ -.735178656 \end{pmatrix}$$

Principal component analysis

- **Step 6**: Derive the new dataset:

$$FinalData = FeatureVector^T \times RowDataAdjust$$

  where *RowDataAdjust* is the mean-adjusted data transposed

- It will give us the original data solely in terms of the vectors we chose

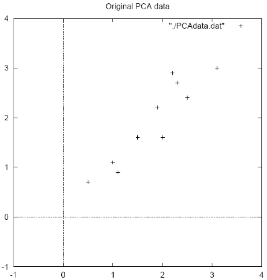
Principal component analysis

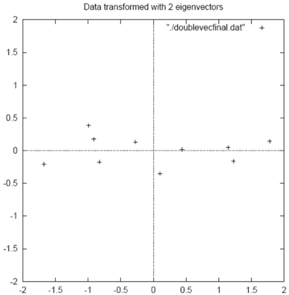
Data =

x	y
2.5	2.4
0.5	0.7
2.2	2.9
1.9	2.2
3.1	3.0
2.3	2.7
2	1.6
1	1.1
1.5	1.6
1.1	0.9

Transformed Data=

x	y
-.827970186	-.175115307
1.77758033	.142857227
-.992197494	.384374989
-.274210416	.130417207
-1.67580142	-.209498461
-.912949103	.175282444
.0991094375	-.349824698
1.14457216	.0464172582
.438046137	.0177646297
1.22382056	-.162675287



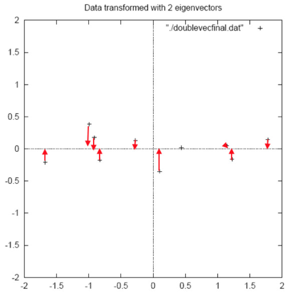


Principal component analysis (PCA)

Transformed Data=

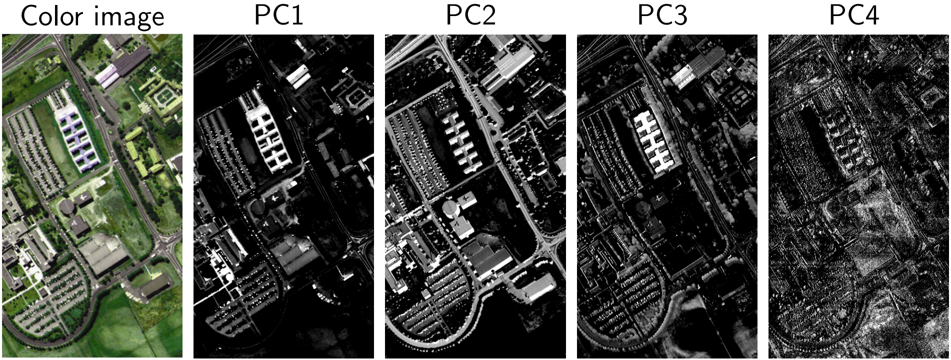
x	y
-.827970186	-.175115307
1.77758033	.142857227
-.992197494	.384374989
-.274210416	.130417207
-1.67580142	-.209498461
-.912949103	.175282444
.0991094375	-.349824698
1.14457216	.0464172582
.438046137	.0177646297
1.22382056	-.162675287

- If only one eigenvector was kept, the transformed data will have only one dimension



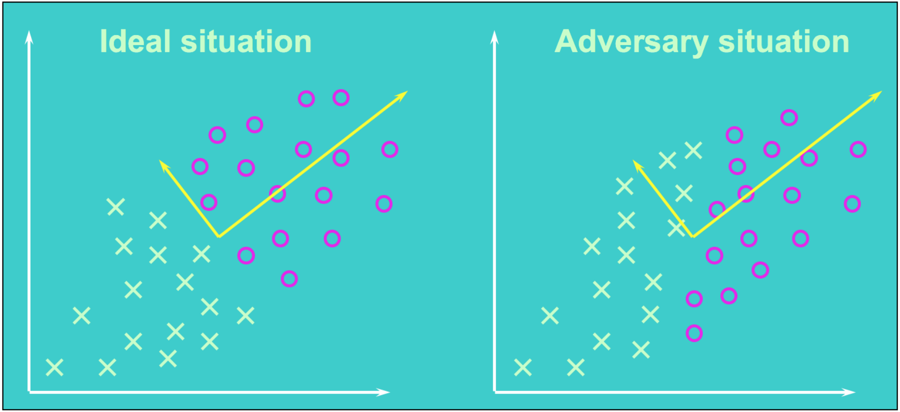
Example of PCA for hyperspectral image analysis

- Principal component analysis in the spectral space
  - Principal components (PCs) 1-3 contain 97% of information from original 103 channels



Principal component analysis

- Projection onto eigenvectors that correspond to the first few largest eigenvalues of the covariance matrix

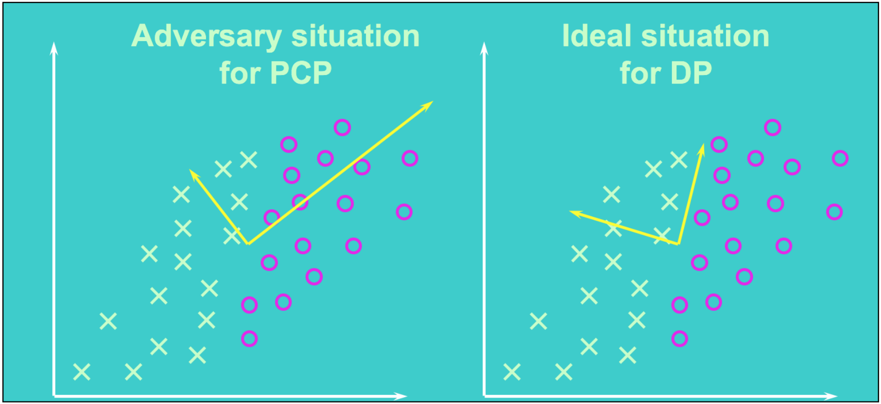


Discriminant analysis

- PCA seeks directions that are efficient for **representation**
  - *Unsupervised technique*
- Discriminant analysis seeks directions that are efficient for **discrimination**
  - *Supervised technique*

Discriminant analysis

- Projection onto directions that can best separate data of different classes



Discriminant analysis

- Theory of Fisher linear discriminant: `http://www.csd.uwo.ca/~olga/Courses//CS434a_541a//Lecture8.pdf`
- Project on line in the direction v which maximizes:

$$J(v) = \frac{\overbrace{(\tilde{\mu}_1 - \tilde{\mu}_2)^2}^{\text{want projected means are far from each other}}}{\tilde{s}_1^2 + \tilde{s}_2^2}$$

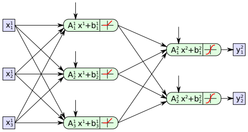
want scatter in class 1 is as small as possible, i.e. samples of class 1 cluster around the projected mean $\tilde{\mu}_1$

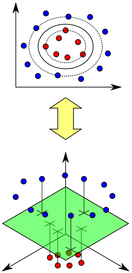
want scatter in class 2 is as small as possible, i.e. samples of class 2 cluster around the projected mean $\tilde{\mu}_2$

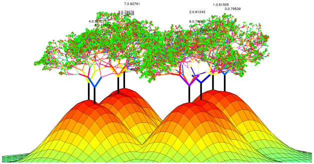
- Main drawback: in most real-life cases, projection to even the best line results in unseparable projected samples

Models

1. Classification based on Features
   - Decision Boundary
   - Linear Decision Boundary
   - Non-linear Decision Boundary
2. Feature extraction
3. Machine Learning Methods
   - Support Vector Machine (SVM)
   - Multi-Layer Perceptron (MLP)
   - Random Forest (RF)

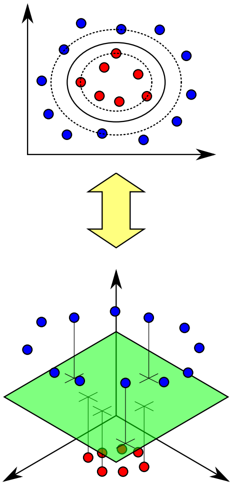






Models

1. Classification based on Features
    - Decision Boundary
    - Linear Decision Boundary
    - Non-linear Decision Boundary
2. Feature extraction
3. Machine Learning Methods
    - Support Vector Machine (SVM)
    - Multi-Layer Perceptron (MLP)
    - Random Forest (RF)

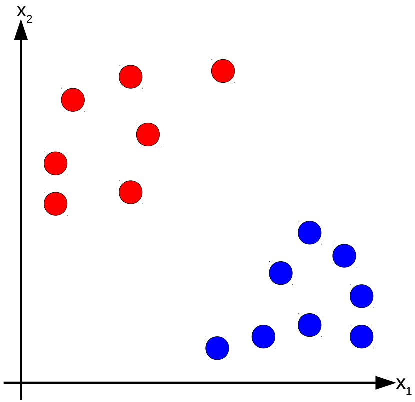


SVM

Reconsider the perceptron:

Perceptron

$$y = \text{sign}(\mathbf{w}^T \mathbf{x} + b) \qquad (2)$$

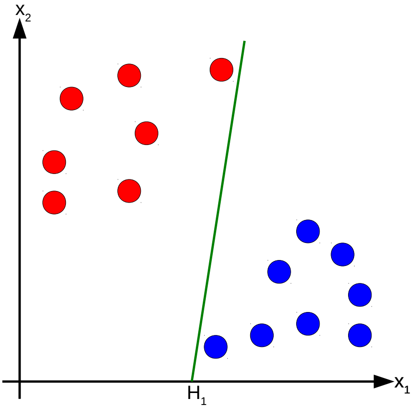


SVM

Reconsider the perceptron:

Perceptron

$$y = \text{sign}(\mathbf{w}^T\mathbf{x} + b) \qquad (2)$$

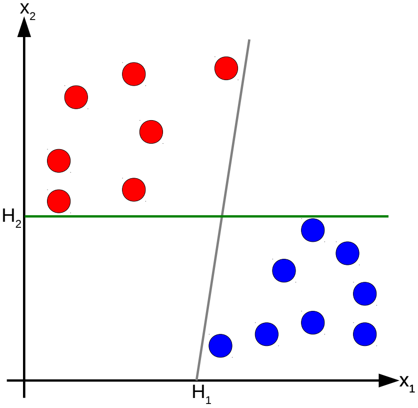


SVM

Reconsider the perceptron:

Perceptron

$$y = \text{sign}(\mathbf{w}^T\mathbf{x} + b) \qquad (2)$$

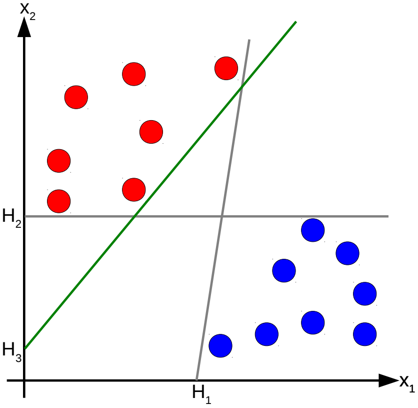


SVM

Reconsider the perceptron:

Perceptron

$$y = \text{sign}(\mathbf{w}^T\mathbf{x} + b) \qquad (2)$$

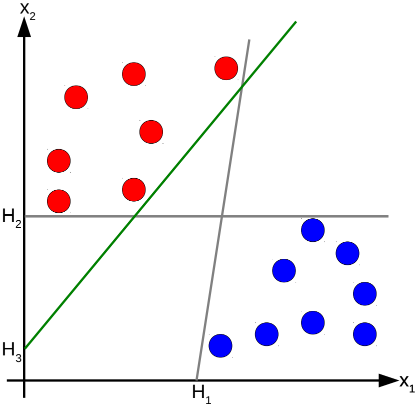


SVM

Reconsider the perceptron:

Perceptron

$$y = \text{sign}(\mathbf{w}^T \mathbf{x} + b) \qquad (2)$$

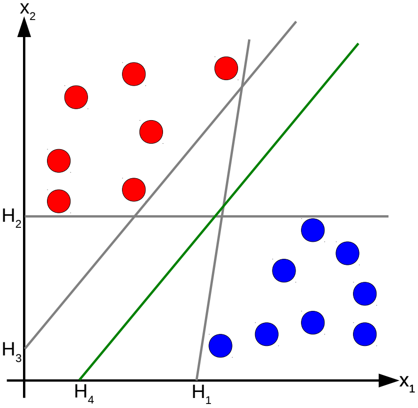


SVM

Reconsider the perceptron:

Perceptron

$$y = \text{sign}(\mathbf{w}^T\mathbf{x} + b) \tag{2}$$



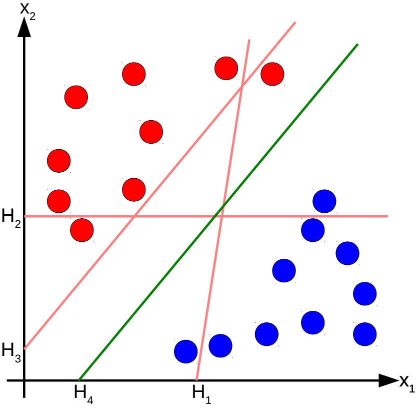
SVM

Reconsider the perceptron:

Perceptron

$$y = \mathrm{sign}(\mathbf{w}^T\mathbf{x} + b) \qquad (2)$$

- Don't just pick any decision boundary
- Pick the one with the *maximal margin*
- Perceptron of maximal stability



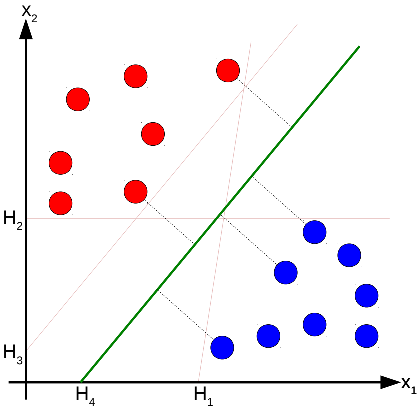
SVM

Reconsider the perceptron:

Perceptron

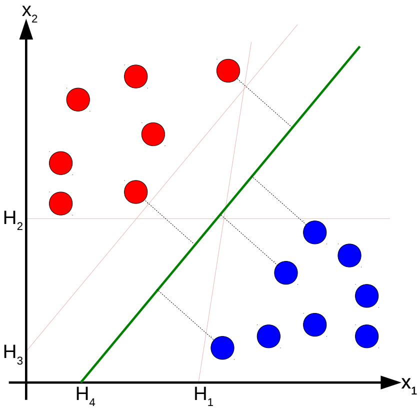
$$y = \text{sign}(\mathbf{w}^T\mathbf{x} + b) \qquad (2)$$

- Don't just pick any decision boundary
- Pick the one with the *maximal margin*
- Perceptron of maximal stability



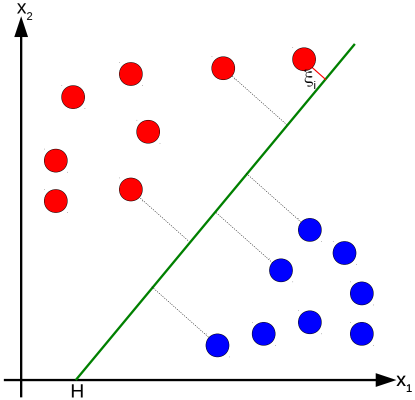
SVM

- Maximal margin equivalent to:
  Minimize $||w||^2$
  subject to $\hat{y}_i(w^T \mathbf{x}_i - b) \geq 1$



SVM

- Maximal margin equivalent to:
  Minimize $||\mathbf{w}||^2$
  subject to $\hat{y}_i(\mathbf{w}^T\mathbf{x}_i - b) \geq 1$
- Allow small errors (soft margin):
  Minimize $\lambda||w||^2 + \frac{1}{n}\sum_{i=1}^{n}\xi_i$
  subject to $\hat{y}_i(w^T\mathbf{x}_i - b) \geq 1 - \xi_i$
  $(\xi_i \geq 0)$

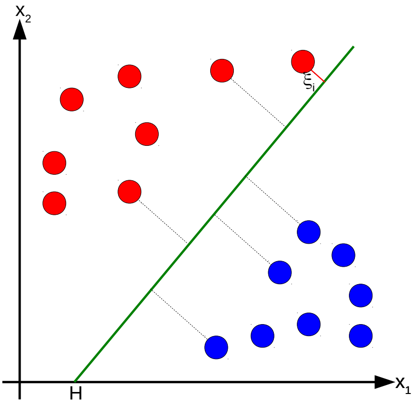


SVM

- The Lagrangian dual gives:
  Maximize

$$\sum_{i=1}^{n} \alpha_i - \frac{1}{2} \sum_{i=1}^{n} \sum_{j=1}^{n} \hat{y}_i \alpha_i (\mathbf{x}_i \cdot \mathbf{x}_j) \hat{y}_j \alpha_j$$

  subject to $\sum_{i=1}^{n} \alpha_i \hat{y}_i = 0$
- Support vectors: $\mathbf{x}_i$ if $\alpha_i \neq 0$
- Classification: $sign(\mathbf{w}^T \mathbf{x} + b)$
  with $\mathbf{w} = \sum_{i=1}^{n} \alpha_i \hat{y}_i \mathbf{x}_i$

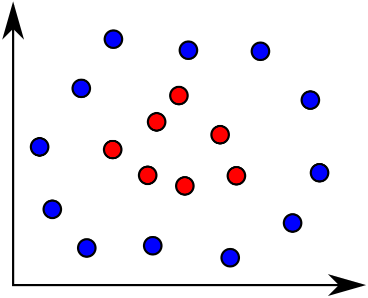


SVM

- The Lagrangian dual gives:
  Maximize

$$\sum_{i=1}^{n}\alpha_i - \frac{1}{2}\sum_{i=1}^{n}\sum_{j=1}^{n}\hat{y}_i\alpha_i(\mathbf{x}_i\cdot\mathbf{x}_j)\hat{y}_j\alpha_j$$

  subject to $\sum_{i=1}^{n}\alpha_i\hat{y}_i = 0$
- Support vectors: $\mathbf{x}_i$ if $\alpha_i \neq 0$
- Classification: $sign(\mathbf{w}^T\mathbf{x} + b)$
  with $\mathbf{w} = \sum_{i=1}^{n}\alpha_i\hat{y}_i\mathbf{x}_i$
- What if $\mathbf{x}_i$ not linear separable at all?

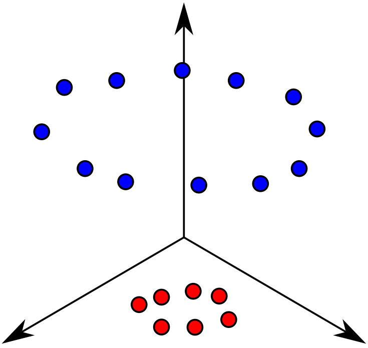


SVM

- The Lagrangian dual gives:
  Maximize

$$\sum_{i=1}^{n} \alpha_i - \frac{1}{2} \sum_{i=1}^{n} \sum_{j=1}^{n} \hat{y}_i \alpha_i (\mathbf{x}_i \cdot \mathbf{x}_j) \hat{y}_j \alpha_j$$

  subject to $\sum_{i=1}^{n} \alpha_i \hat{y}_i = 0$
- Support vectors: $\mathbf{x}_i$ if $\alpha_i \neq 0$
- Classification: $sign(\mathbf{w}^T \mathbf{x} + b)$
  with $\mathbf{w} = \sum_{i=1}^{n} \alpha_i \hat{y}_i \mathbf{x}_i$
- What if $\mathbf{x}_i$ not linear separable at all?
  $\rightarrow$ Compute new features $\mathbf{x} \mapsto \phi(\mathbf{x})$

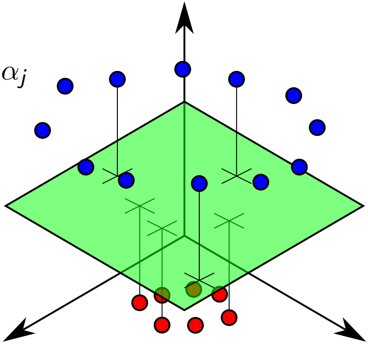


SVM

- The Lagrangian dual gives:
  Maximize

$$\sum_{i=1}^{n} \alpha_i - \frac{1}{2} \sum_{i=1}^{n} \sum_{j=1}^{n} \hat{y}_i \alpha_i (\phi(\mathbf{x}_i) \cdot \phi(\mathbf{x}_j)) \hat{y}_j \alpha_j$$

  subject to $\sum_{i=1}^{n} \alpha_i \hat{y}_i = 0$
- Support vectors: $\mathbf{x}_i$ if $\alpha_i \neq 0$
- Classification: $sign(\mathbf{w}^T \phi(\mathbf{x}) + b)$
  with $\mathbf{w} = \sum_{i=1}^{n} \alpha_i \hat{y}_i \phi(\mathbf{x}_i)$
- What if $\mathbf{x}_i$ not linear separable at all?
  $\rightarrow$ Compute new features $\mathbf{x} \mapsto \phi(\mathbf{x})$

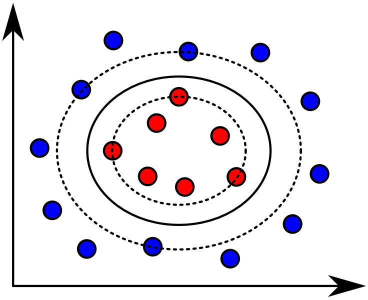


SVM

- The Lagrangian dual gives:
  Maximize

$$\sum_{i=1}^{n} \alpha_i - \frac{1}{2} \sum_{i=1}^{n} \sum_{j=1}^{n} \hat{y}_i \alpha_i k(\mathbf{x}_i, \mathbf{x}_j) \hat{y}_j \alpha_j$$

  subject to $\sum_{i=1}^{n} \alpha_i \hat{y}_i = 0$
- Support vectors: $\mathbf{x}_i$ if $\alpha_i \neq 0$
- Classification:
  $sign(\sum_{i=1}^{n} \alpha_i \hat{y}_i k(\mathbf{x}_i, \mathbf{x}) + b)$
- What if $\mathbf{x}_i$ not linear separable at all?
  $\rightarrow$ Compute new features $\mathbf{x} \mapsto \phi(\mathbf{x})$
- Use $k(\mathbf{x}_i, \mathbf{x}_j) = \phi(\mathbf{x}_i) \cdot \phi(\mathbf{x}_j)$



SVM Kernels

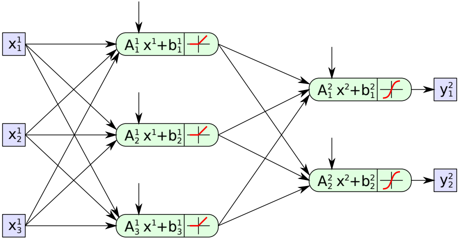
- Multiple kernels exist
  - Linear $k(\mathbf{x}_i, \mathbf{x}_j) = \mathbf{x}_i \cdot \mathbf{x}_j$
  - Polynomial $k(\mathbf{x}_i, \mathbf{x}_j) = (\mathbf{x}_i \cdot \mathbf{x}_j)^d$
  - RBF $k(\mathbf{x}_i, \mathbf{x}_j) = \exp(-\gamma||\mathbf{x}_i - \mathbf{x}_j)||^2$
  - Hyperbolic tangent $k(\mathbf{x}_i, \mathbf{x}_j) = \tanh(\kappa \cdot \mathbf{x}_i \cdot \mathbf{x}_j + c)$
- Linear kernel very fast and easy to train, but very simple
- RBF kernel very powerful and most often used
- Kernel can (should) be adapted to task and data
  $\rightarrow$ e.g. complex-valued kernels are possible [Moser and Serpico, 2014]
  $$k(\mathbf{z}, \mathbf{s}) = \Re\left[\exp\left(-\frac{1}{2\sigma^2}\sum_{r=1}^{d}(z_r - s_r^*)^2\right)\right]$$
- Kernels for different features can be fused into one common kernel

SVM Conclusion

- Kernels can be designed to different purposes
- Hyperparameter tuning not easy
  $\rightarrow$ Usually grid search with cross validation
- Slow for large amounts of data
  $\rightarrow$ Potentially results in many support vectors and thus scalar products during prediction
- (Usually) all data needs to be considered at once
  $\rightarrow$ No “streaming” of data
- Designed for binary tasks
  $\rightarrow$ Extension to multi-class problems usually decreases performance and increases computational load

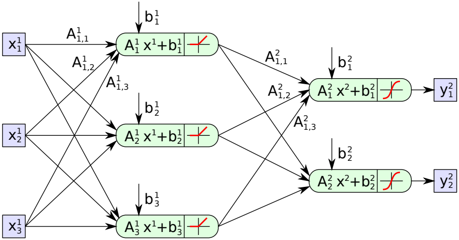
Models

1. Classification based on Features
   - Decision Boundary
   - Linear Decision Boundary
   - Non-linear Decision Boundary
2. Feature extraction
3. Machine Learning Methods
   - Support Vector Machine (SVM)
   - Multi-Layer Perceptron (MLP)
   - Random Forest (RF)



Multi-Layer Perceptron

- Feed forward neural network
- Neural networks “inspired by biology”
  - But work quite differently
- Core idea: concatenate multiple simple mappings to get one powerful mapping
- Multiple simple steps more powerful than one complex step
- Keep everything (mostly) differentiable
- Train by doing gradient descend on classification error



Building Blocks

Standard Layers:

- Fully connected layer with...
- ... activation function

Building Blocks

Standard Layers:

- Fully connected layer with...
- ... activation function

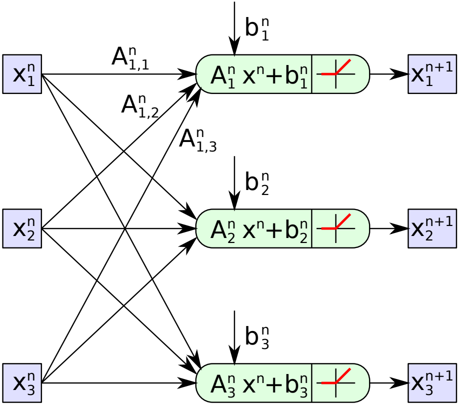
Special Layers (selection):

- Dropout (for regularization)
- Normalization (Improves training)
- Softmax (Produces nice classification output)

Fully Connected Layer

$$\mathbf{x}^{n+1} = \mathbf{y}^n = f(\mathbf{A}^n \cdot \mathbf{x}^n + \mathbf{b}^n) \quad (3)$$

- $\mathbf{x}^n$: Layer input
- $\mathbf{y}^n = \mathbf{x}^{n+1}$: Layer output
- $\mathbf{A}^n$: Weights
- $\mathbf{b}^n$: Bias
- $f(\cdot)$: Activation function



Activation Functions

$$\mathbf{y}^n = f(\mathbf{A}^n \cdot \mathbf{x}^n + \mathbf{b}^n) \tag{4}$$

- Assume $f(x) = x$
- Layer can assume any linear function (plus offset)

Activation Functions

$$\mathbf{y}^n = f(\mathbf{A}^n \cdot \mathbf{x}^n + \mathbf{b}^n) \tag{4}$$

- Assume $f(x) = x$
- Layer can assume any linear function (plus offset)
- Stacked layers can't improve that
- Activation function must be non-linear

Activation Functions

Typical choices:

ReLU

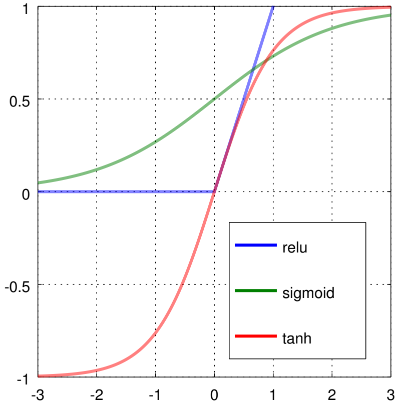
$$f(\mathbf{x}_i)_i = max(\mathbf{x}_i, 0) \tag{5}$$

Sigmoid / Logistic

$$f(\mathbf{x}_i)_i = \frac{1}{1 + e^{-\mathbf{x}_i}} \tag{6}$$

TanH

$$f(\mathbf{x}_i)_i = tanh(\mathbf{x}_i) = \frac{e^{\mathbf{x}_i} - e^{-\mathbf{x}_i}}{e^{\mathbf{x}_i} + e^{-\mathbf{x}_i}} \tag{7}$$



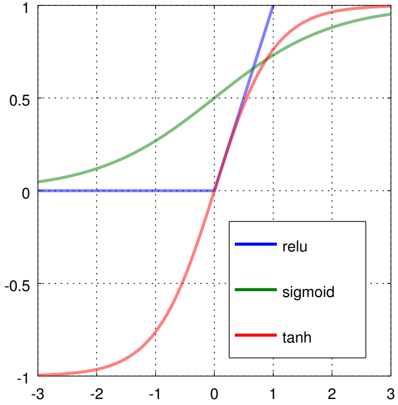
Activation Functions

Typical choices:

ReLU

$$f(\mathbf{x}_i)_i = max(\mathbf{x}_i, 0) \tag{8}$$

- ReLU (and variations of it) today the most common choice
- Better for deep networks
  - Derivative of activation function = 1 (in positive direction)
  - No saturation (in positive direction)
  - Gradients propagate better



Training

- How to find correct model parameters $\boldsymbol{\theta}$?
  - weight values
  - bias values
  - sometimes aux parameters

Training

- How to find correct model parameters $\boldsymbol{\theta}$?
  - weight values
  - bias values
  - sometimes aux parameters
- Setup/define energy function objective $E(\boldsymbol{\theta})$
- Derive analytic gradients $\frac{\partial E(\boldsymbol{\theta})}{\partial \boldsymbol{\theta}}$
- Perform gradient descent $\Delta\boldsymbol{\theta} = -\lambda \cdot \frac{\partial E(\boldsymbol{\theta})}{\partial \boldsymbol{\theta}}$
  - Usually slightly more sophisticated, more later

Training Objective

Empirical Risk Minimization (over N training samples)

$$E(\boldsymbol{\theta}) = \sum_{\alpha}^{N} e(\mathbf{y}^L(\underbrace{\boldsymbol{x}_\alpha}_{\text{Training sample}}, \boldsymbol{\theta}), \overbrace{\hat{\boldsymbol{y}}_\alpha}^{\text{Known/Desired value/label of } \boldsymbol{x}_\alpha}) \tag{9}$$

with, e.g.,:

$$e(\mathbf{y}^a, \mathbf{y}^b) = \left|\mathbf{y}^a - \mathbf{y}^b\right|^2 \tag{10}$$

Though bad for classification, see softmax layer later.

- Energy function defines training loss
- Gradient descent will try to minimize this
- Usually not convex (as network not convex)

Backpropagation

- How to compute $\frac{\partial E(\boldsymbol{\theta})}{\partial \boldsymbol{\theta}}$?
- MLP is concatenation of "simple" functions
  $\mathbf{y}^L(\ldots \mathbf{y}^2(\mathbf{y}^1(\mathbf{x}^1, \boldsymbol{\theta}^1), \boldsymbol{\theta}^2), \ldots \boldsymbol{\theta}^L)$

Backpropagation

- How to compute $\frac{\partial E(\boldsymbol{\theta})}{\partial \boldsymbol{\theta}}$?
- MLP is concatenation of "simple" functions $\mathbf{y}^L(\ldots \mathbf{y}^2(\mathbf{y}^1(\mathbf{x}^1, \boldsymbol{\theta}^1), \boldsymbol{\theta}^2), \ldots \boldsymbol{\theta}^L)$
- Exploit chain rule

$$\frac{\partial E(\boldsymbol{\theta})}{\partial \boldsymbol{\theta}^1} = \frac{\partial E(\boldsymbol{\theta})}{\partial \mathbf{y}^L} \cdot \ldots \cdot \underbrace{\frac{\partial \mathbf{y}^3}{\partial \mathbf{y}^2}}_{\text{per layer output derivative}} \cdot \frac{\partial \mathbf{y}^2}{\partial \mathbf{y}^1} \cdot \overbrace{\frac{\partial \mathbf{y}^1}{\partial \boldsymbol{\theta}^1}}^{\text{per layer parameter derivative}} \tag{11}$$

Backpropagation

- How to compute $\frac{\partial E(\boldsymbol{\theta})}{\partial \boldsymbol{\theta}}$?
- MLP is concatenation of "simple" functions $\mathbf{y}^L(\ldots \mathbf{y}^2(\mathbf{y}^1(\mathbf{x}^1, \boldsymbol{\theta}^1), \boldsymbol{\theta}^2), \ldots \boldsymbol{\theta}^L)$
- Exploit chain rule

$$\frac{\partial E(\boldsymbol{\theta})}{\partial \boldsymbol{\theta}^1} = \frac{\partial E(\boldsymbol{\theta})}{\partial \mathbf{y}^L} \cdot \ldots \cdot \underbrace{\frac{\partial \mathbf{y}^3}{\partial \mathbf{y}^2}}_{\text{per layer output derivative}} \cdot \frac{\partial \mathbf{y}^2}{\partial \mathbf{y}^1} \cdot \overbrace{\frac{\partial \mathbf{y}^1}{\partial \boldsymbol{\theta}^1}}^{\text{per layer parameter derivative}} \tag{11}$$

- Gradient computation happens in two passes:
  - Forward pass:
    - Feeds training data through network
    - Computes all $\mathbf{y}^n$ and training loss
  - Backward pass:
    - Feeds error gradient backward through network
    - Computes all $\frac{\partial E(\boldsymbol{\theta})}{\partial \mathbf{y}^n}$ and $\frac{\partial E(\boldsymbol{\theta})}{\partial \boldsymbol{\theta}^n}$

Stochastic Gradient Descent

- Exact gradient usually not needed or wanted
- Just empirical average over N samples anyways
- Stochastic Gradient Descent: Split into batches of $M < N$ samples and update weights after every batch

$$\Delta\boldsymbol{\theta} = -\lambda \cdot \frac{\partial \hat{E}(\boldsymbol{\theta})}{\partial \boldsymbol{\theta}} = \frac{\partial}{\partial \boldsymbol{\theta}} \sum_{\alpha}^{M} e(\mathbf{y}^L(\boldsymbol{x}_\alpha, \boldsymbol{\theta}), \hat{\boldsymbol{y}}_\alpha) \tag{12}$$

- Usually small batch sizes (eg. around 128) sufficient
  - Stepsize limited by curvature of energy function, not by precision of gradient
  - Computation time increases with $O(M)$, precision of gradient only with $O(\sqrt{M})$
  - Large batch sizes lead to sharp minimizers that don't generalize
  - Further reading: [Keskar et al., 2016]

Parameter Update Rule

- $\Delta\boldsymbol{\theta} = -\lambda \cdot \frac{\partial \hat{E}(\boldsymbol{\theta})}{\partial \boldsymbol{\theta}}$ most simple update rule
- Momentum
  - Accumulate "momentum" over time
  - Pick up speed in the valley direction, average out noise
- Adam [Kingma and Ba, 2014]/Adagrad/Adadelta [Zeiler, 2012]
  - Normalize based on average gradient variance in the past

Parameter Initialization

- How to initialize $\boldsymbol{\theta}$?
- Random Gaussian
- Xavier (and some variants) [Glorot and Bengio, 2010]
  - Draw weights randomly
  - Choose variance per layer depending on input/output size
  - Balance variance to keep signal/gradient variance constant

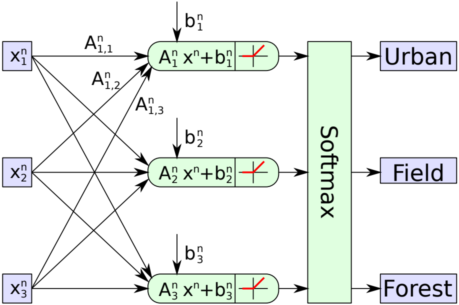
Special Layers

- Softmax
- Normalization
- Dropout

Softmax

$$f(x_i)_i = \frac{exp(x_i)}{\sum_j exp(x_j)} \qquad (13)$$

- Special (last) layer/activation for classification
- Creates vector that sums to one (read probabilities), one element per class
- Usually together with a specific optimization objective: Cross-entropy loss
    - Comparing the predicted probability mass distribution to the ground truth one



Dropout

- [Srivastava et al., 2014]
- During training, randomly disable neurons with probability p
- During application, scale output with $1 - p$
- Prevents co-adaptation
- Fosters redundancy throughout the network
- Reduces overfitting and improves generalization

Normalization

- Normalization can be important for learning
- Neither signal (forward) nor gradients (backward) must explode/shrink in magnitude

Normalization

- Normalization can be important for learning
- Neither signal (forward) nor gradients (backward) must explode/shrink in magnitude
- Input Normalization
  - Normalize input to have zero mean and unit stddev

Normalization

- Normalization can be important for learning
- Neither signal (forward) nor gradients (backward) must explode/shrink in magnitude
- Input Normalization
  - Normalize input to have zero mean and unit stddev
- Local Response Normalization (LRN) [Krizhevsky et al., 2012]
  - Special layer placed at strategic locations
  - Let strong activations inhibit activations of other neurons in same layer
  - Normalizes the otherwise unbounded output of ReLU

Normalization

- Normalization can be important for learning
- Neither signal (forward) nor gradients (backward) must explode/shrink in magnitude
- Input Normalization
    - Normalize input to have zero mean and unit stddev
- Local Response Normalization (LRN) [Krizhevsky et al., 2012]
    - Special layer placed at strategic locations
    - Let strong activations inhibit activations of other neurons in same layer
    - Normalizes the otherwise unbounded output of ReLU
- Batch Normalization [Ioffe and Szegedy, 2015]
    - Special layer placed at strategic locations
    - Normalize mean and variance of activations across training batch (or accumulate running averages)
    - After learning, becomes fixed scale & offset

Handling Overfitting

- Dropout

Handling Overfitting

- Dropout
- Weight regularization
  - Penalize large weight values
  - e.g., add $\lambda \cdot |\boldsymbol{\theta}|^2$ to optimization objective
  - Soft limit on model complexity

Handling Overfitting

- Dropout
- Weight regularization
  - Penalize large weight values
  - e.g., add $\lambda \cdot |\boldsymbol{\theta}|^2$ to optimization objective
  - Soft limit on model complexity
- Data Augmentation
  - Randomly modify training data
  - Based on what kind of invariances you want to have
    - Resistance to noise: add noise
    - Resistance to brightness/contrast/hue changes: Change those
    - Translation/Rotation (ex. for images)
    - Can also be applied to data before extracting features!

Increasing Depth

- Recent trend goes towards deeper networks
- Networks more powerful, but ...
- ... more difficult to train
  - Gradients collapse/explode/diffuse through the layers
- This is the book to read: Deep Learning [Goodfellow et al., 2016]

Complex-valued MLPs

- MLPs provide a functional mapping $f : \mathcal{X} \to \mathcal{Y}$
- f needs to be differentiable (due to backprop)
- Usually $\mathcal{X} \equiv \mathbb{R}^d$ (or $\mathbb{R}^{N \times M}$)
- But: (e.g.) PolSAR images are $\mathbb{C}^{N \times M}$
  - One solution: Compute real-valued features, then use standard MLP
    $\rightarrow$ Advantage: Usage of common MLPs and their extensions
    $\rightarrow$ Disadvantage: Dependency on feature extraction
  - Second solution: Use complex-valued MLP
    $\rightarrow$ Advantage: No dependency on feature extraction
    $\rightarrow$ Disadvantage: Math slightly more complicated

Complex-valued MLPs

Gradient in $\mathbb{R}$

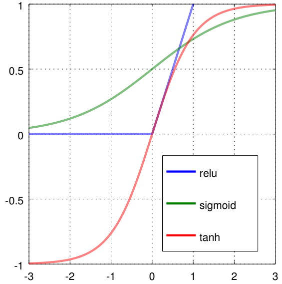
$$\begin{aligned} f &: \mathbb{R} \to \mathbb{R}, x \mapsto f(x) \\ \frac{\partial f}{\partial x} &= f'(x) \end{aligned}$$

Gradient in $\mathbb{C}$

$$\begin{aligned} f &: \mathbb{C} \to \mathbb{C}, z \mapsto f(z) \\ \frac{\partial f}{\partial z} &= \frac{1}{2}\left(\frac{\partial f}{\partial \Re z} - i\frac{\partial f}{\partial \Im z}\right) \\ \frac{\partial f}{\partial z^*} &= \frac{1}{2}\left(\frac{\partial f}{\partial \Re z} + i\frac{\partial f}{\partial \Im z}\right) \end{aligned}$$

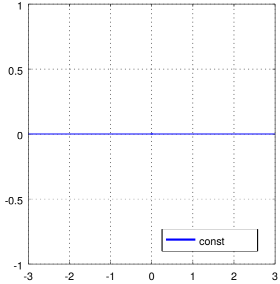
Complex-valued MLPs

Activation in $\mathbb{R}$



Analytical and bounded
$\rightarrow$ e.g. tanh, logistic function
(ReLU as exception)

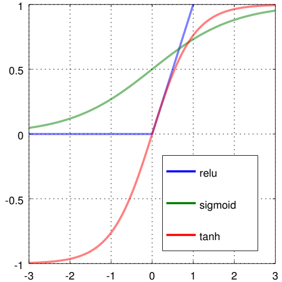
Activation in $\mathbb{C}$



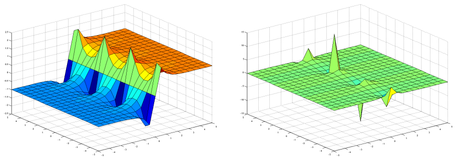
Analytical and bounded?
$\rightarrow$ only constant functions

Complex-valued MLPs

Activation in $\mathbb{R}$



Activation in $\mathbb{C}$

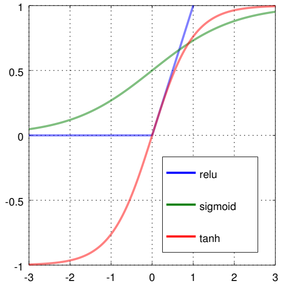


Analytical and bounded
$\rightarrow$ e.g. tanh, logistic function

Analytical or bounded
$\rightarrow$ e.g. tanh

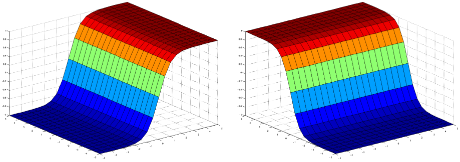
Complex-valued MLPs

Activation in $\mathbb{R}$



Analytical and bounded
$\rightarrow$ e.g. tanh, logistic function

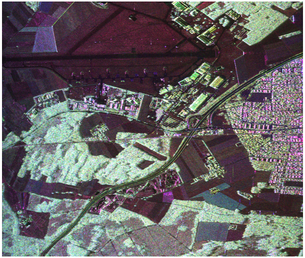
Activation in $\mathbb{C}$



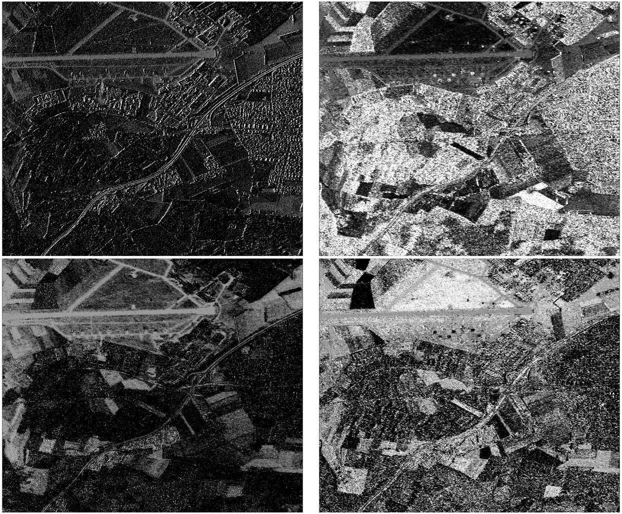
Analytical or **bounded**
$\rightarrow$ e.g. split-tanh

$$f(z) = \tanh(\Re(z)) + i \tanh(\Im(z))$$

Complex-valued MLPs



- PolSAR Data: $\mathbb{C}^{N\times M\times 3\times 3}$
- Input: Local patches, each pixel a Hermitan matrix (local covariance matrix of complex-valued scattering vector)
- Activation of few neurons in first layer is shown.

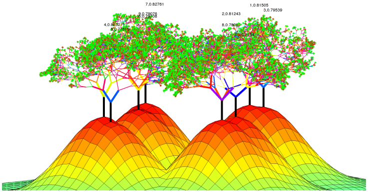


MLP Conclusion

- Architecture design a bit of an art
    - Though some tips/tricks exist
- Can ingest a lot of training data
- Training/Application not fast
- With modern tricks (ReLU, normalization, ...) scale surprisingly well
    - Up to very complex networks
    - Trained on lots of data

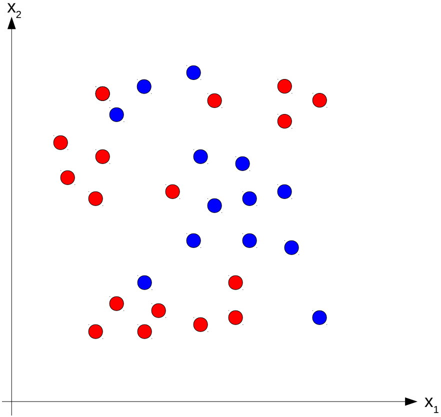
Models

1. Classification based on Features
   - Decision Boundary
   - Linear Decision Boundary
   - Non-linear Decision Boundary
2. Feature extraction
3. Machine Learning Methods
   - Support Vector Machine (SVM)
   - Multi-Layer Perceptron (MLP)
   - Random Forest (RF)



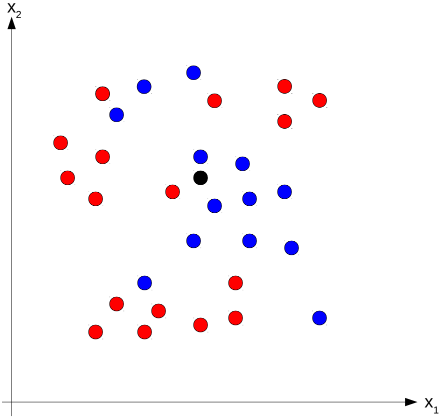
From kNN to Search Trees

- Data samples $\mathbf{x}$
  - Pixel information, image patch, feature vector, etc.
  - Often $\mathbf{x} \in \mathbb{R}^n$
- Classification:
  $\Rightarrow$ Estimate class label
- Training data: Values of target variable given e.g. class label



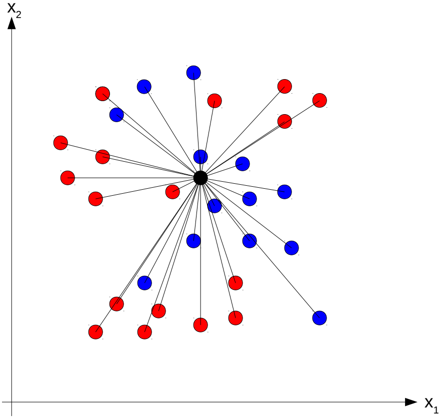
From kNN to Search Trees

- Task: Given training data, estimate label of query sample



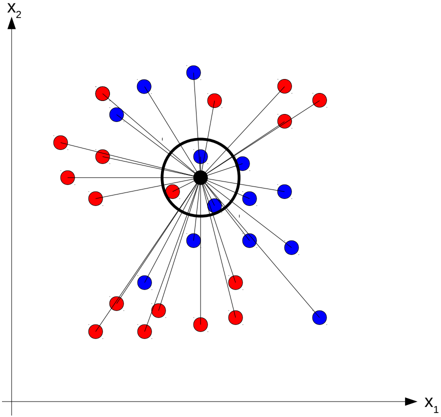
From kNN to Search Trees

- Task: Given training data, estimate label of query sample
- kNN/Parzen Window:
  - Compute distance to **all** samples



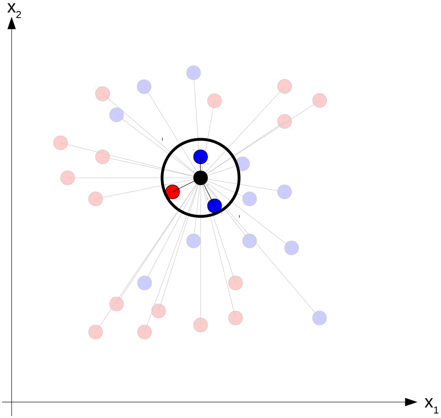
From kNN to Search Trees

- Task: Given training data, estimate label of query sample
- kNN/Parzen Window:
  - Compute distance to **all** samples
  - Select samples within window of given size (Parzen)



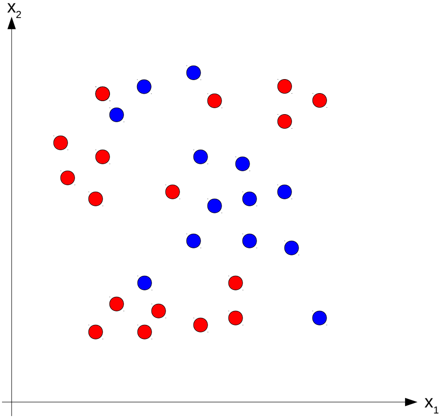
From kNN to Search Trees

- Task: Given training data, estimate label of query sample
- kNN/Parzen Window:
    - Compute distance to **all** samples
    - Select samples within window of given size (Parzen)
    - Use these samples to estimate target variable, e.g. class label
- Problem: Computationally expensive (exhaustive search)



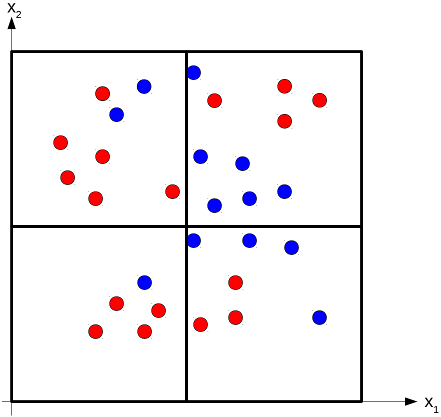
From kNN to Search Trees

- Search trees
  $\rightarrow$ Quad/Octree, KD tree, etc.



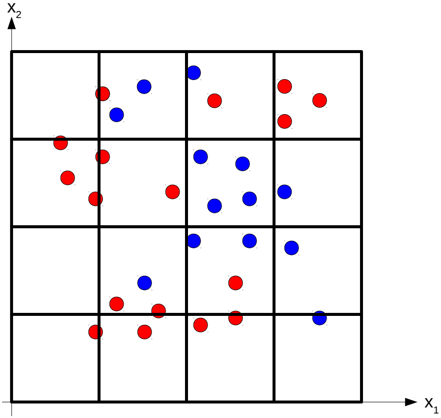
From kNN to Search Trees

- Search trees
  $\rightarrow$ Quad/Octree, KD tree, etc.
  - Divide space recursively into cells



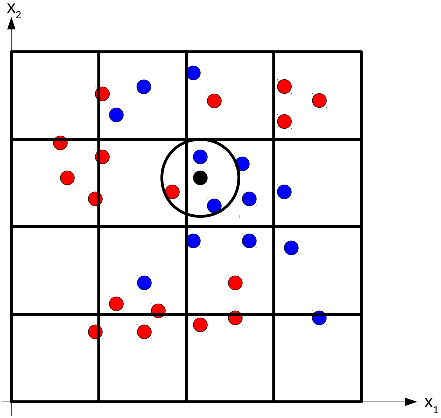
From kNN to Search Trees

- Search trees
  → Quad/Octree, KD tree, etc.
  - Divide space recursively into cells



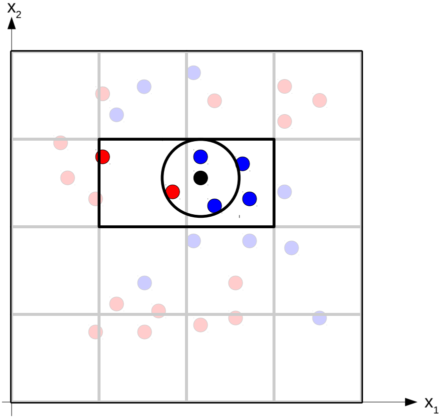
From kNN to Search Trees

- Search trees
  $\rightarrow$ Quad/Octree, KD tree, etc.
  - Divide space recursively into cells
  - Given a query, find relevant cells



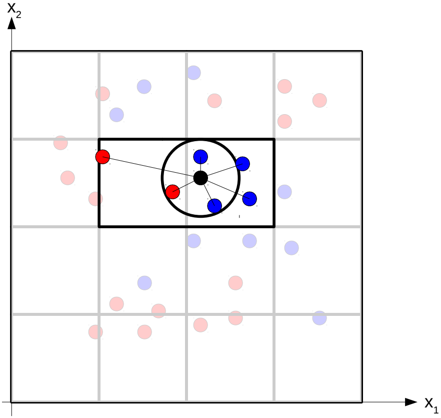
From kNN to Search Trees

- Search trees
  $\rightarrow$ Quad/Octree, KD tree, etc.
  - Divide space recursively into cells
  - Given a query, find relevant cells



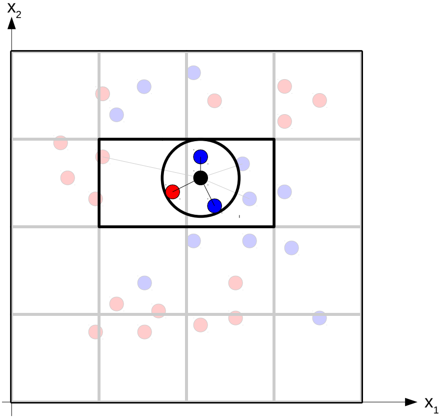
From kNN to Search Trees

- Search trees
  $\rightarrow$ Quad/Octree, KD tree, etc.
  - Divide space recursively into cells
  - Given a query, find relevant cells
  - Perform exhaustive search in these cells ONLY



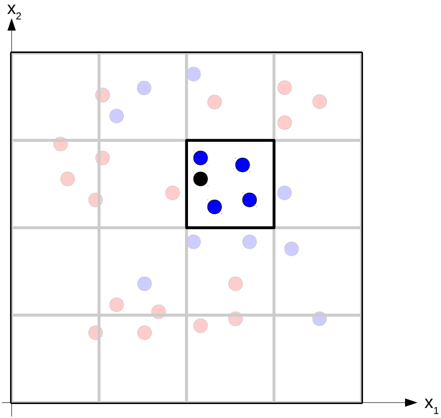
From kNN to Search Trees

- Search trees
  $\rightarrow$ Quad/Octree, KD tree, etc.
  - Divide space recursively into cells
  - Given a query, find relevant cells
  - Perform exhaustive search in these cells ONLY
- Exact search: Leads to equivalent results



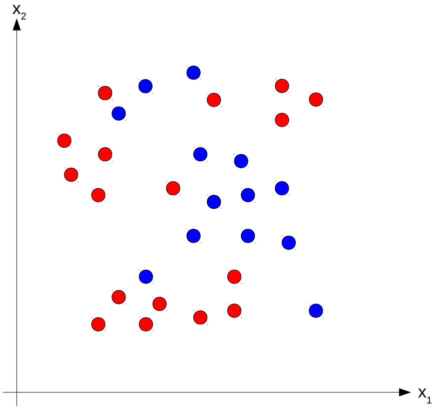
From kNN to Search Trees

- Search trees
  $\rightarrow$ Quad/Octree, KD tree, etc.
  - Divide space recursively into cells
  - Given a query, find relevant cells
  - Perform exhaustive search in these cells ONLY
- Exact search: Leads to equivalent results
- Approximation: Use samples within query cell directly



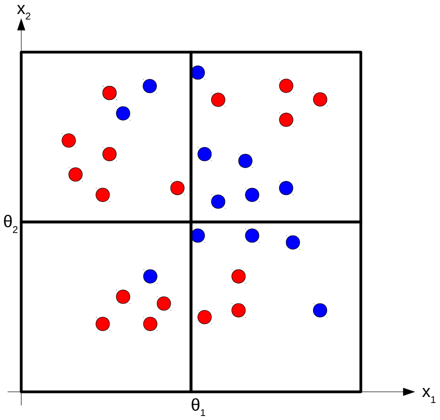
From Search Trees to (Random) Decision Trees

- Cell construction



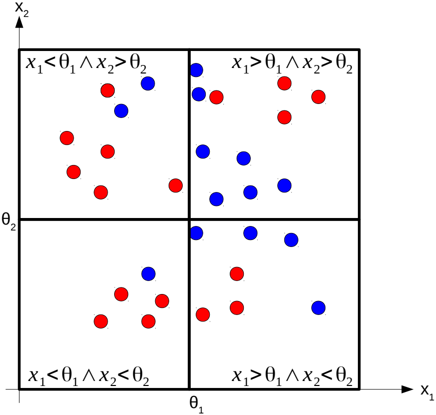
From Search Trees to (Random) Decision Trees

- Cell construction



From Search Trees to (Random) Decision Trees

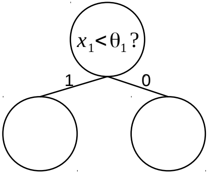
- Cell construction
  → Simple threshold operation
  → Different threshold definitions (e.g. equi-sized cells, threshold as data median) lead to different search tree variants (e.g. quad-tree, k-D tree).

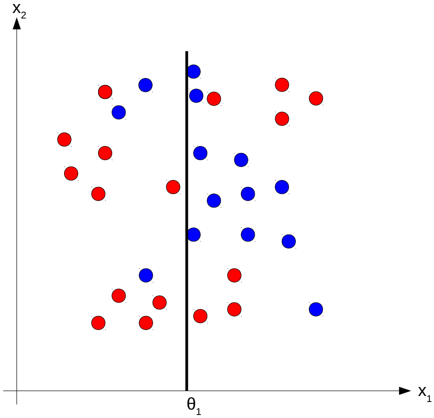


From Search Trees to (Random) Decision Trees

- Cell construction
  $\rightarrow$ Simple threshold operation
- Decision stump:

$$t(\mathbf{x}) = \begin{cases} 0 & \text{if } x_1 < \theta_1 \\ 1 & \text{otherwise.} \end{cases}$$

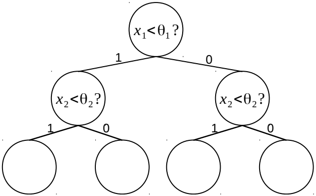


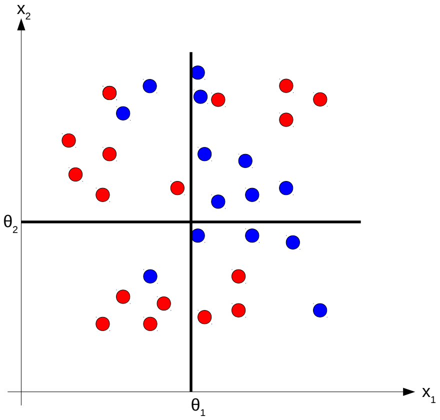


From Search Trees to (Random) Decision Trees

- Cell construction
  $\rightarrow$ Simple threshold operation
- Decision stump:

$$t(\mathbf{x}) = \begin{cases} 0 & \text{if } x_1 < \theta_1 \\ 1 & \text{otherwise.} \end{cases}$$

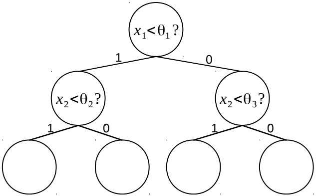


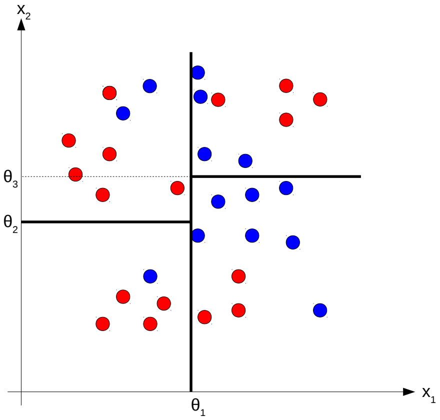


From Search Trees to (Random) Decision Trees

- Cell construction
  $\rightarrow$ Simple threshold operation
- Decision stump:

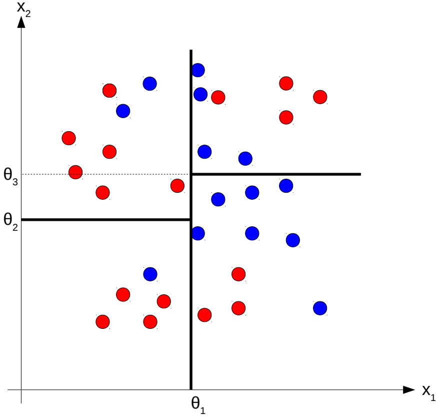
$$t(\mathbf{x}) = \begin{cases} 0 & \text{if } x_1 < \theta_1 \\ 1 & \text{otherwise.} \end{cases}$$



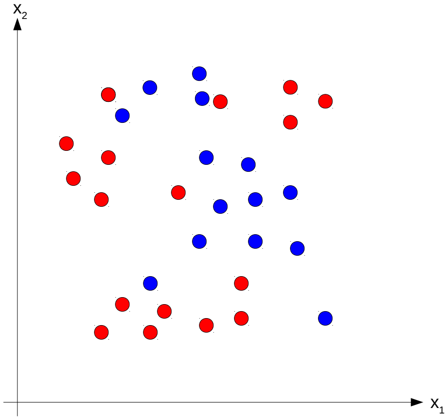


From Search Trees to (Random) Decision Trees

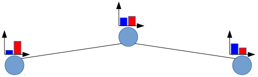
- Cell construction
  $\rightarrow$ Simple threshold operation
- Decision stump:
  $$t(\mathbf{x}) = \begin{cases} 0 & \text{if } x_1 < \theta_1 \\ 1 & \text{otherwise.} \end{cases}$$
- When to stop? Minimal resolution reached, purity, ...
- How to select split points? Randomly, optimized selection

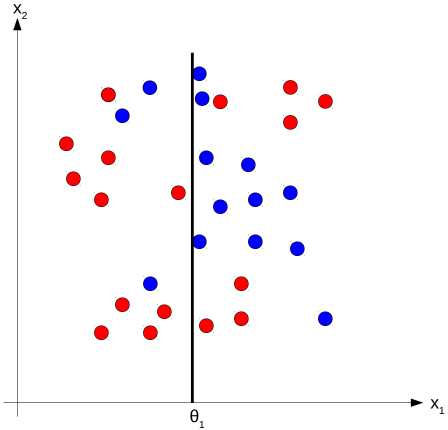


From Search Trees to (Random) Decision Trees

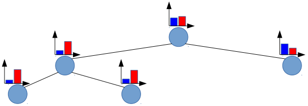


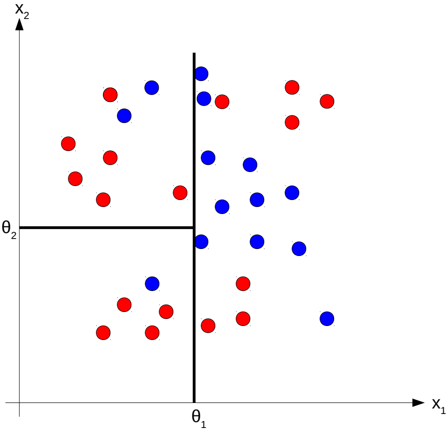
From Search Trees to (Random) Decision Trees



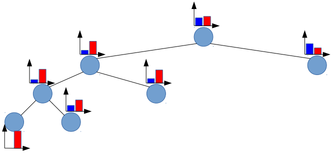


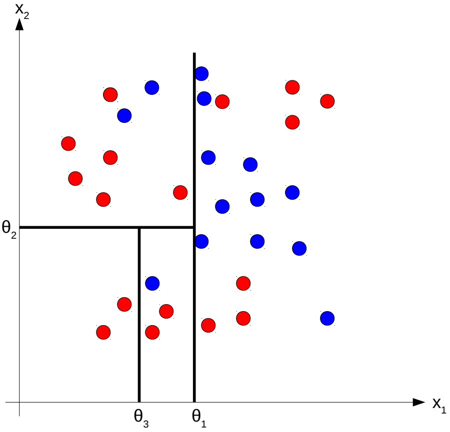
From Search Trees to (Random) Decision Trees



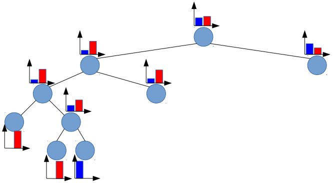


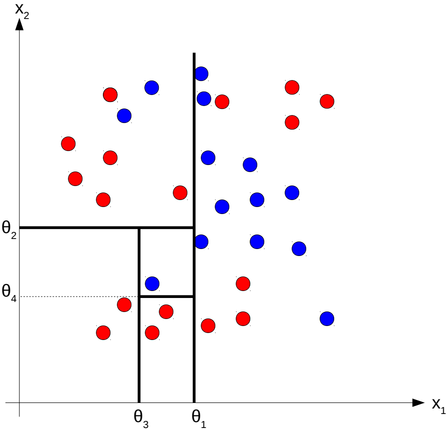
From Search Trees to (Random) Decision Trees



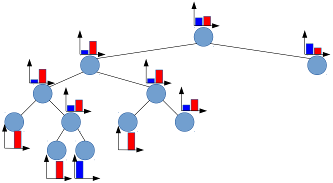


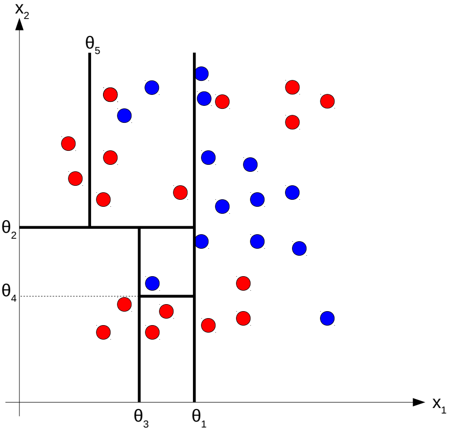
From Search Trees to (Random) Decision Trees



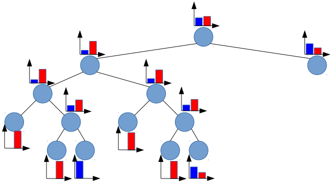


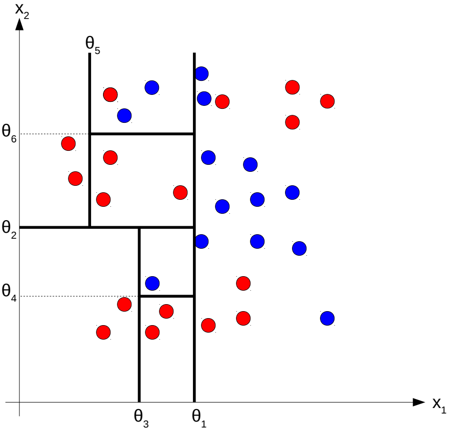
From Search Trees to (Random) Decision Trees



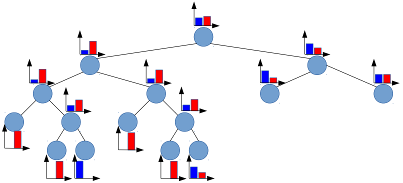


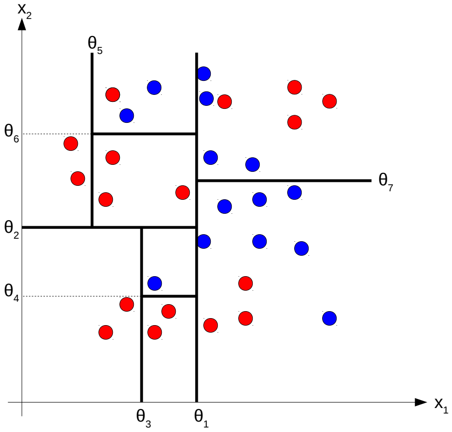
From Search Trees to (Random) Decision Trees



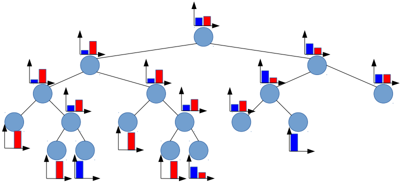


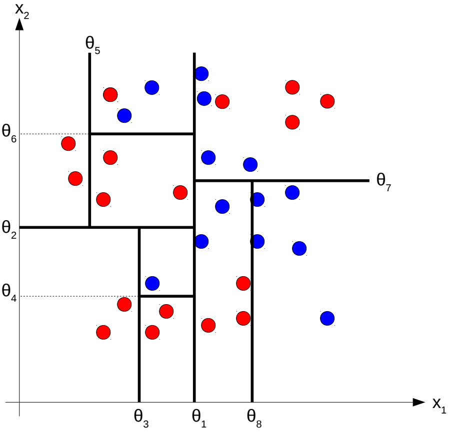
From Search Trees to (Random) Decision Trees



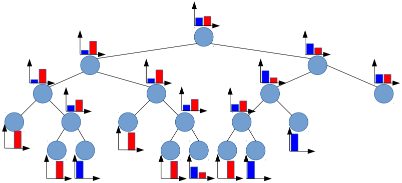


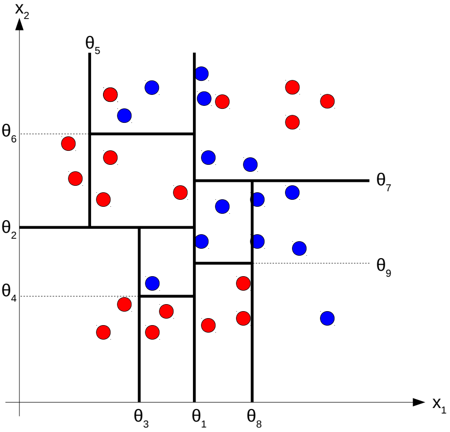
From Search Trees to (Random) Decision Trees



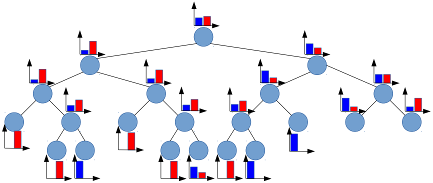


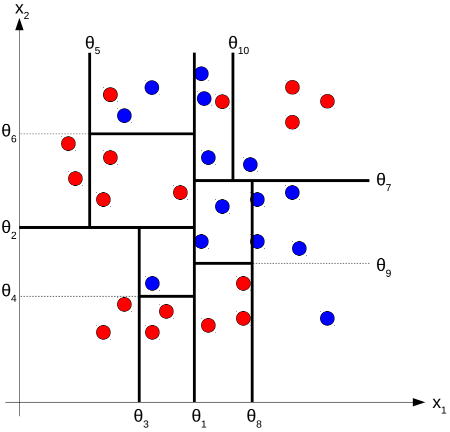
From Search Trees to (Random) Decision Trees



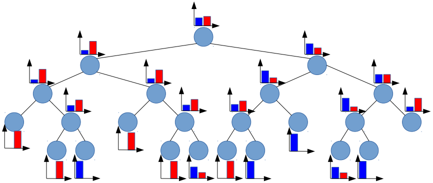


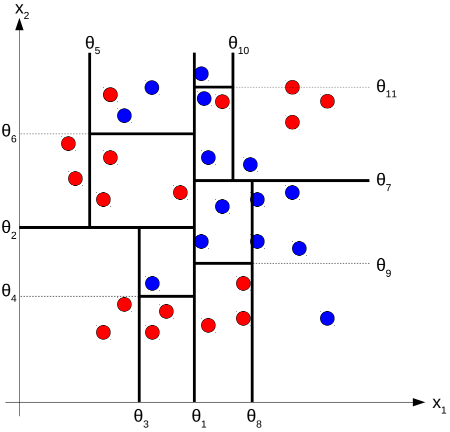
From Search Trees to (Random) Decision Trees



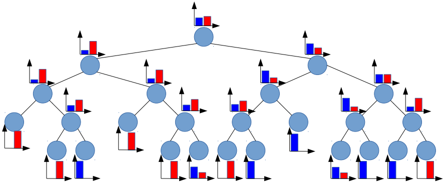


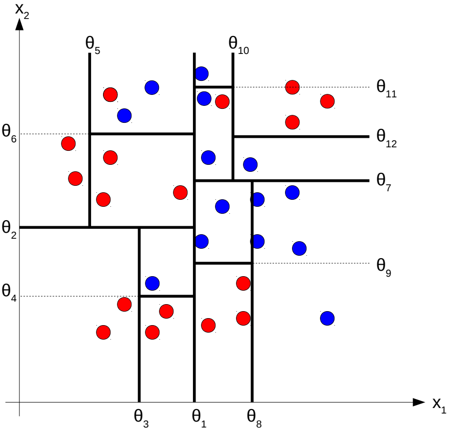
From Search Trees to (Random) Decision Trees





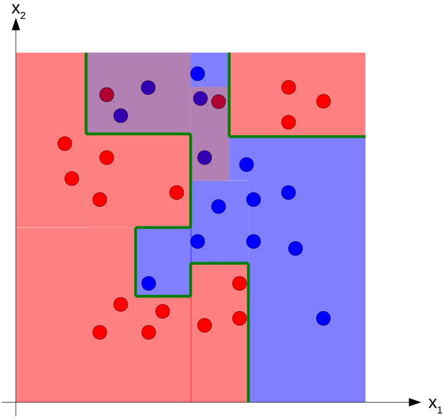
From Search Trees to (Random) Decision Trees





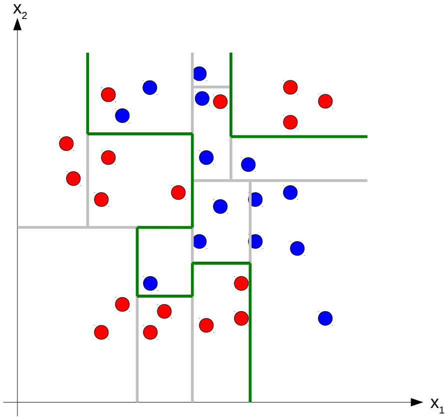
From Search Trees to (Random) Decision Trees

- Local estimate of the target variable (e.g. class posterior) is assigned to cells



From Search Trees to (Random) Decision Trees

- Local estimate of the target variable (e.g. class posterior) is assigned to cells
- Results in highly non-linear, even non-connected (but piece-wise constant) decision boundaries



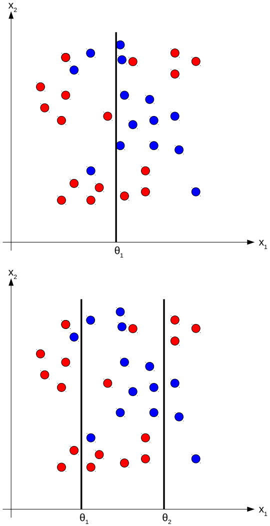
From Search Trees to (Random) Decision Trees

Other node tests are possible:

- Axis-aligned:

$$t(\mathbf{x}) = \begin{cases} 0 & \text{if } x_1 < \theta_1 \\ 1 & \text{otherwise.} \end{cases}$$

$$t(\mathbf{x}) = \begin{cases} 0 & \text{if } \theta_1 < x_1 < \theta_2 \\ 1 & \text{otherwise.} \end{cases}$$



From Search Trees to (Random) Decision Trees

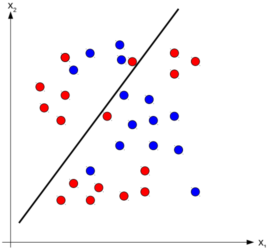
Other node tests are possible:

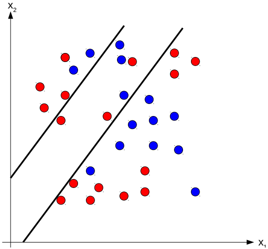
- Axis-aligned
- Linear:

  $\tilde{\mathbf{x}} = [\mathbf{x}, 1] \in \mathbb{R}^{d+1}, \psi \in \mathbb{R}^{d+1}$

  $$t(\mathbf{x}) = \begin{cases} 0 & \text{if } \psi^T \tilde{\mathbf{x}} < \theta_1 \\ 1 & \text{otherwise.} \end{cases}$$

  $$t(\mathbf{x}) = \begin{cases} 0 & \text{if } \theta_1 < \psi^T \tilde{\mathbf{x}} < \theta_2 \\ 1 & \text{otherwise.} \end{cases}$$



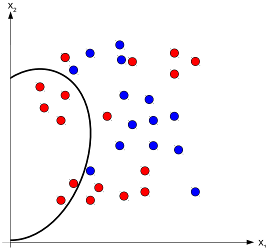


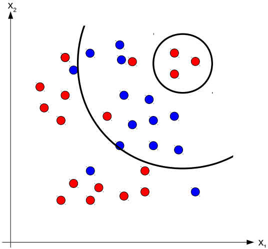
From Search Trees to (Random) Decision Trees

Other node tests are possible:

- Axis-aligned
- Linear
- Conic section:

$$
\begin{aligned}
\tilde{\mathbf{x}} &= [\mathbf{x}, 1] \in \mathbb{R}^{d+1}, \psi \in \mathbb{R}^{(d+1)\times(d+1)} \\
t(\mathbf{x}) &= \begin{cases} 0 & \text{if } \tilde{\mathbf{x}}^T \psi \tilde{\mathbf{x}} < \theta_1 \\ 1 & \text{otherwise.} \end{cases} \\
t(\mathbf{x}) &= \begin{cases} 0 & \text{if } \theta_1 < \tilde{\mathbf{x}}^T \psi \tilde{\mathbf{x}} < \theta_2 \\ 1 & \text{otherwise.} \end{cases}
\end{aligned}
$$



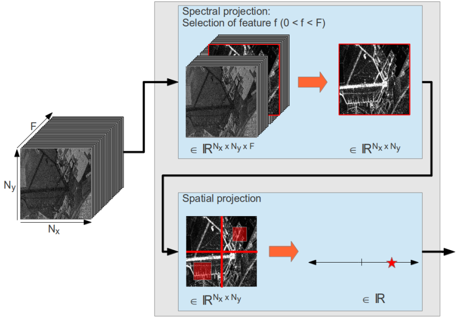


From Search Trees to (Random) Decision Trees

Other node tests are possible:

- Axis-aligned
- Linear
- Conic section
- Other data spaces than $\mathbb{R}^d$
  - PolSAR: $\mathbb{C}^3, \mathbb{C}^{3\times 3}$
  - Image patches: $\mathbb{R}^{n\times n}$
  - Non-scalar features, e.g. histograms, cardinal features such as pre-classification
  - ...

Spoiler alert
Part 3: ML and Images



From (Random) Decision Trees to Random Forests

Advantages

- Can deal with very heterogeneous data
  $\rightarrow$ Different, data-specific types of node tests
- Not prone to the curse of dimensionality
  $\rightarrow$ Each node only works on a very limited set of dimensions
- Very efficient
  $\rightarrow$ Each sample passes maximal H nodes (H = maximal height)
- Easy to implement
  $\rightarrow$ Binary trees are one of the most basic data structures
- Easy to interprete
  $\rightarrow$ Path through tree is a connected set of decision rules
- Well understood
  $\rightarrow$ Theoretical and practical implications of design decisions have been researched for more than 4 decades

From (Random) Decision Trees to Random Forests

Disadvantages

- Optimized by greedy algorithms
  $\rightarrow$ A chain of individually optimal decisions, might not lead to an overall optimum
- The optimal solution (i.e. decision boundary) might not be part of the model class (e.g. piece-wise linear and axis-aligned functions)
- Prone to overfitting
- Model capacity depends on amount of data
  $\rightarrow$ Few samples lead to small trees: Only few questions can be asked.
  $\rightarrow$ Many samples (might) lead to very high trees: Long processing times, large memory footprint.

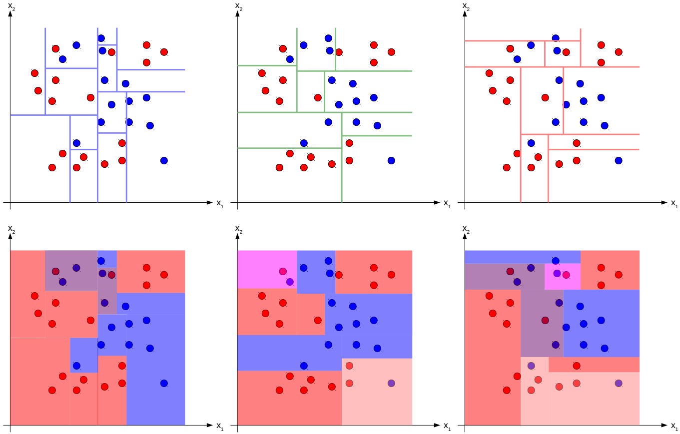
From (Random) Decision Trees to Random Forests

Disadvantages

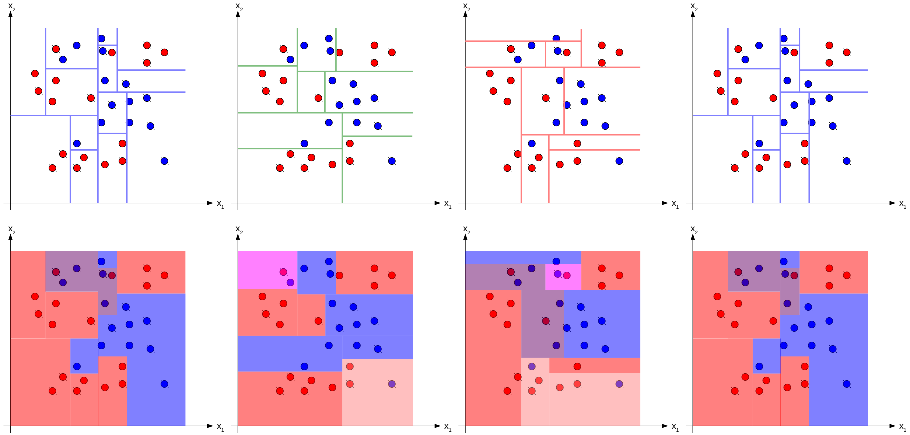
- Optimized by greedy algorithms
  $\rightarrow$ A chain of individually optimal decisions, might not lead to an overall optimum
- The optimal solution (i.e. decision boundary) might not be part of the model class (e.g. piece-wise linear and axis-aligned functions)
- Prone to overfitting
- Model capacity depends on amount of data
  $\rightarrow$ Few samples lead to small trees: Only few questions can be asked.
  $\rightarrow$ Many samples (might) lead to very high trees: Long processing times, large memory footprint.

How to
$\rightarrow$ keep (most) of the advantages
$\rightarrow$ getting rid of (most) disadvantages?

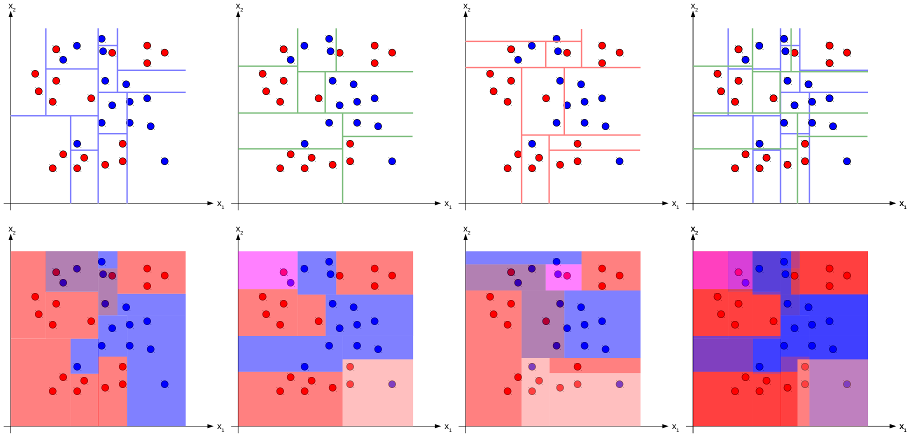
From (Random) Decision Trees to Random Forests



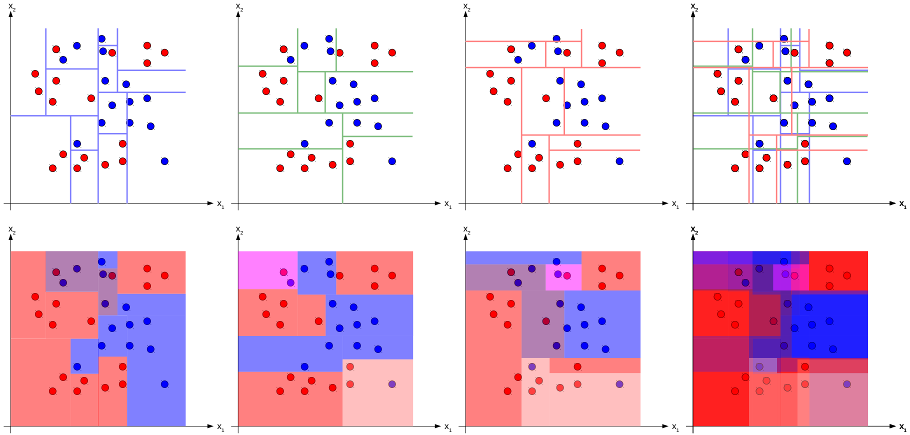
From (Random) Decision Trees to Random Forests



From (Random) Decision Trees to Random Forests



From (Random) Decision Trees to Random Forests



Random Forests

- Many (suboptimal) baselearners, i.e. decision trees
- Fusion of the individual output
- Minimization of the risk to use wrong model
- Extension of the model space
- Decreased dependence on initialization
- One name to rule them all
  - Bagged Decision Trees
  - Randomized Trees
  - Decision Forests
  - ERT, PERT, Rotation Forests, Hough Forests, Semantic Texton Forests, ...

Random Forests - Randomization through node tests

Before: $t(\mathbf{x}) = \begin{cases} 0 & \text{if } x_1 < \theta_1 \\ 1 & \text{otherwise.} \end{cases}$

Now: More general

$\rightarrow$ Concatenation of several functions with different tasks

$$t_\tau : \mathbb{D} \rightarrow \{0, 1\} \qquad \tau \in \mathrm{T} \equiv \text{Parameter set}$$

$$t_\tau = \xi \circ \psi \circ \phi$$

$$\begin{aligned}
\phi : \mathbb{D} \rightarrow \mathbb{R}^n &\equiv \text{Implicit feature extraction} \\
&\quad \text{e.g. } x \in \mathbb{R}^n : \ \phi : \mathbb{R}^n \rightarrow \mathbb{R}^2, x \mapsto (x_i, x_j)^T \\
\psi : \mathbb{R}^n \rightarrow \mathbb{R} &\equiv \text{Feature fusion} \\
&\quad \text{e.g. } \phi(x) \in \mathbb{R}^2 : \ \psi : \mathbb{R}^2 \rightarrow \mathbb{R}, \phi(x) \mapsto [\psi_i, \psi_j] \cdot \phi(x) \\
\xi : \mathbb{R} \rightarrow \{0, 1\} &\equiv \text{Child node assignment} \\
&\quad \text{e.g. thresholding}
\end{aligned}$$

Decision trees perform exhaustive search for optimal parameters τ in T
Random Forests use random subset $\tilde{\mathrm{T}}$ (Note: $|\tilde{\mathrm{T}}| = 1$ possible)

Random Forests - Randomization through Bagging

Given: Training set $\mathrm{D} \subset \mathbb{D}$ with $|\mathrm{D}| = N$ samples.
Bagging (**B**ootstrap **agg**regat**ing**):

1. Randomly sample M data sets D_m with replacement ($|\mathrm{D}_m| = N$).
2. Train M models where m-th model has only access to m-th dataset.
3. Average all models.

- Meta learning technique
- Works if small change in input data leads to large model variation
- Reduces variance (of final model), avoids overfitting.
- Leads to diverse decision trees, even if all other parameters are fixed

Random Forests - Key questions

- What kind of node tests?
  $\rightarrow$ For images, for other data spaces than $\mathbb{R}^n$
- How to select node tests?
  $\rightarrow$ How to measure good tests?
- What kind of target variables?
  $\rightarrow$ More than a single class label?
- How to limit model capacity (tree height, tree number)?
  $\rightarrow$ The more the better? What about overfitting?
- How to fuse tree decisions?
  $\rightarrow$ Whom to trust?
- How to interprete results?
  $\rightarrow$ Tree properties and visualization.

Glorot, X. and Bengio, Y. (2010). Understanding the difficulty of training deep feedforward neural networks. In *Proceedings of the Thirteenth International Conference on Artificial Intelligence and Statistics, AISTATS 2010, Chia Laguna Resort, Sardinia, Italy, May 13-15, 2010*, pages 249–256.

Goodfellow, I., Bengio, Y., and Courville, A. (2016). *Deep Learning*. MIT Press. http://www.deeplearningbook.org.

Ioffe, S. and Szegedy, C. (2015). Batch normalization: Accelerating deep network training by reducing internal covariate shift. *CoRR*, abs/1502.03167.

Keskar, N. S., Mudigere, D., Nocedal, J., Smelyanskiy, M., and Tang, P. T. P. (2016). On large-batch training for deep learning: Generalization gap and sharp minima. *CoRR*, abs/1609.04836.

Kingma, D. P. and Ba, J. (2014). Adam: A method for stochastic optimization. *CoRR*, abs/1412.6980.

Krizhevsky, A., Sutskever, I., and Hinton, G. E. (2012). Imagenet classification with deep convolutional neural networks. In Pereira, F., Burges, C. J. C., Bottou, L., and Weinberger, K. Q., editors, *Advances in Neural Information Processing Systems 25*, pages 1097–1105. Curran Associates, Inc.

Glorot, X. and Bengio, Y. (2010). Understanding the difficulty of training deep feedforward neural networks. In *Proceedings of the Thirteenth International Conference on Artificial Intelligence and Statistics, AISTATS 2010, Chia Laguna Resort, Sardinia, Italy, May 13-15, 2010*, pages 249–256.

Goodfellow, I., Bengio, Y., and Courville, A. (2016). *Deep Learning*. MIT Press. http://www.deeplearningbook.org.

Ioffe, S. and Szegedy, C. (2015). Batch normalization: Accelerating deep network training by reducing internal covariate shift. *CoRR*, abs/1502.03167.

Keskar, N. S., Mudigere, D., Nocedal, J., Smelyanskiy, M., and Tang, P. T. P. (2016). On large-batch training for deep learning: Generalization gap and sharp minima. *CoRR*, abs/1609.04836.

Kingma, D. P. and Ba, J. (2014). Adam: A method for stochastic optimization. *CoRR*, abs/1412.6980.

Krizhevsky, A., Sutskever, I., and Hinton, G. E. (2012). Imagenet classification with deep convolutional neural networks. In Pereira, F., Burges, C. J. C., Bottou, L., and Weinberger, K. Q., editors, *Advances in Neural Information Processing Systems 25*, pages 1097–1105. Curran Associates, Inc.