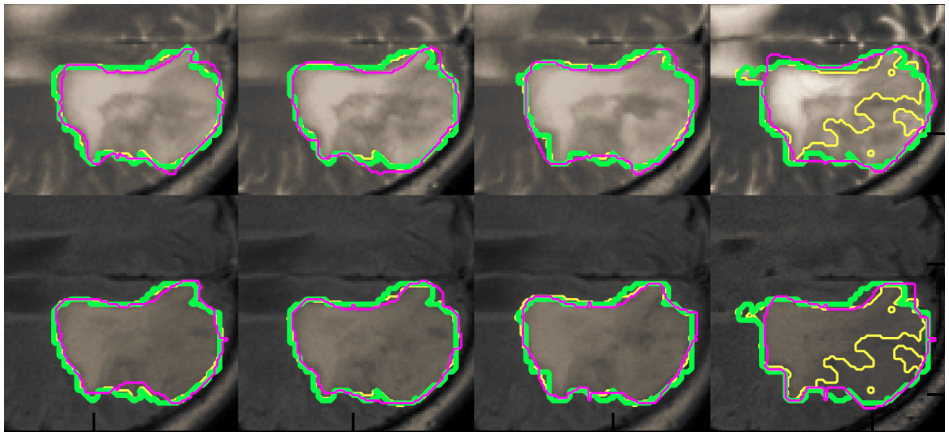
Optimisation and Principal Algorithms of Machine Learning

Yuliya Tarabalka

LuxCarta Technology
Inria Sophia Antipolis-Méditerranée - TITANE team
Université Côte d'Azur - France

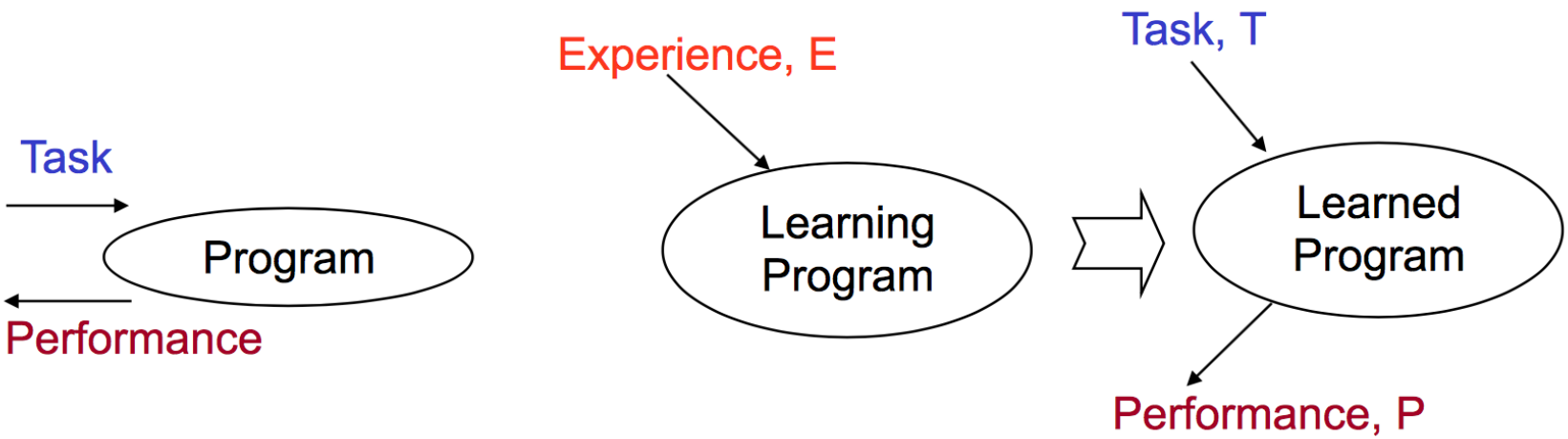


What is machine learning?

- Term introduced by Arthur Samuel in 1959:
  - Machine learning gives computers the ability to learn without being explicitly programmed
- Evolved from the study of pattern recognition and computational learning theory
- Machine learning explores the study and construction of algorithms that can learn from and make predictions on data

Machine learning formal definition

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



Main categories of tasks

- **Supervised learning**:
  - The computer is presented with example inputs and desired outputs = **training data** $D = \{(\mathbf{x}, y)\}$
  - The goal is to learn a general rule f_w that maps inputs to outputs $f_w(\mathbf{x}) = y$
  - **Classification**: y is discrete
  - **Regression**: y is continuous

Image classification

Classification:

$\Rightarrow$ "Airplane"

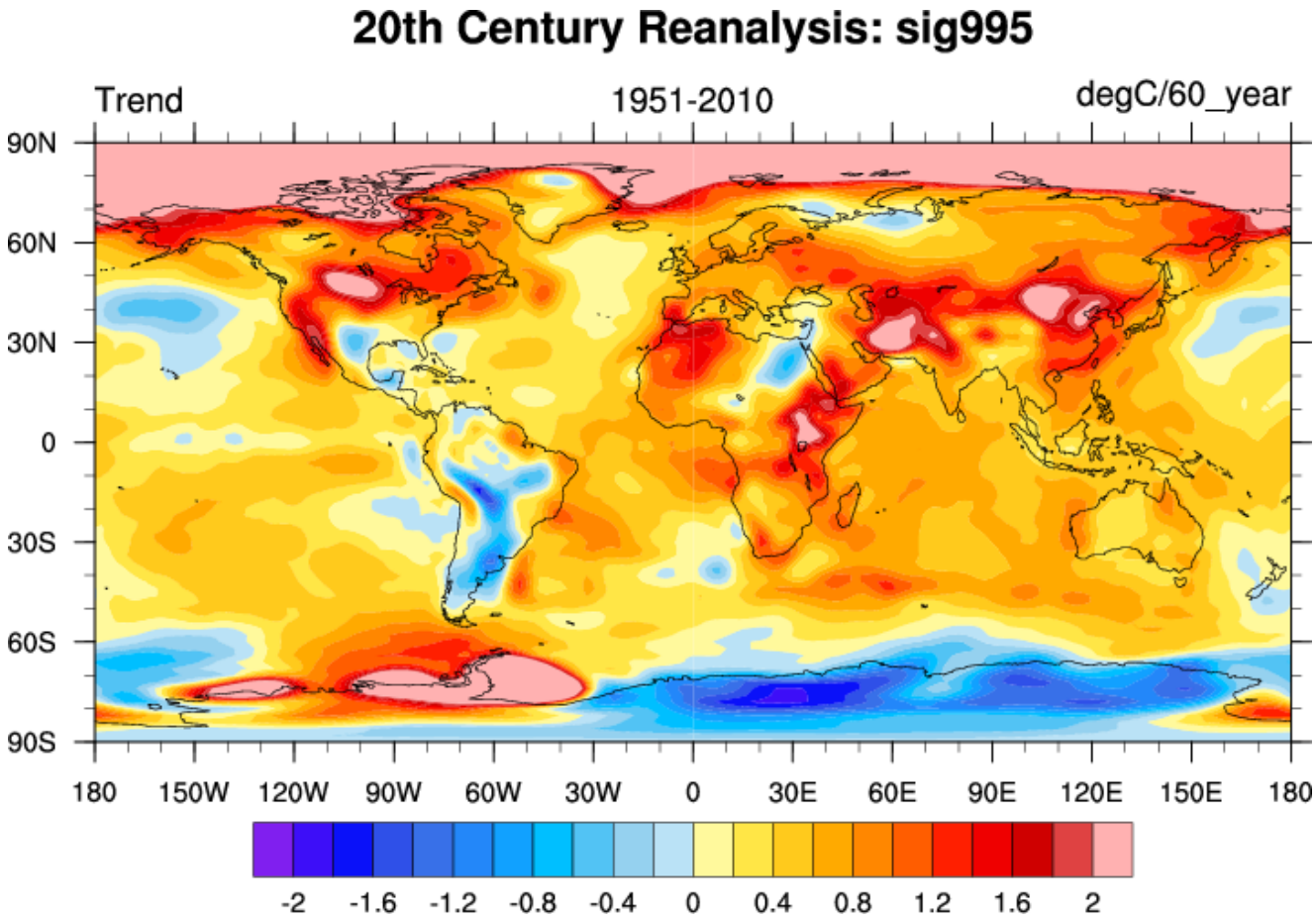
Semantic segmentation, dense labeling:

$\Rightarrow$

Input Output

Regression

- **Example of regression:** map of annual mean temperatures

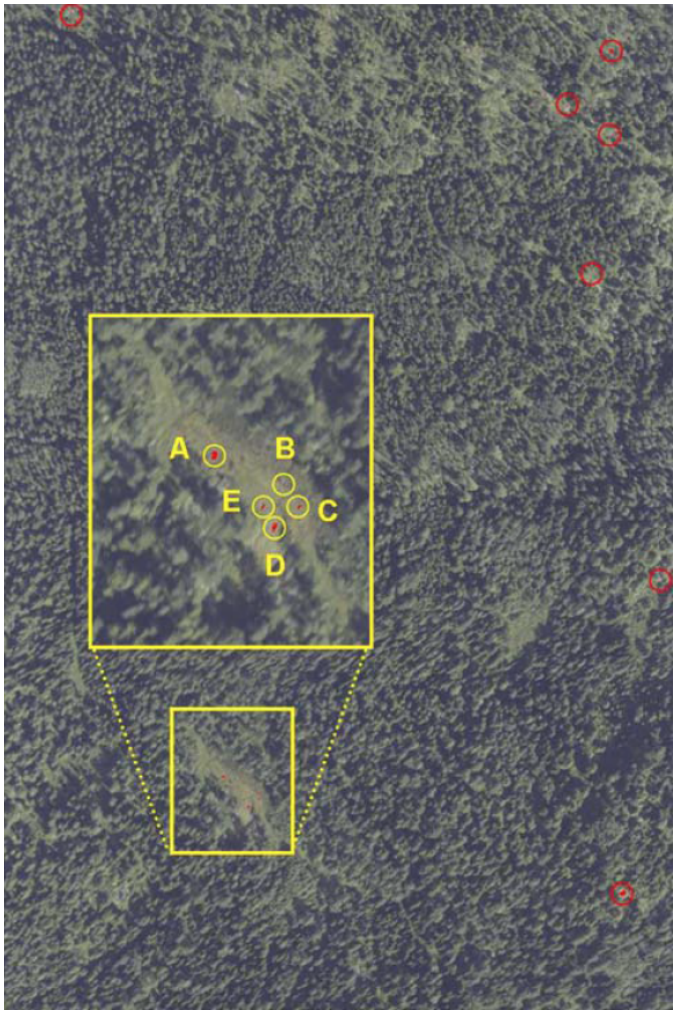


Main categories of tasks

- **Unsupervised learning**:
  - No labels are given to the learning algorithm
  - Goal: find structure in its input
  - Used to discover hidden patterns in data, learn features
  - **Clustering**

Unsupervised learning

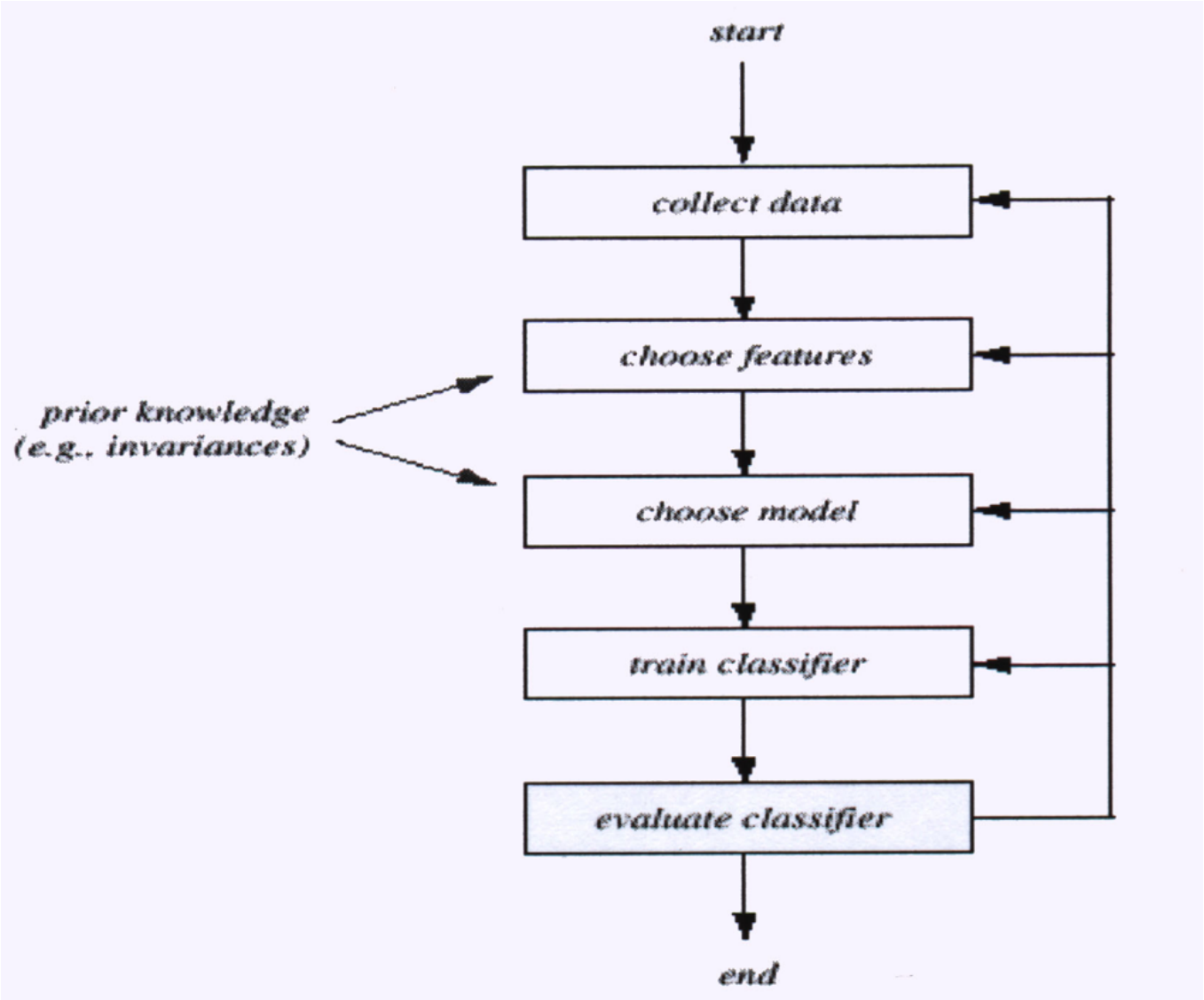
- Example: **Anomaly detection**:
  - **Estimate** a background model by fitting a multivariate normal mixture model to a spatial subset of the image
  - **Evaluate** a background probability value for all pixels in the image based on the estimated model
  - **Detect** anomalous pixels by thresholding on low background probability values



Main categories of tasks

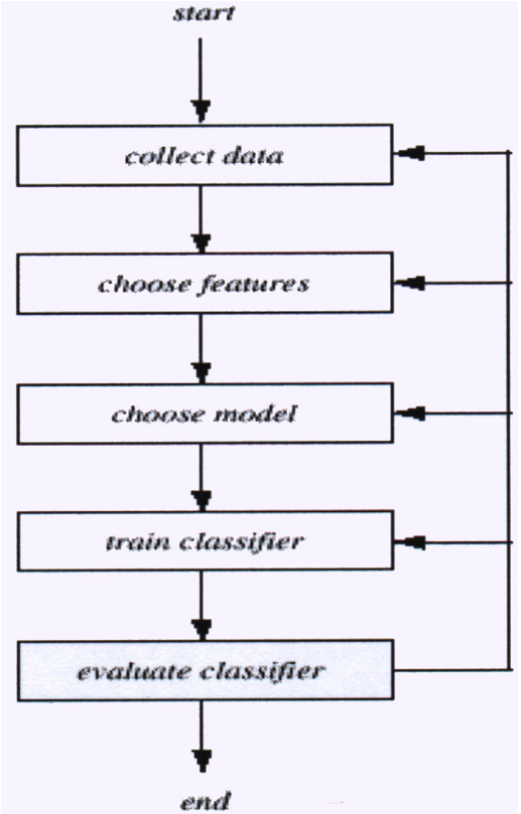
- **Domain adaptation**:
  - We aim at learning from a source data distribution a well performing model on a different (but related) target data distribution
  - Example: images taken on different dates, with different sensors
  - Useful for large-scale classification

Classical design cycle



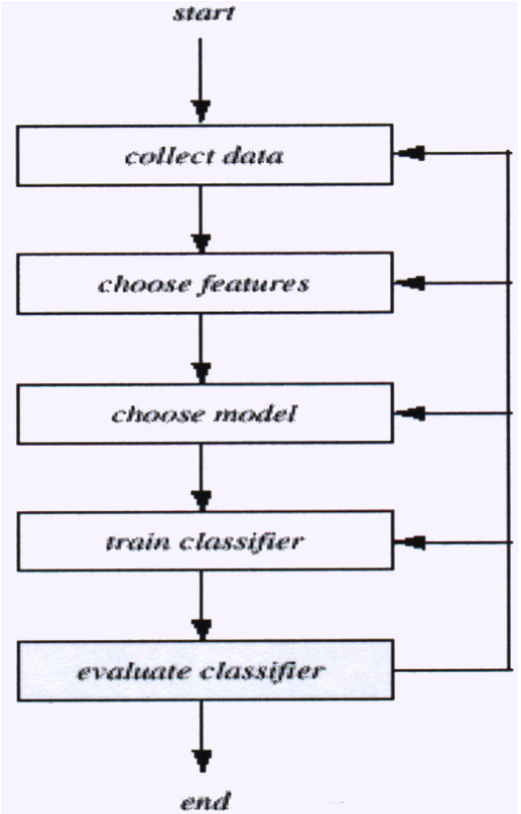
Classical design cycle

- Data collection
  - How do we know when we have collected an adequately large & representative set of examples to train/test the system?



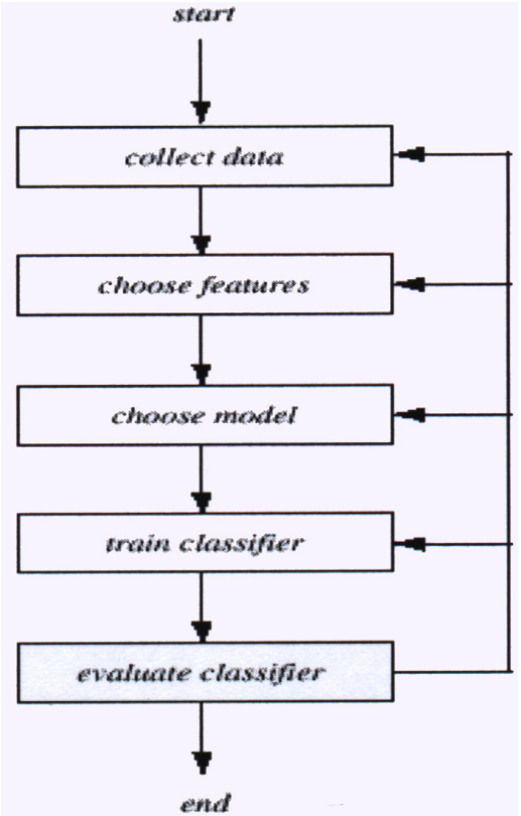
Classical design cycle

- Data collection
  - How do we know when we have collected an adequately large & representative set of examples to train/test the system?
- Feature choice
  - Depends on the characteristics of the problem domain
  - Preferably:
    - Simple to extract
    - Insensitive to noise
    - Useful to discriminate patterns in different categories
    - ...



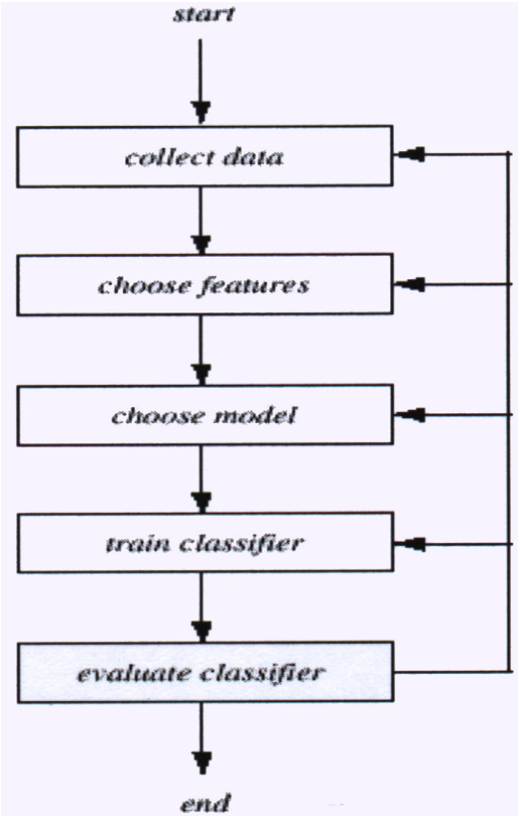
Classical design cycle

- Model choice
  - Depends on the characteristics of the problem domain
  - Often useful to test several models
- Training
  - Training set must be representative
  - Importance of cross validation / separate datasets



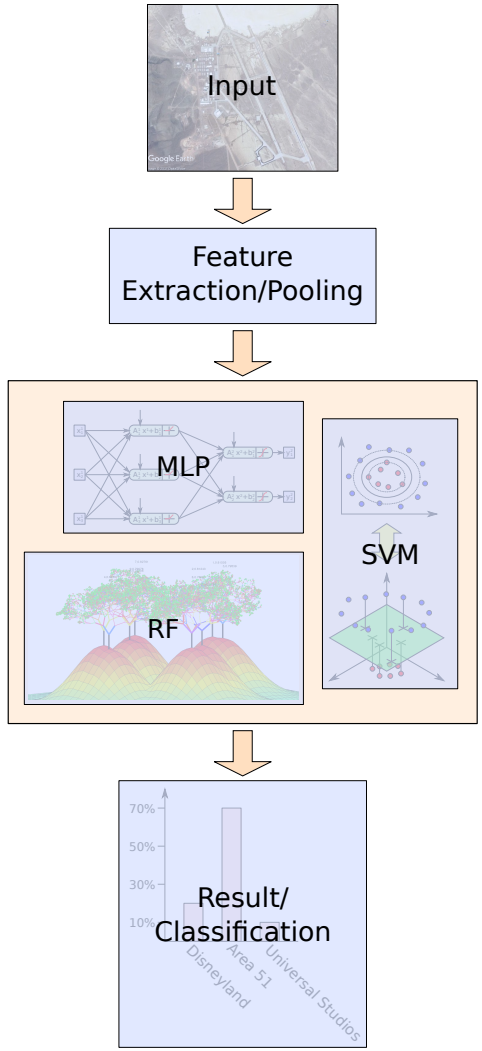
Classical design cycle

- Evaluation
  - Measure the error rate (or performance)
  - Test different set of features/models to compare performace
- Computational Complexity
  - What is the trade off between computational ease and performance?
  - How an algorithm scales as a function of the number of features or categories?



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



Bayesian decision theory

- Fundamental statistical approach to the problem of pattern classification
- Assumptions:
  1. Decision problem is posed in probabilistic terms
  2. Ideal case: probability structure underlying the categories is known perfectly

Statistical decision theory

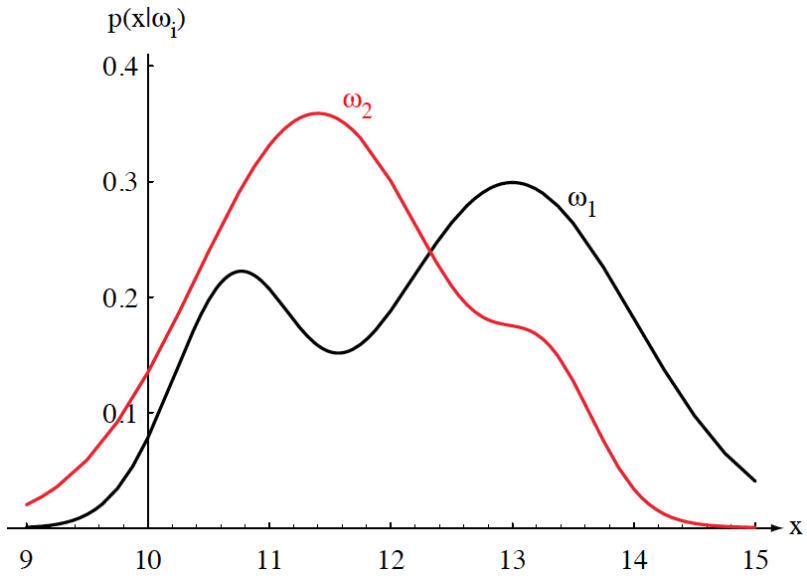
- Fish classification example:
  - Sort incoming fish on a conveyor according to species using optical sensing
  - 2 species: sea bass & salmon
- State of nature is unpredictable
  - We denote state of nature by a random variable ω
  - $\omega = \omega_1$ for sea bass; $\omega = \omega_2$ for salmon
- Prior probability
  - Reflect our prior knowledge of how likely the next fish is sea bass/salmon
  - $P(\omega_1) = P(\omega_2) \rightarrow$ the catch of sea bass/salmon is equiprobable

Statistical decision theory

- Exclusivity and exhaustivity:
  - If only two species, $P(\omega_1) + P(\omega_2) = 1$
- Decision rule with only the prior information
  - Decide ω_1 if $P(\omega_1) > P(\omega_2)$
  - Otherwise, decide ω_2
- Impossible to make good classifier with so little information

Statistical decision theory

- Use class-conditional information
- Example: we might use a lightness measurement x to improve our classifier
- $p(x|\omega_1)$ and $p(x|\omega_2)$ are **class-conditional probability density** functions
  - Describe the difference in lightness between populations of sea bass and salmon



Bayes' formula

$$P(\omega_j|x) = \frac{p(x|\omega_j)P(\omega_j)}{p(x)}$$

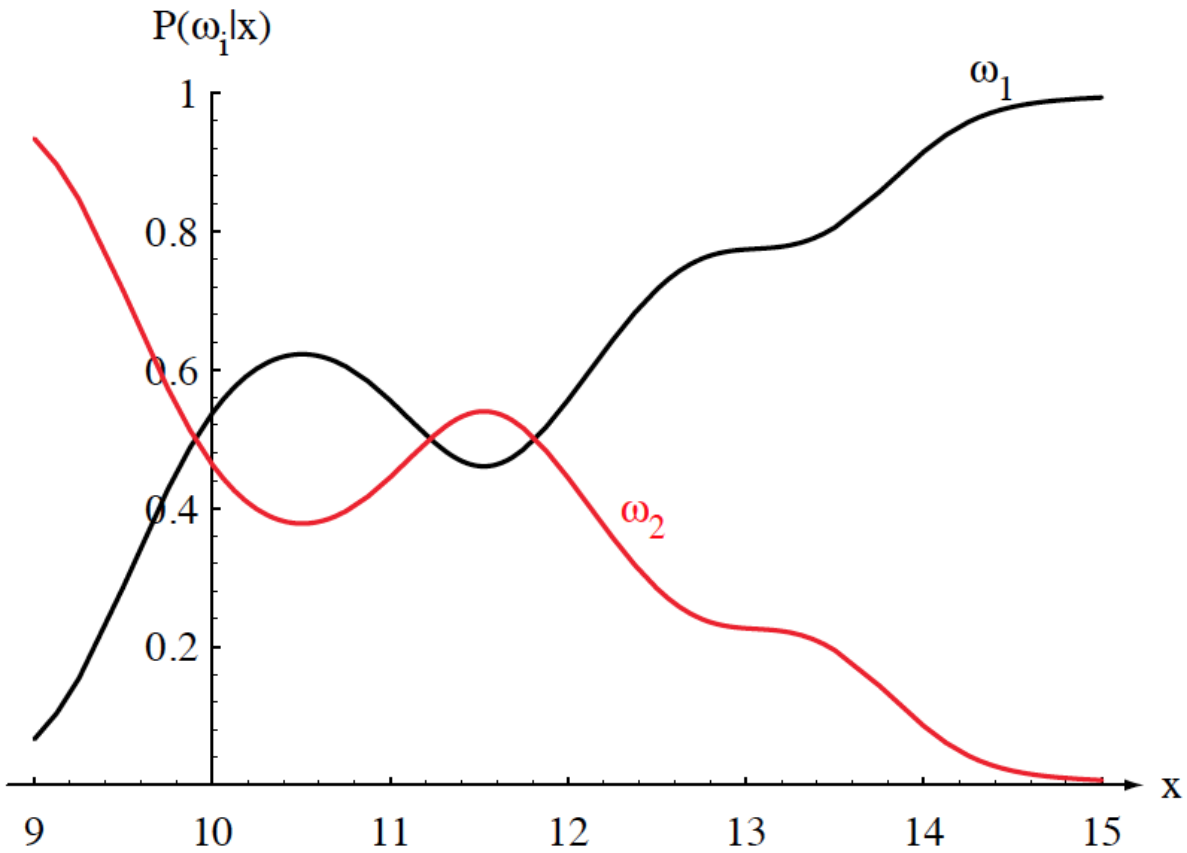
- Where in case of two categories:

$$p(x) = \sum_{j=1}^{2} p(x|\omega_j)P(\omega_j)$$

- $p(x)$ can be seen as a scale factor, guaranteeing that the posterior probabilitiess sum to 1
- Bayes' formula in words:

$$posterior = \frac{likelihood \times prior}{evidence}$$

Posterior probabilities - example



- At every x, the posteriors sum to 1

Decision given the posterior probabilities

- Given an observation x:
  - If $P(\omega_1|x) > P(\omega_2|x) \Rightarrow$ we will choose ω_1
  - If $P(\omega_2|x) > P(\omega_1|x) \Rightarrow$ we will choose ω_2

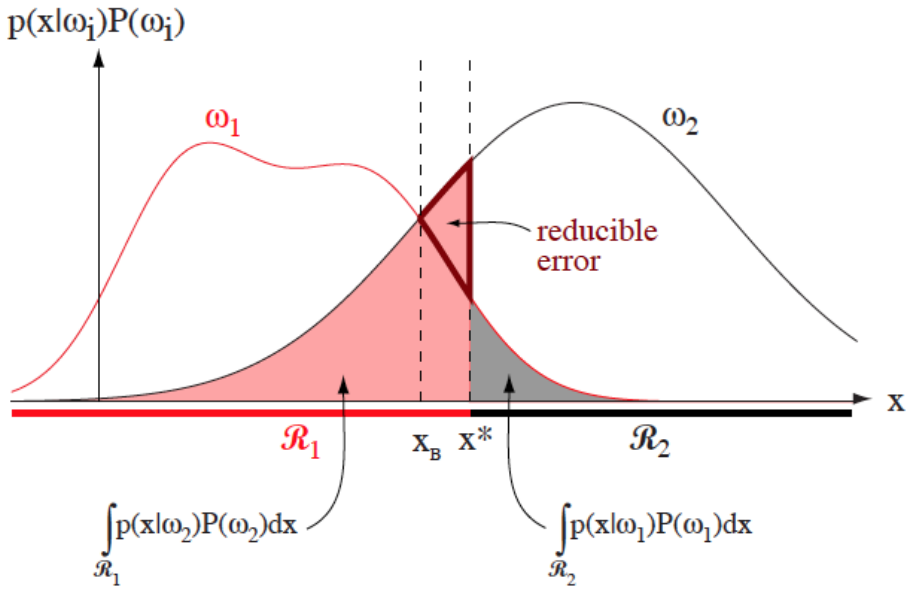
- Therefore, whenever we observe a particular x, the probability of error:

$$P(error|x) = \begin{cases} P(\omega_1|x), & \text{if we decide } \omega_2, \\ P(\omega_2|x), & \text{if we decide } \omega_1 \end{cases}$$

Bayes decision rule

- Minimizing the probability of error:
  - Decide ω_1 if $P(\omega_1|x) > P(\omega_2|x)$
  - Otherwise decide ω_2
- Probability of error:

$$P(error|x) = \min[P(\omega_1|x), P(\omega_2|x)]$$



Generalization of the preceding ideas

- Use of more than one feature
  - **Feature vector** $\mathbf{x}$ lies in a d-dimentional **feature space**
- Use more than two states of nature
- Allow actions and not only decide on the state of nature
- Introduce a loss function which is more general than the probability of error

Loss function

- Allowing actions other than classification allows the possibility of rejection
  - Refuse to make a decision in close or bad cases
- **Loss function** states how costly each action is
  - Let $\omega_1, ..., \omega_c$ be the set of c categories (classes, states of nature)
  - Let $\alpha_1, ..., \alpha_a$ be the set of a possible actions
  - $\lambda(\alpha_i|\omega_j)$ describes the loss incurred for taking action α_i when the category is ω_j

Bayes risk

- **Decision rule** is a function $\alpha(\mathbf{x})$ assuming one of the a values $\alpha_1, ..., \alpha_a$
- **Overall risk** R is the expected loss associated with a given decision rule:

$$R = \int R(\alpha(\mathbf{x})|\mathbf{x})p(\mathbf{x})d\mathbf{x},$$

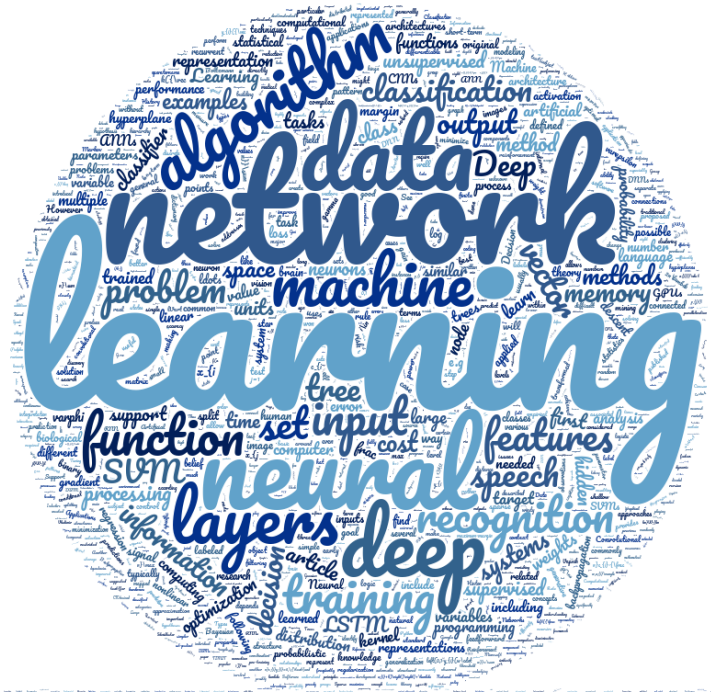
where the **conditional risk:**

$$R(\alpha_i|\mathbf{x}) = \sum_{j=1}^{c} \lambda(\alpha_i|\omega_j)P(\omega_j|\mathbf{x})$$

- **Bayes decision rule:** To minimize R, select the action α_i for which $R(\alpha_i|\mathbf{x})$ is minimum

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

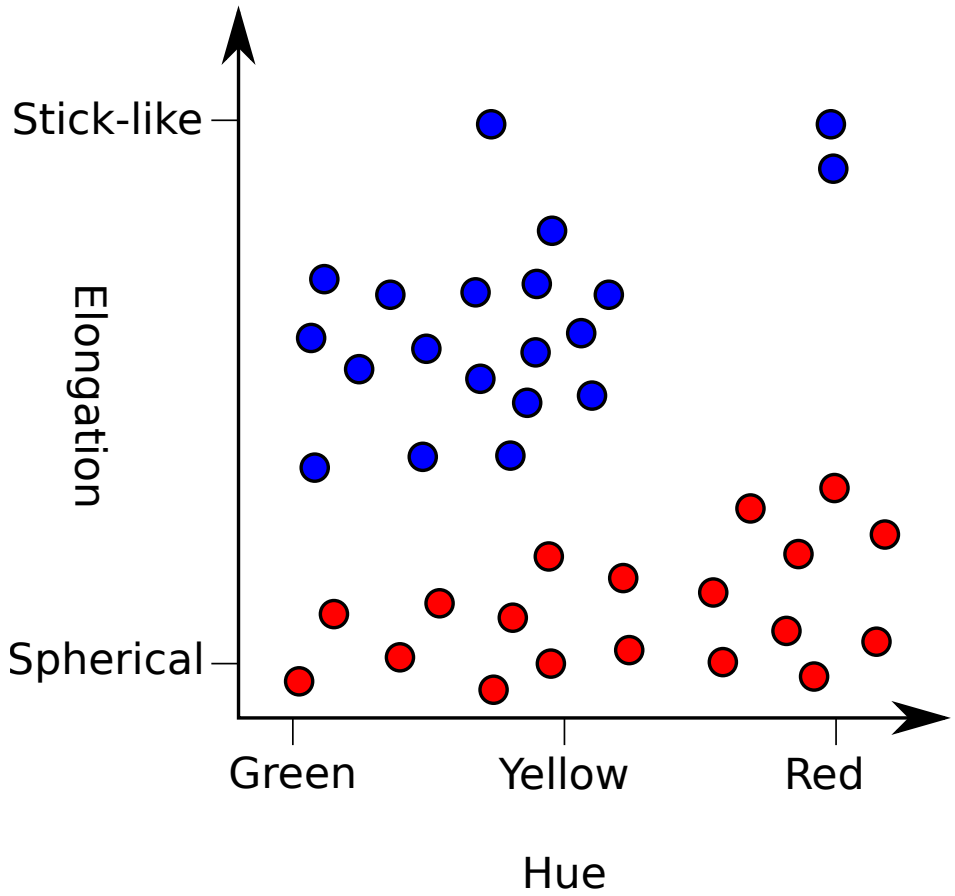
- Hue (yellow to red)
- Elongation (max extend over min extend)

image by Darkone licensed under CC BY SA 2.0

image by Abhijit Tembhekar licensed under CC BY 2.0

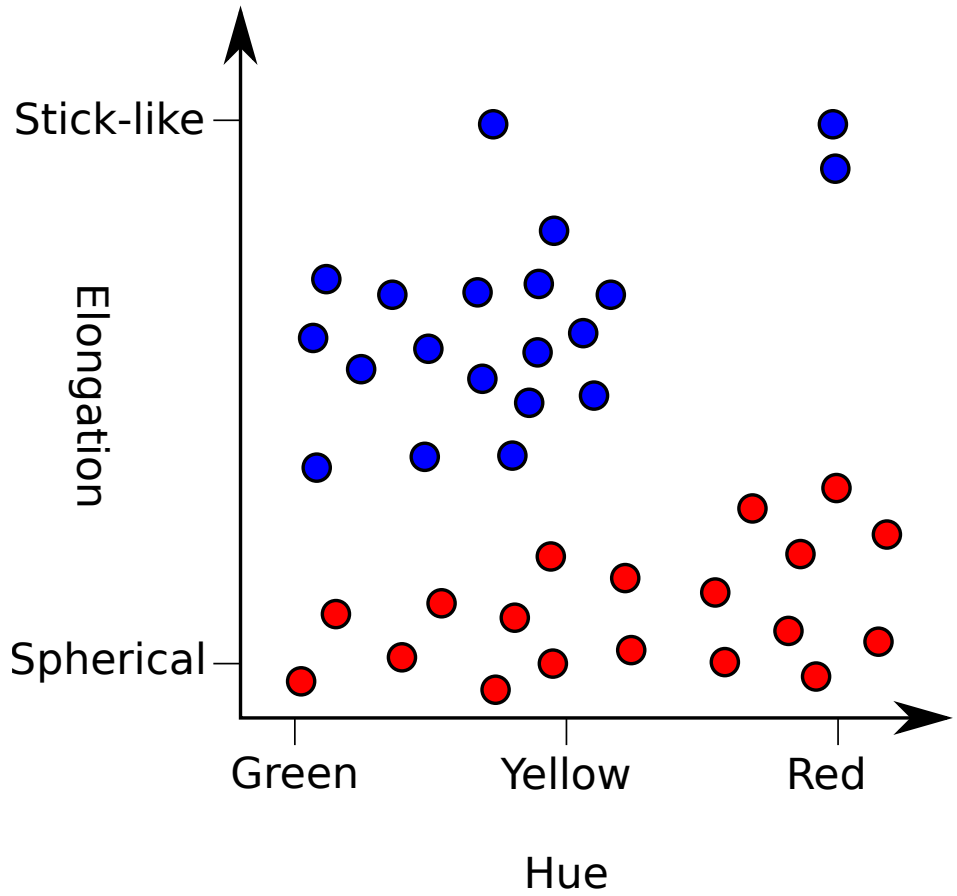
Some training data

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



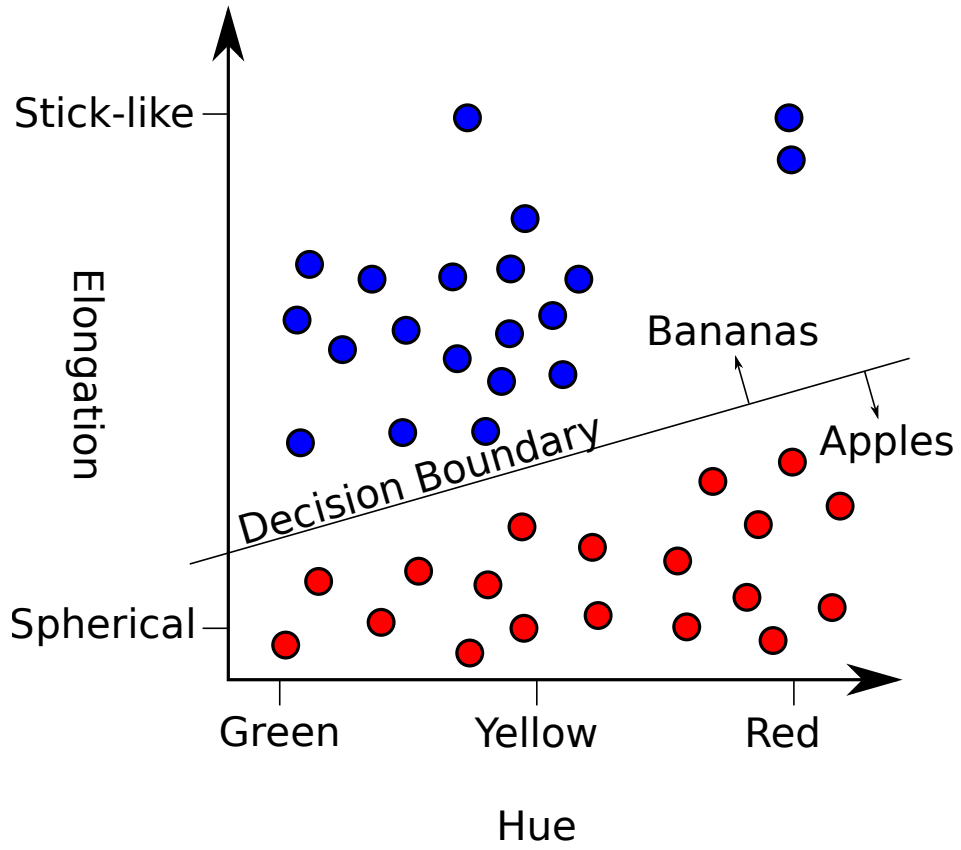
Some training data

- 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?



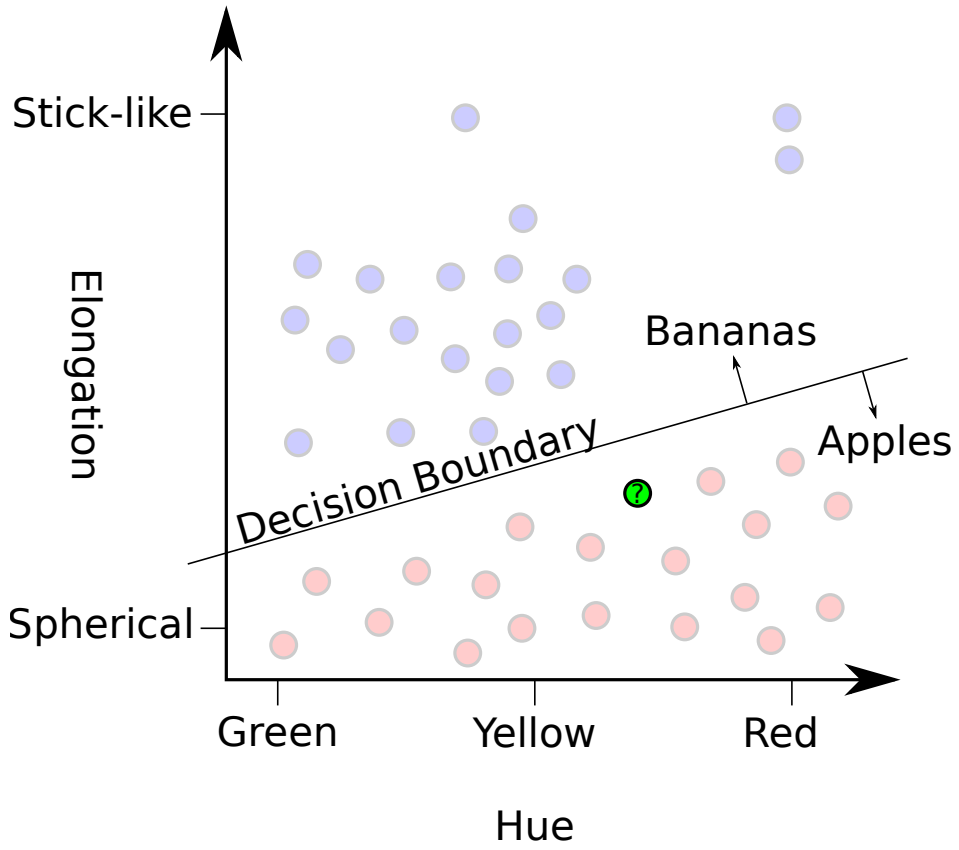
Decision boundary

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



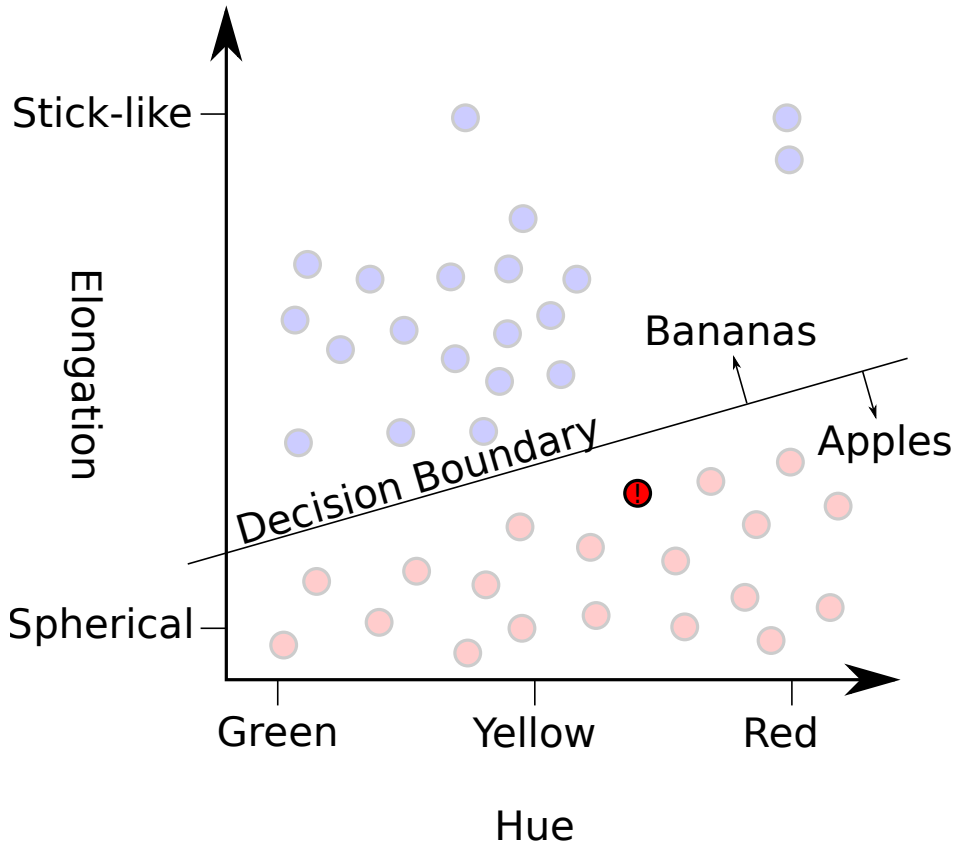
Decision boundary

- (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

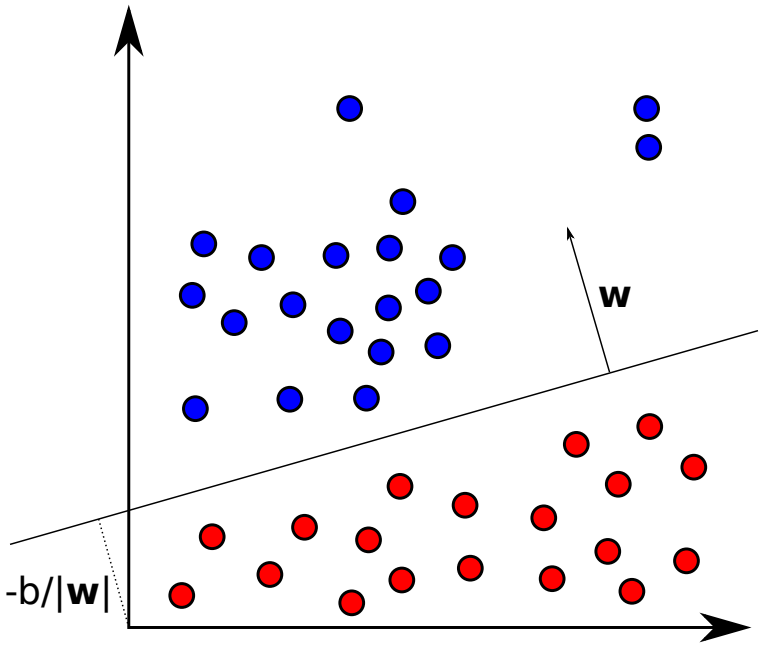


Perceptron

Perceptron

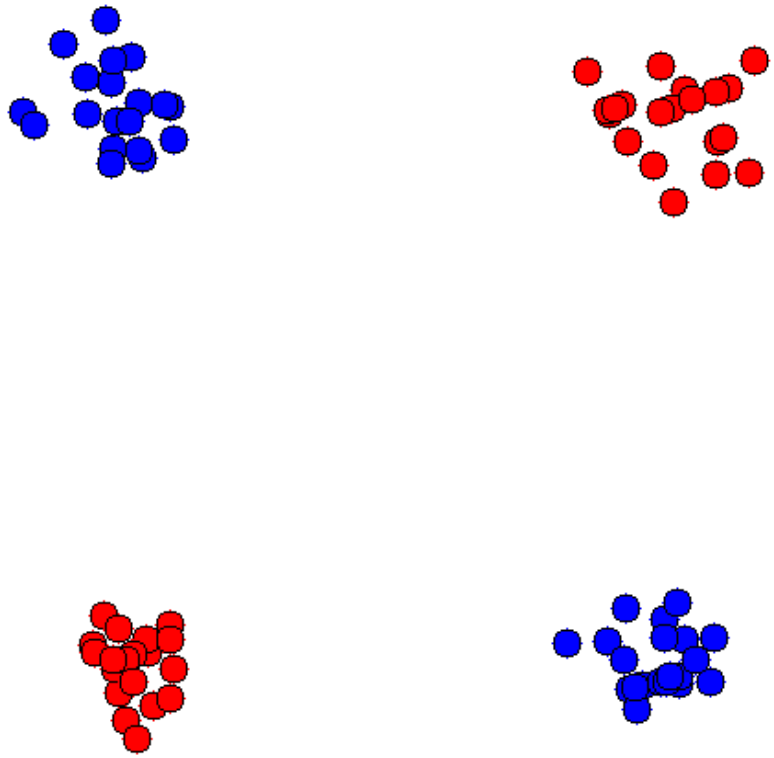
$$y = \text{sign}(\mathbf{w}^T \mathbf{x} + b) \quad (1)$$

- $y \in \{-1, 1\}$: 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)



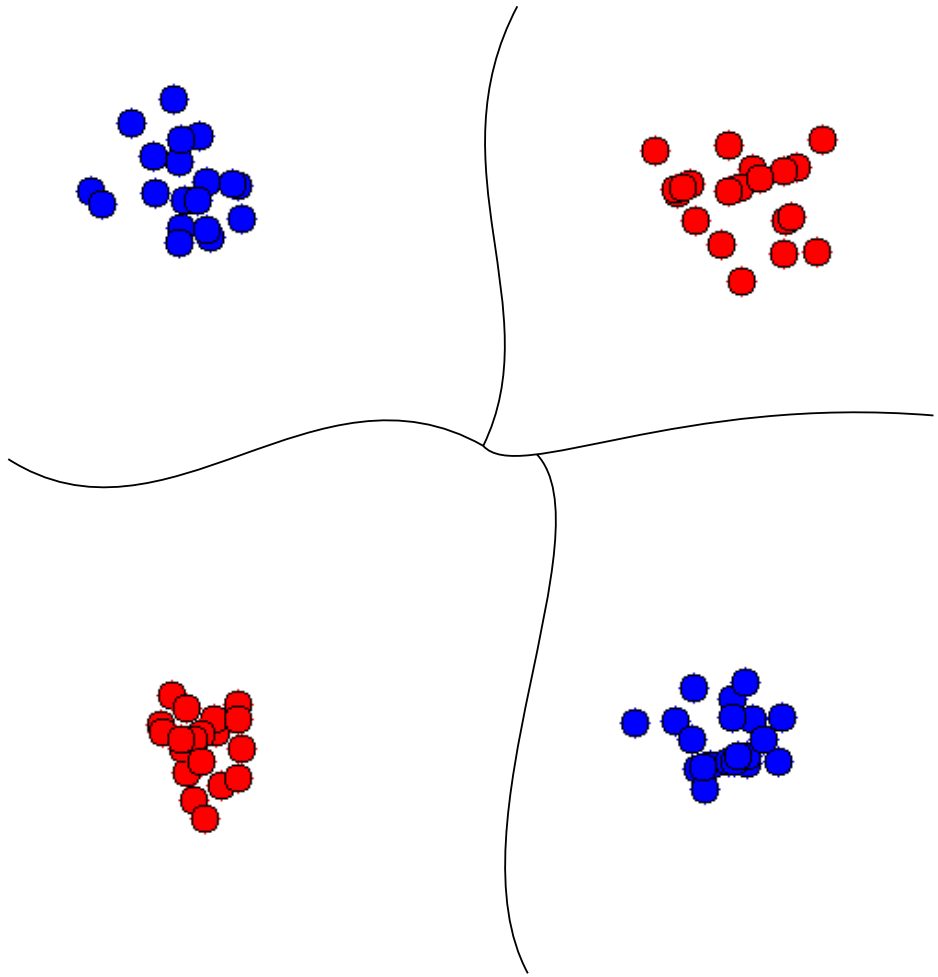
Linear Separability

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



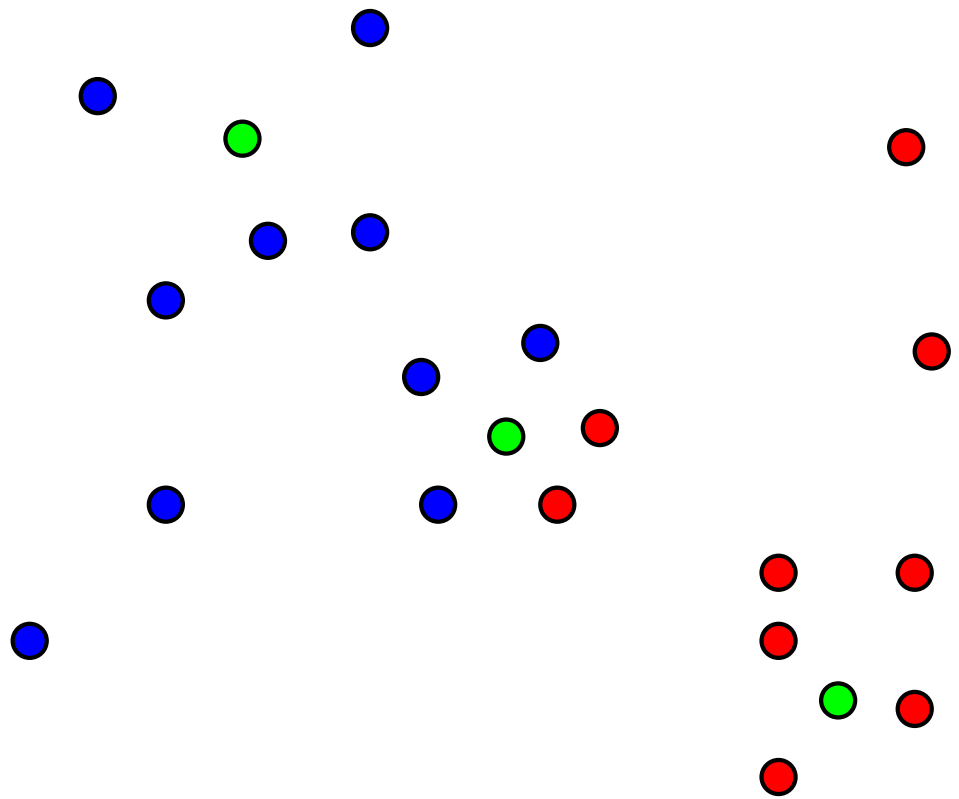
Non-linear Decision Boundary

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



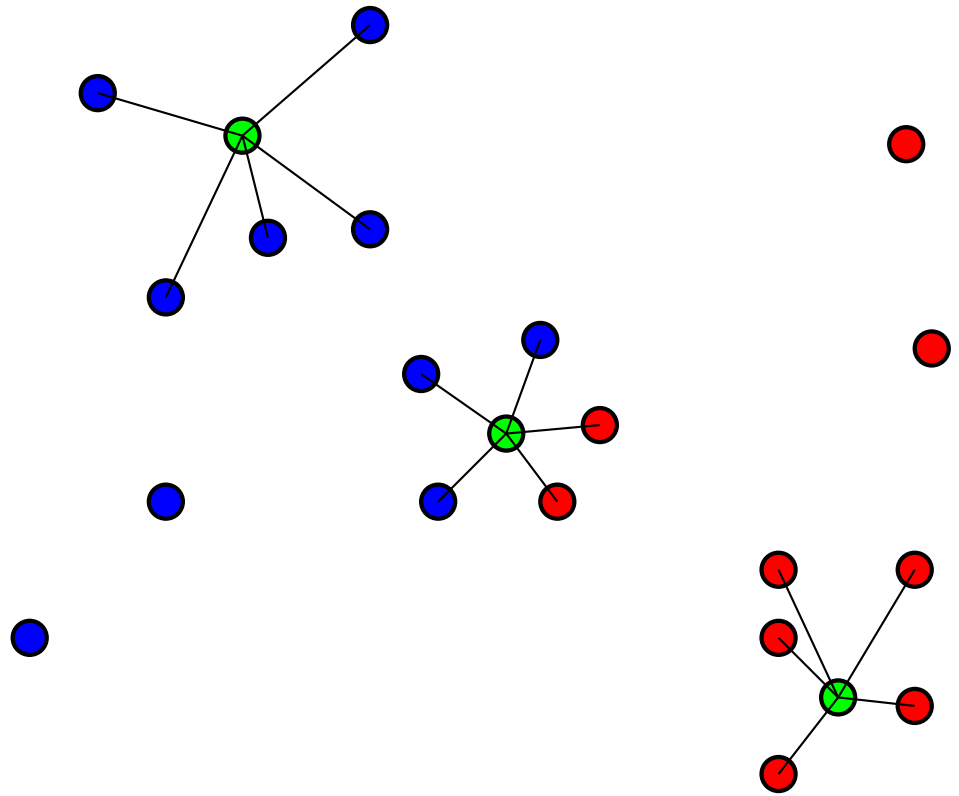
kNN

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



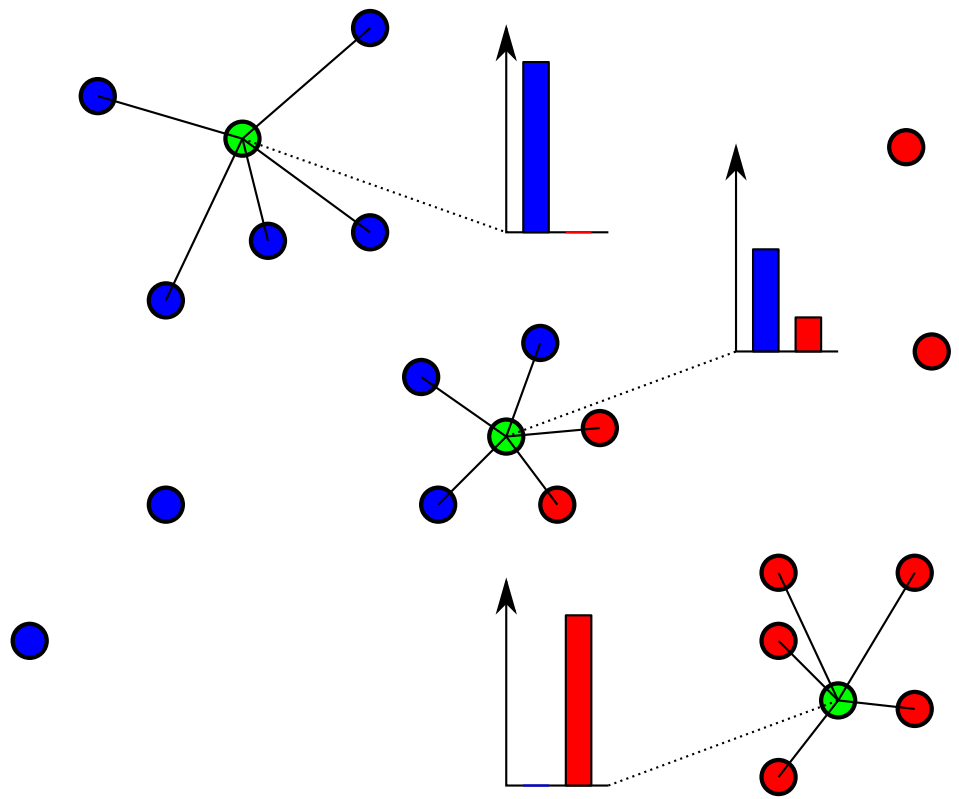
kNN

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



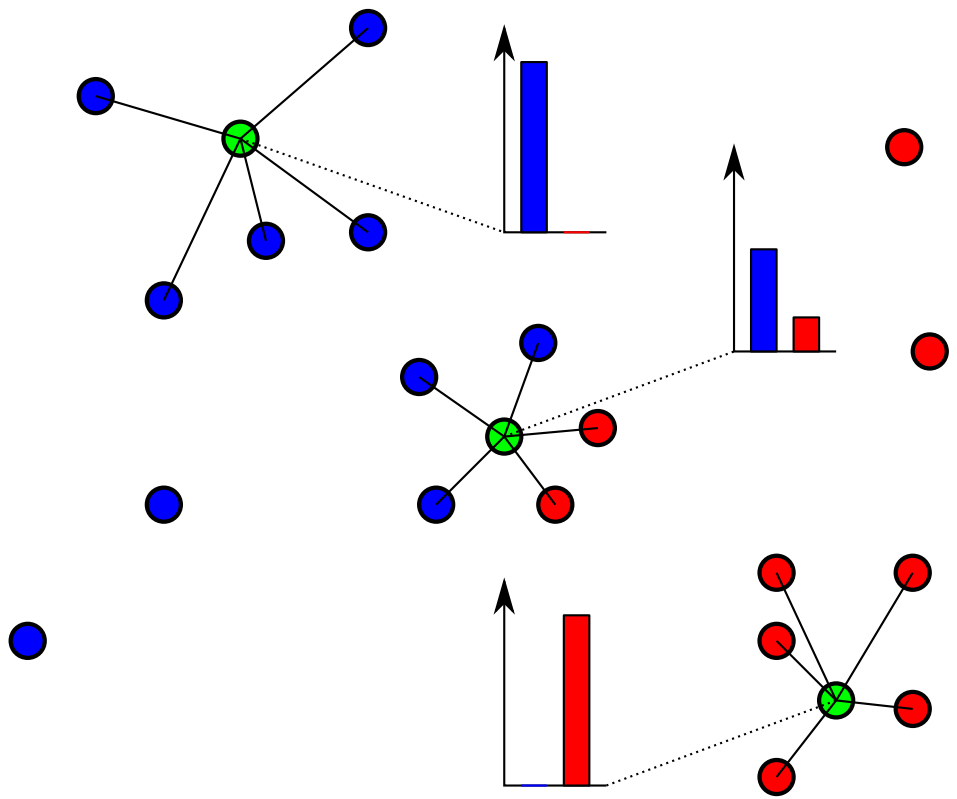
kNN

- 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

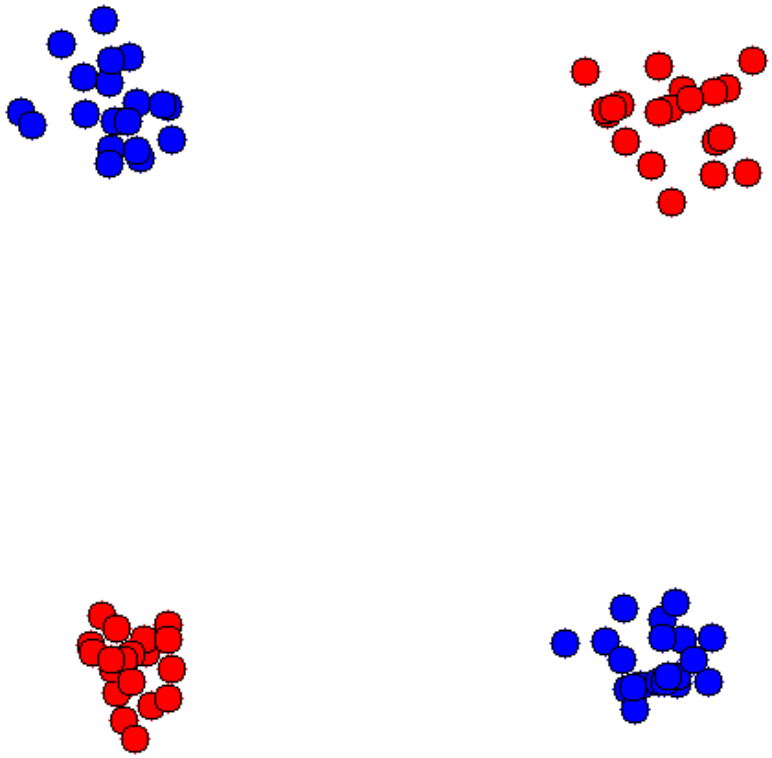


kNN

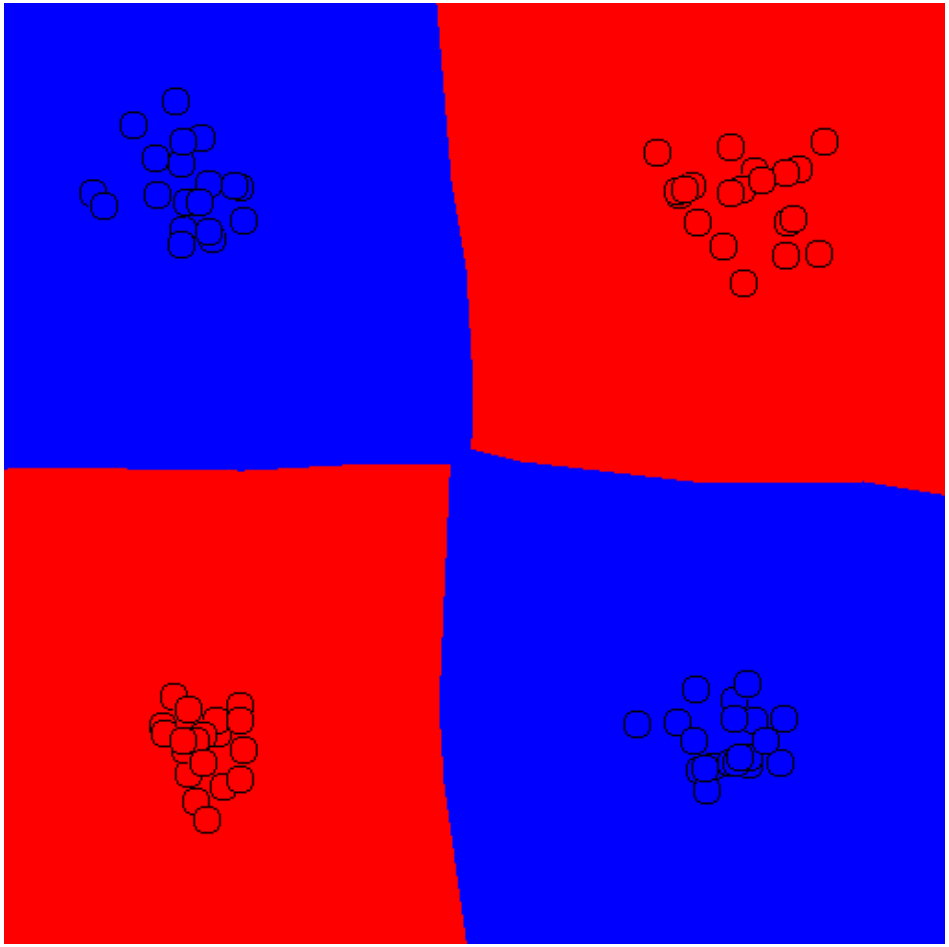
- Very simple idea: k-Nearest-Neighbors for classification
- For a sample find the k (e.g. 5) closest data points in the training dataset
- Look at the labels of those neighbors
- Fast lookup through trees/approximate methods
- Needs to keep all training data around



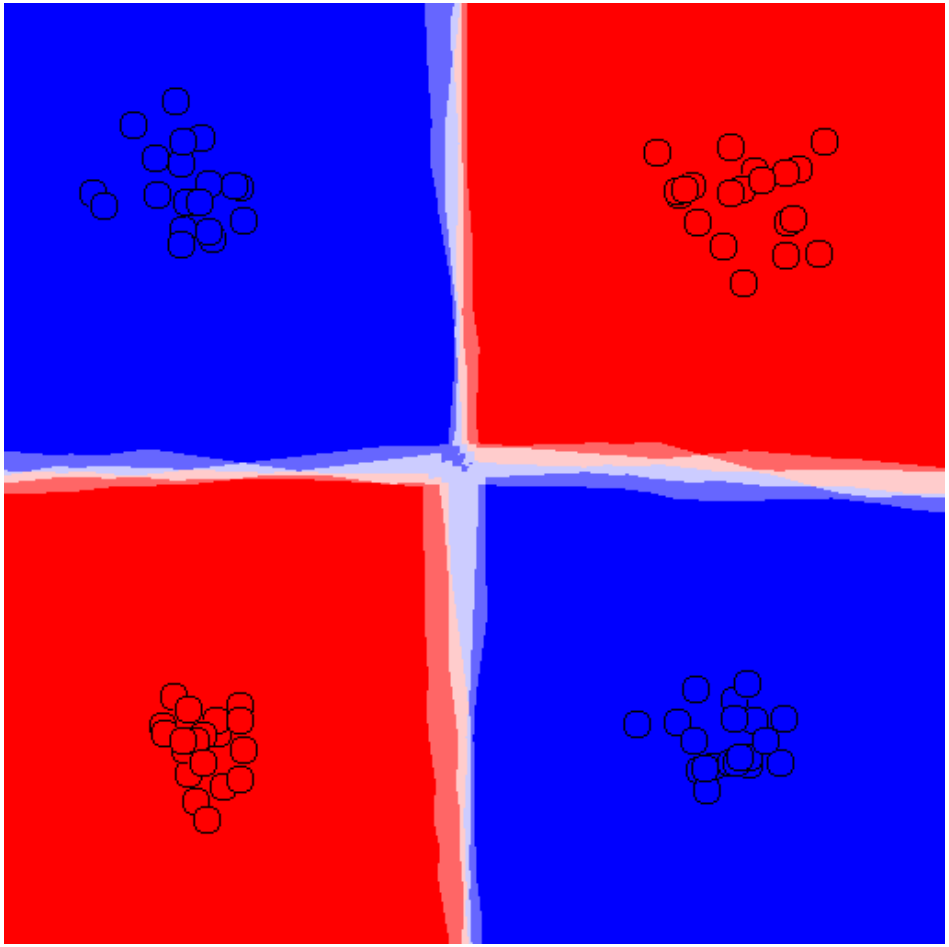
kNN Example - Simple



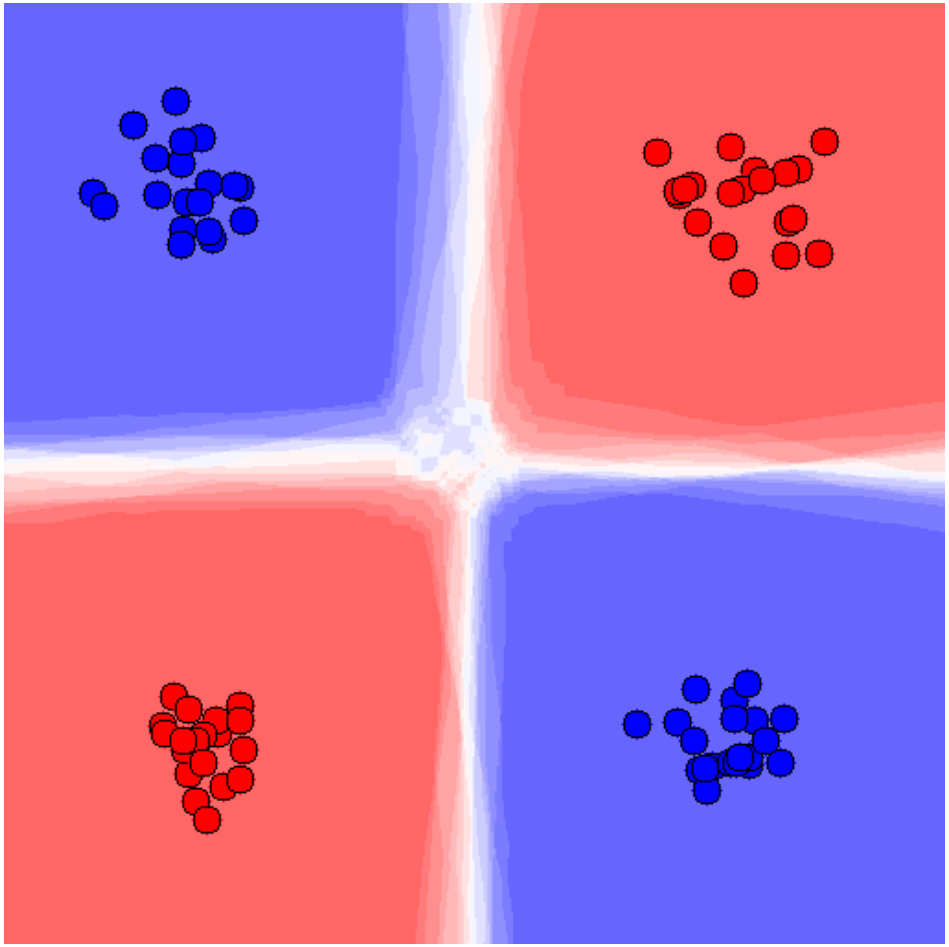
kNN Example - Simple - kNN K=1



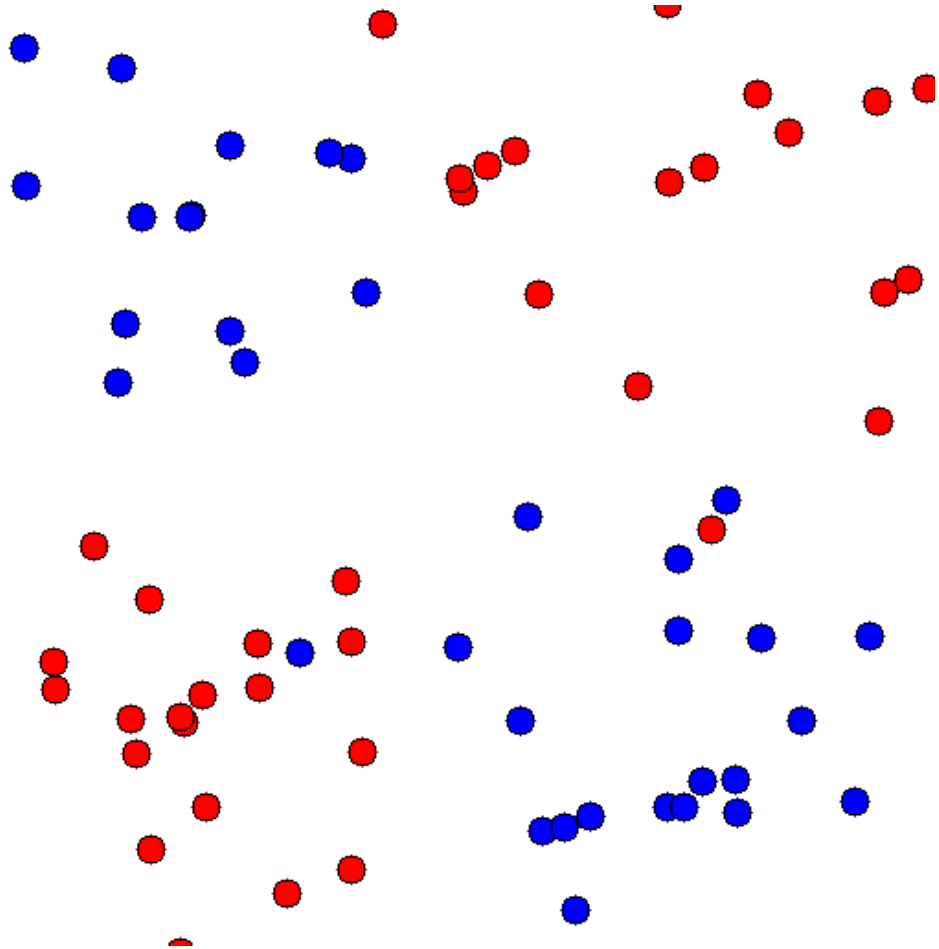
kNN Example - Simple - kNN K=5



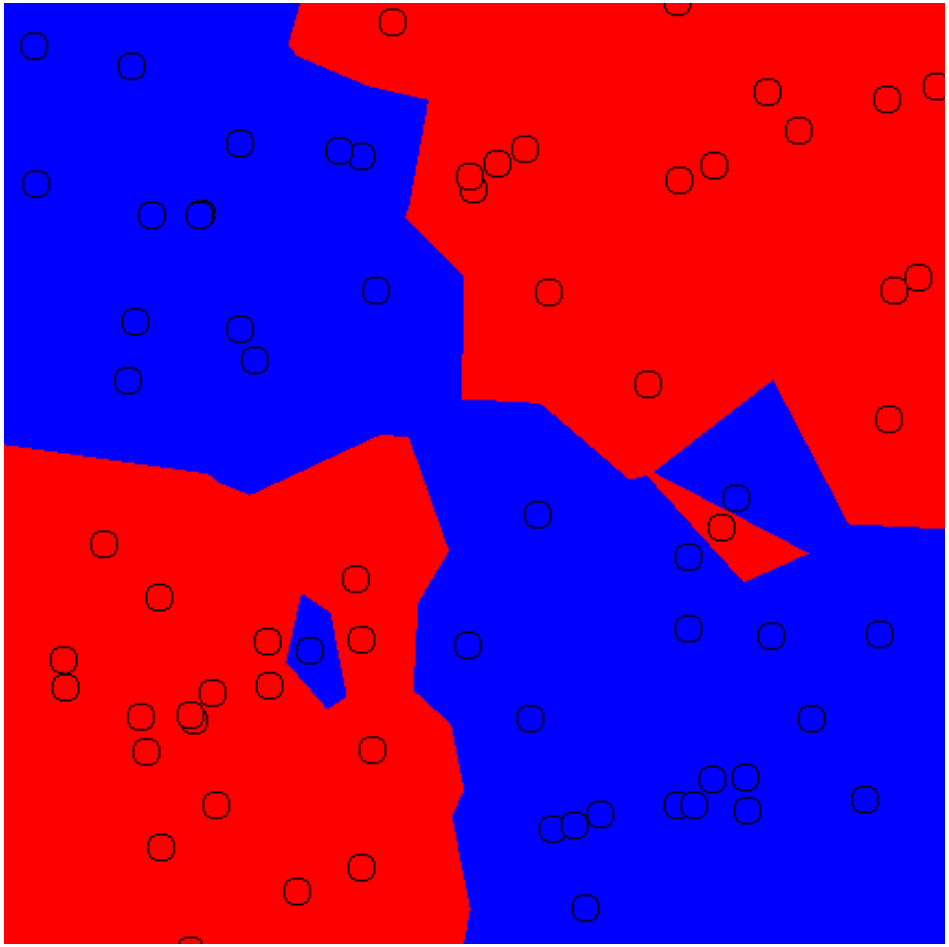
kNN Example - Simple - kNN K=25



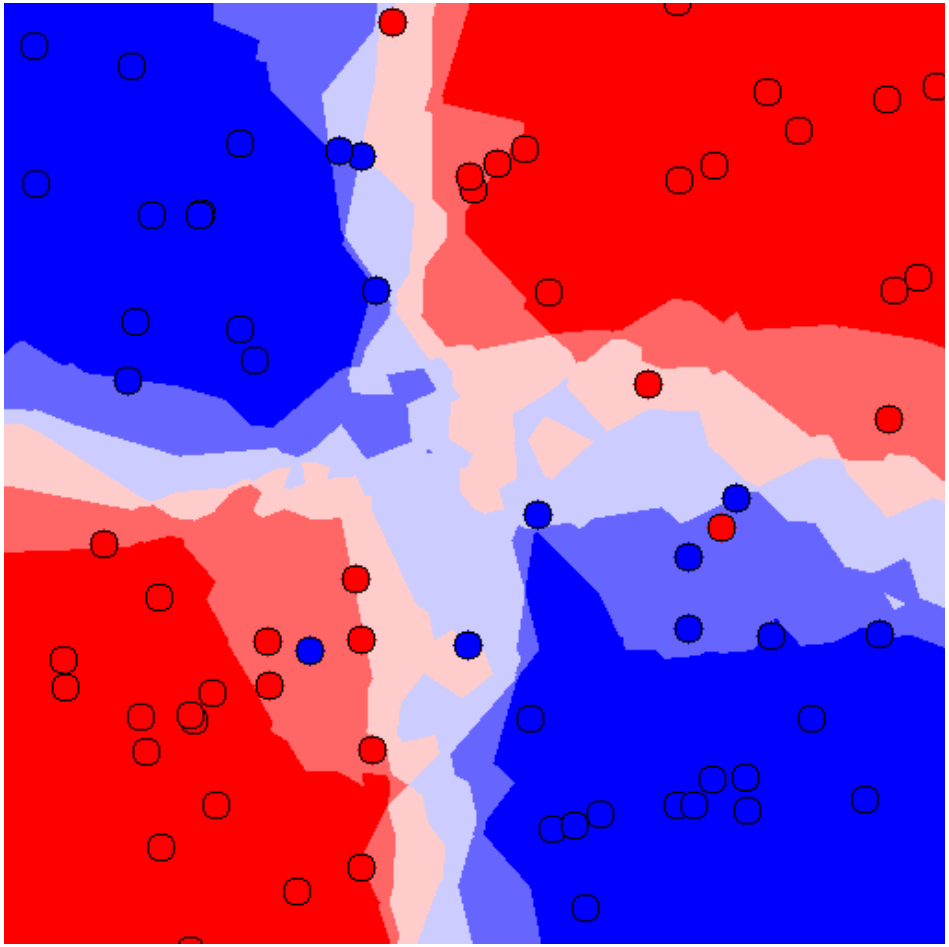
kNN Example - Hard



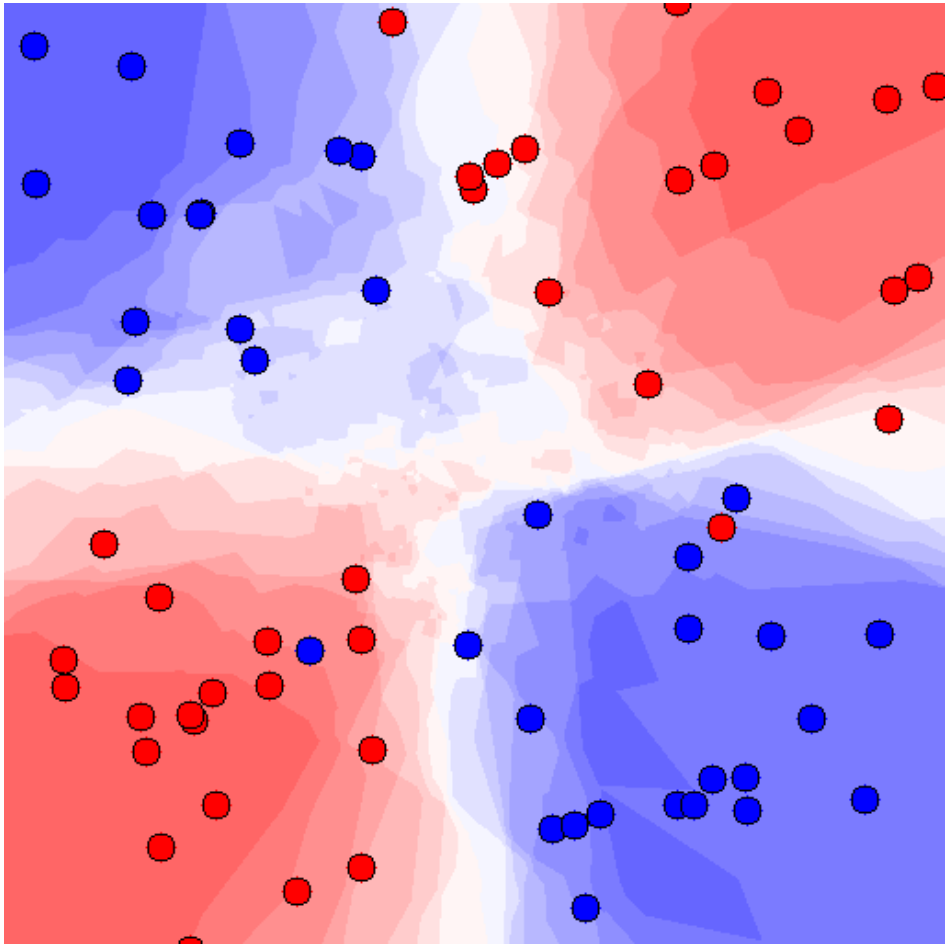
kNN Example - Hard - kNN K=1



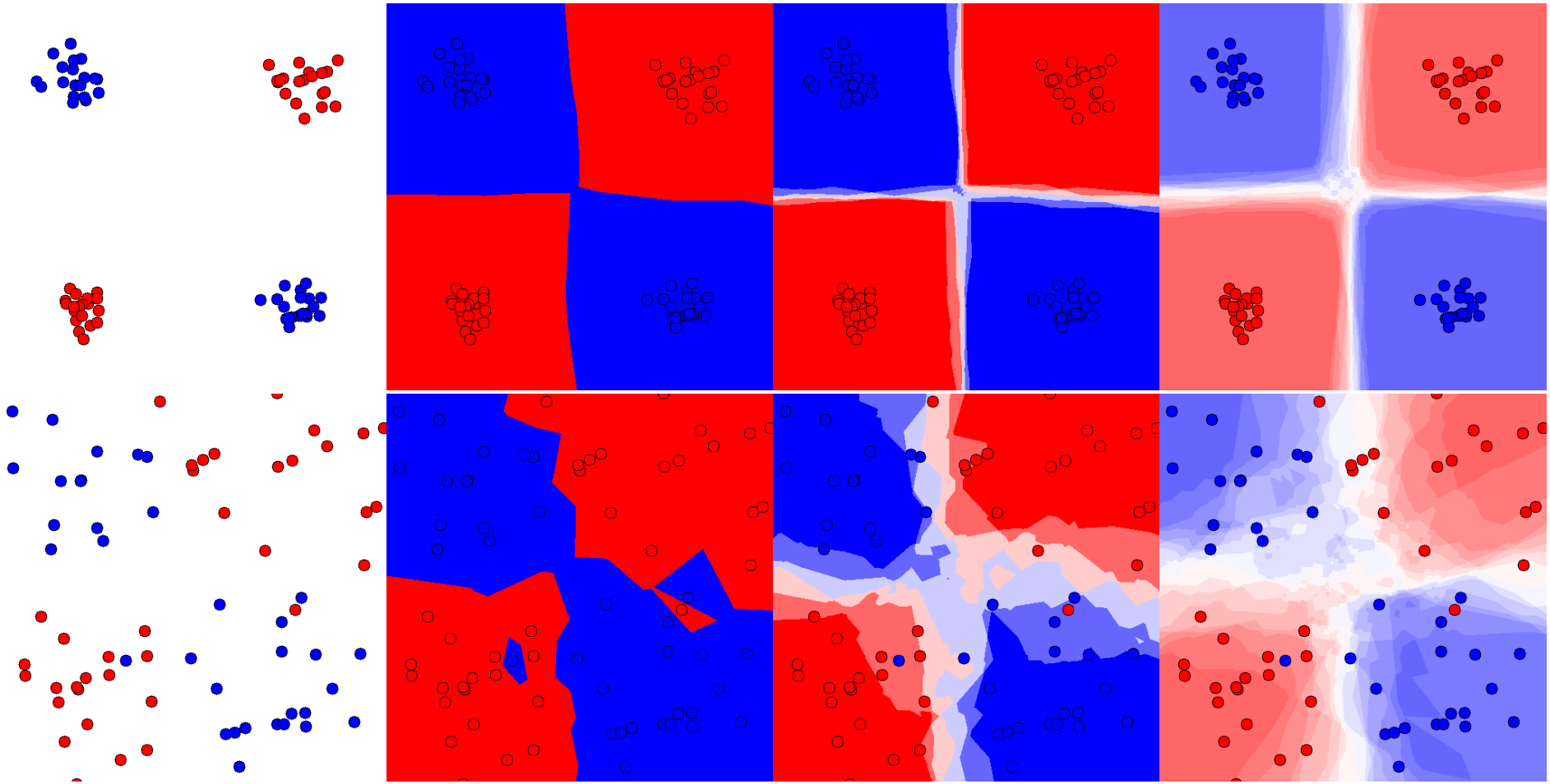
kNN Example - Hard - kNN K=5



kNN Example - Hard - kNN K=25

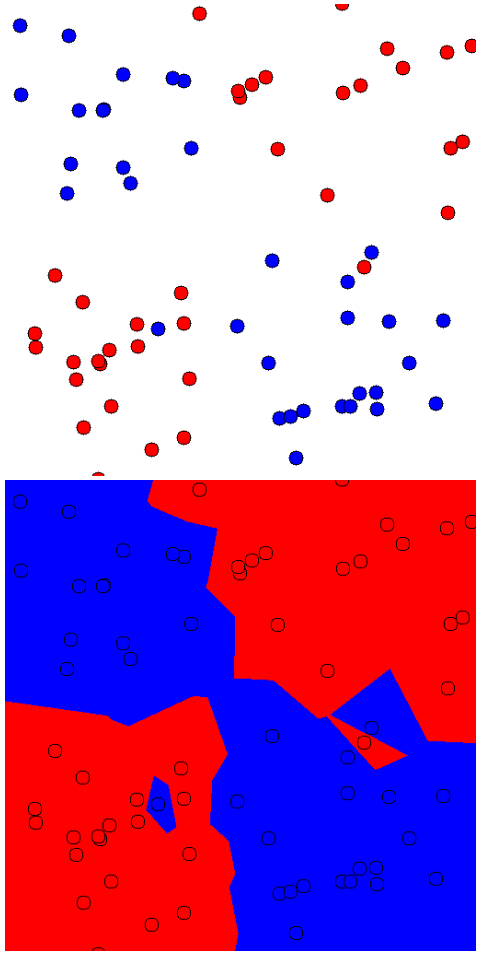


kNN Example



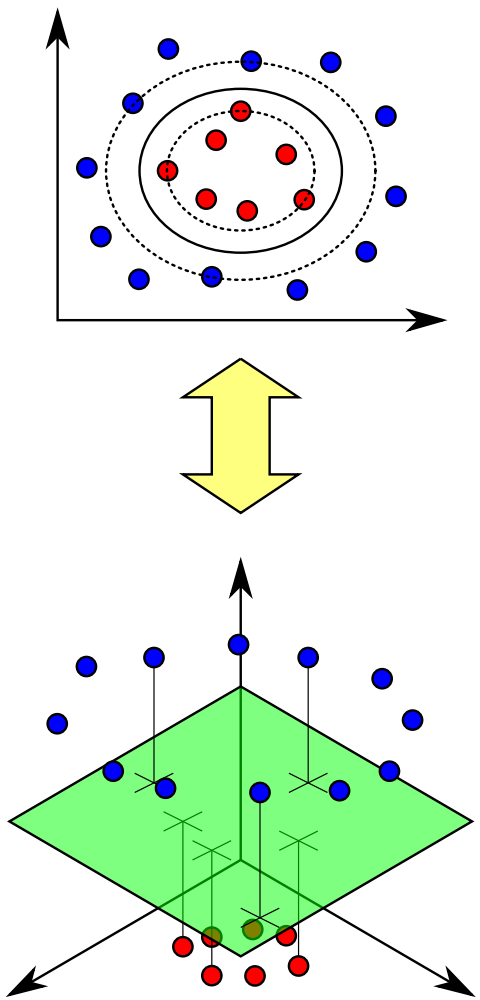
Model Complexity vs Overfitting

- 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



Models

1. Bayesian decision theory
2. Classification based on Features
   - Decision Boundary
   - Linear Decision Boundary
   - Non-linear Decision Boundary
3. Machine Learning Methods
   - Support Vector Machine (SVM)
   - Multi-Layer Perceptron (MLP)
   - Random Forest (RF)
   - Fusion
   - Node Tests
   - Interpretation
   - Application Tips
4. Feature extraction

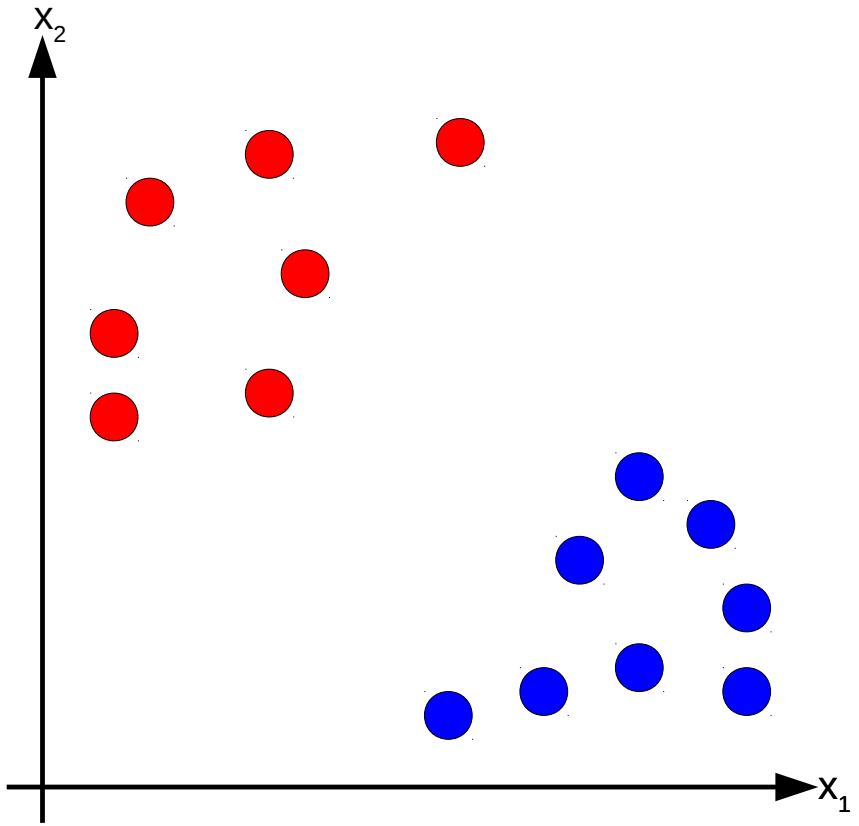


SVM

Reconsider the perceptron:

Perceptron

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

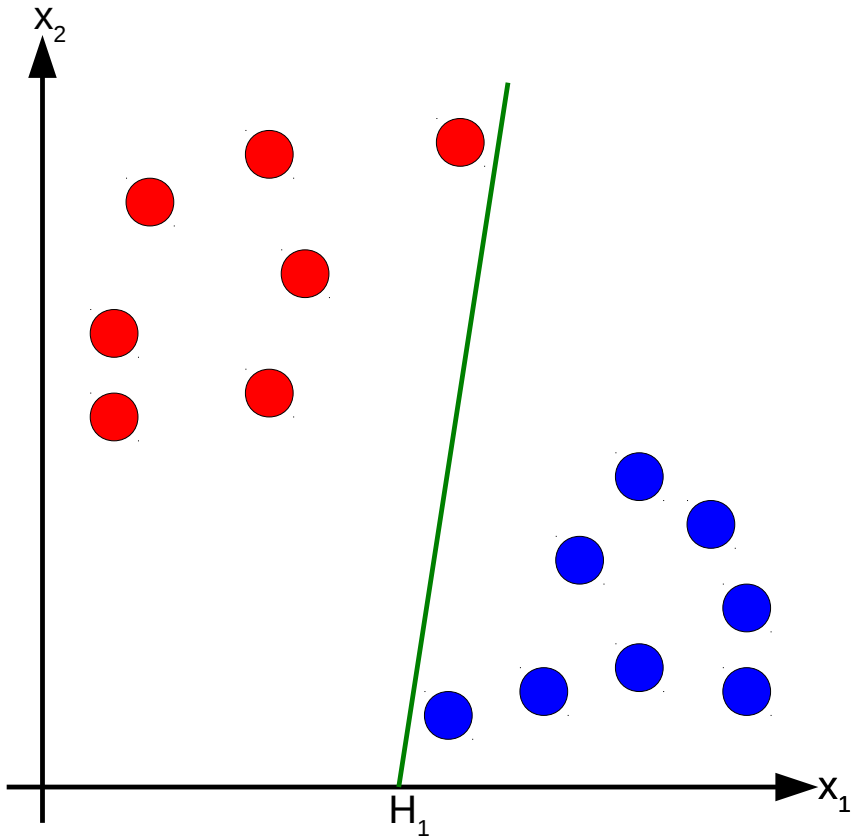


SVM

Reconsider the perceptron:

Perceptron

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

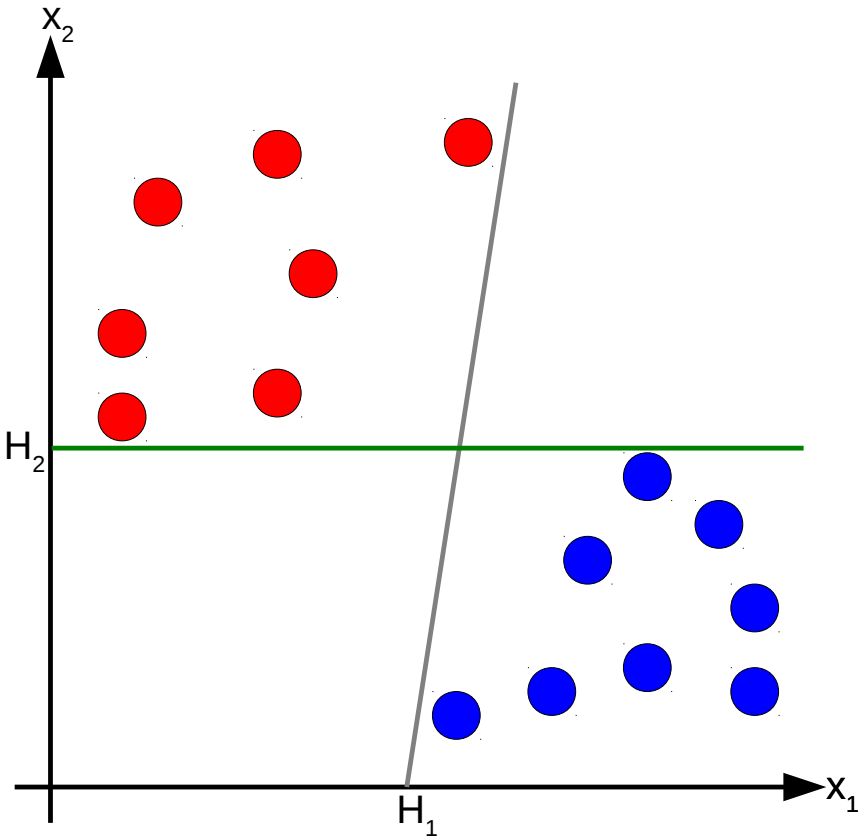


SVM

Reconsider the perceptron:

Perceptron

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

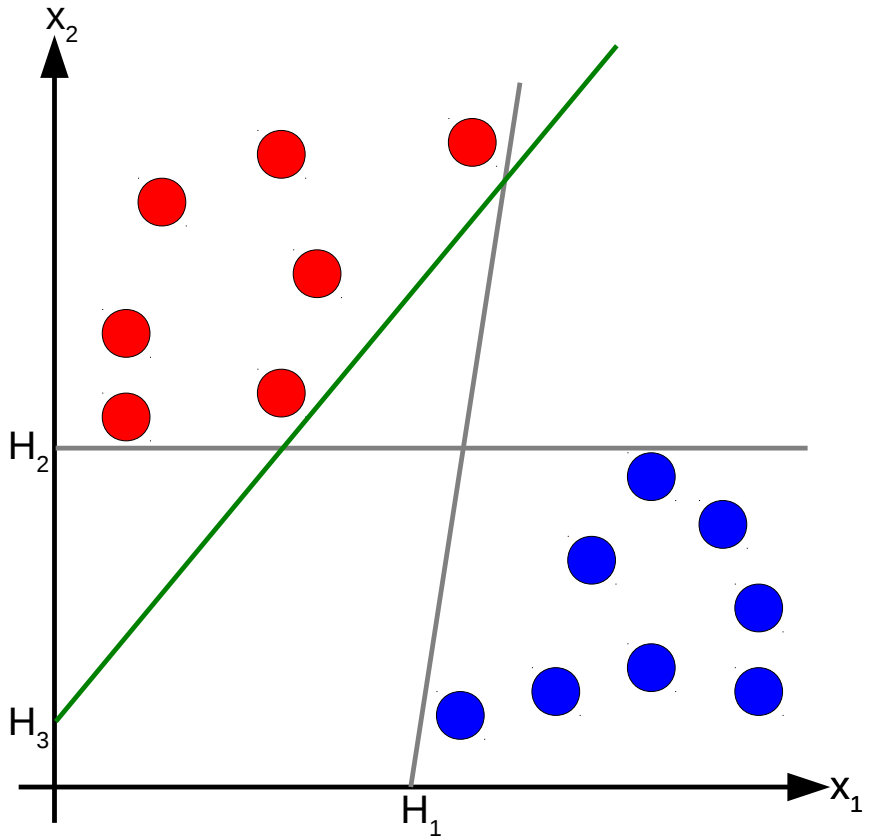


SVM

Reconsider the perceptron:

Perceptron

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

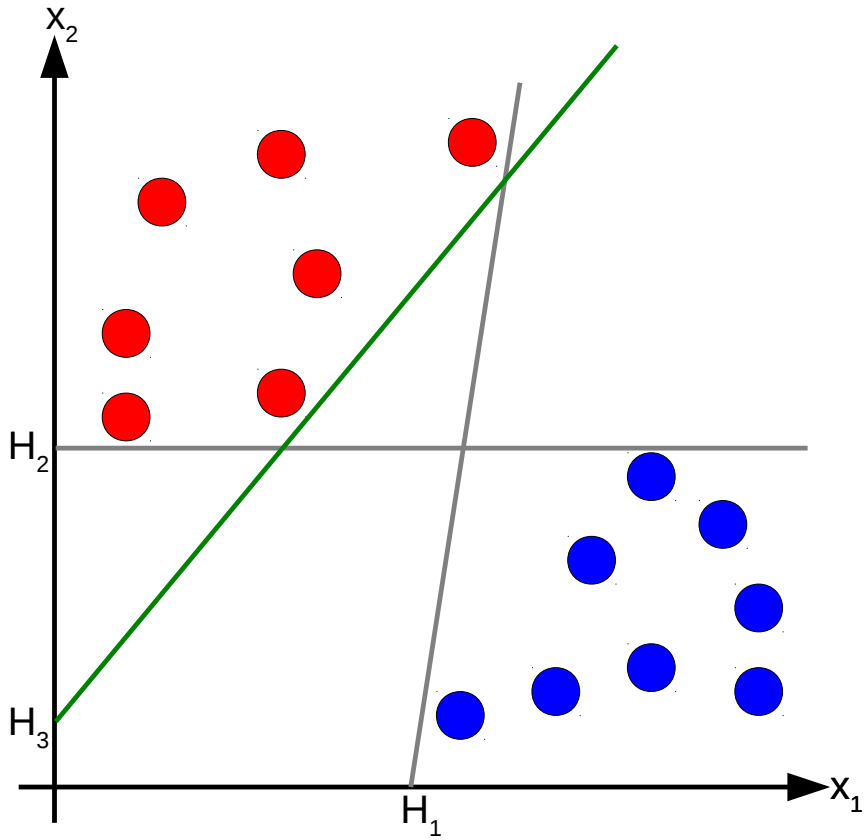


SVM

Reconsider the perceptron:

Perceptron

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

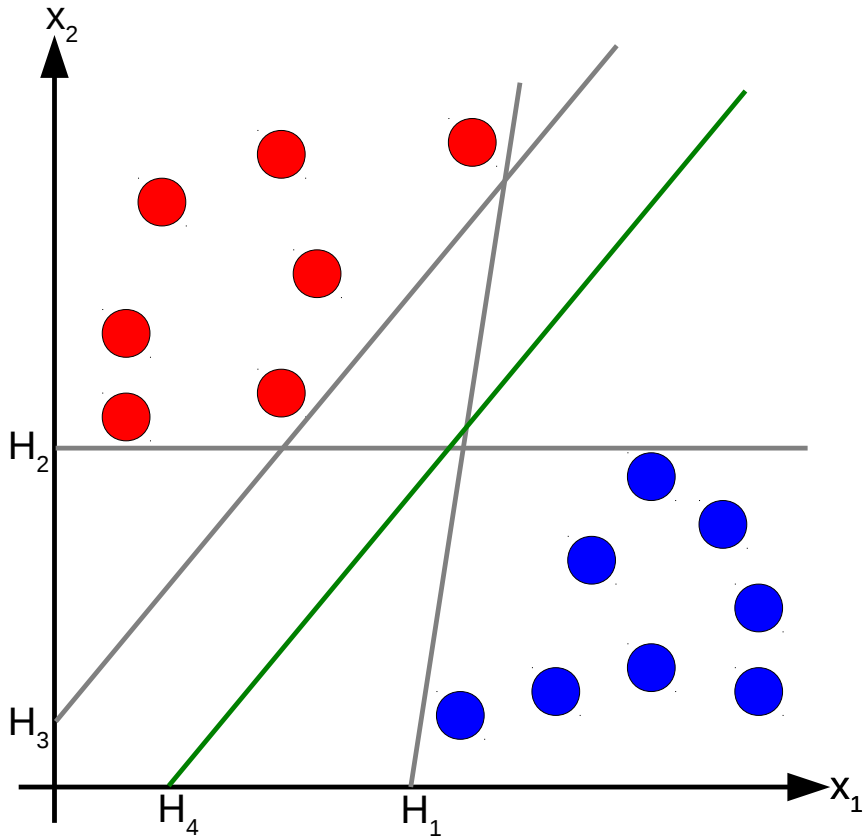


SVM

Reconsider the perceptron:

Perceptron

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



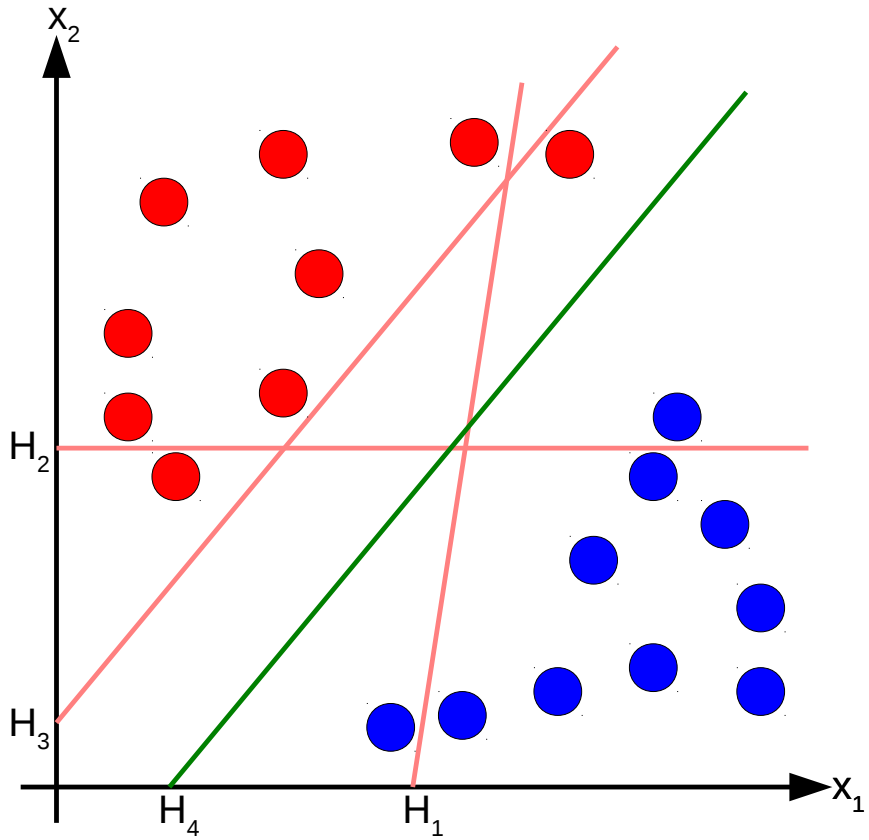
SVM

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



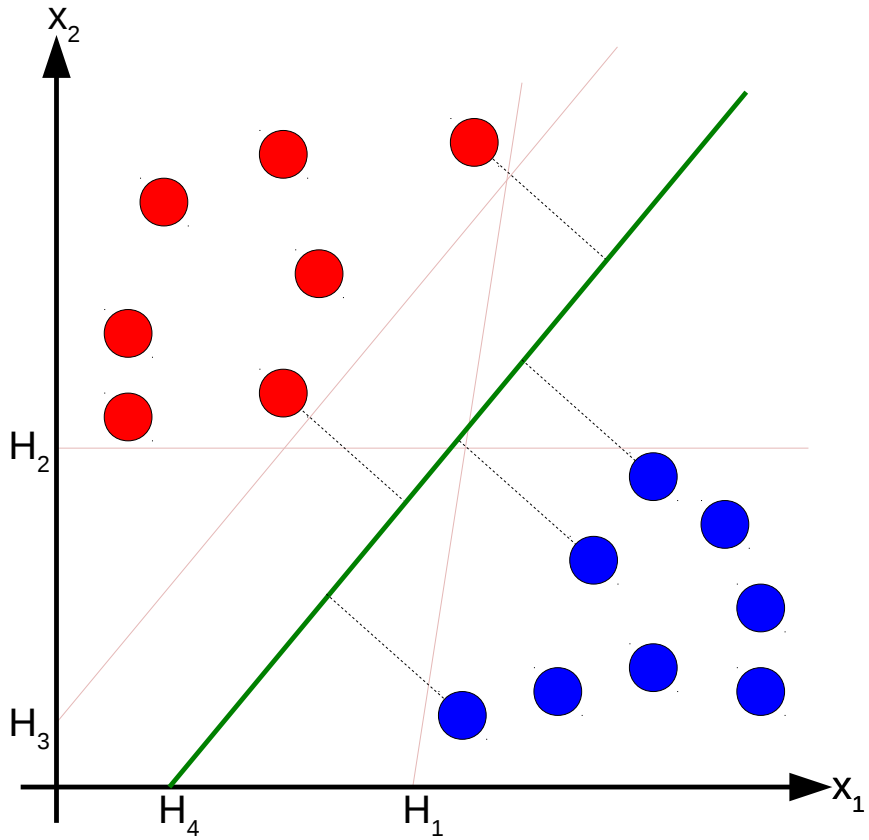
SVM

Reconsider the perceptron:

Perceptron

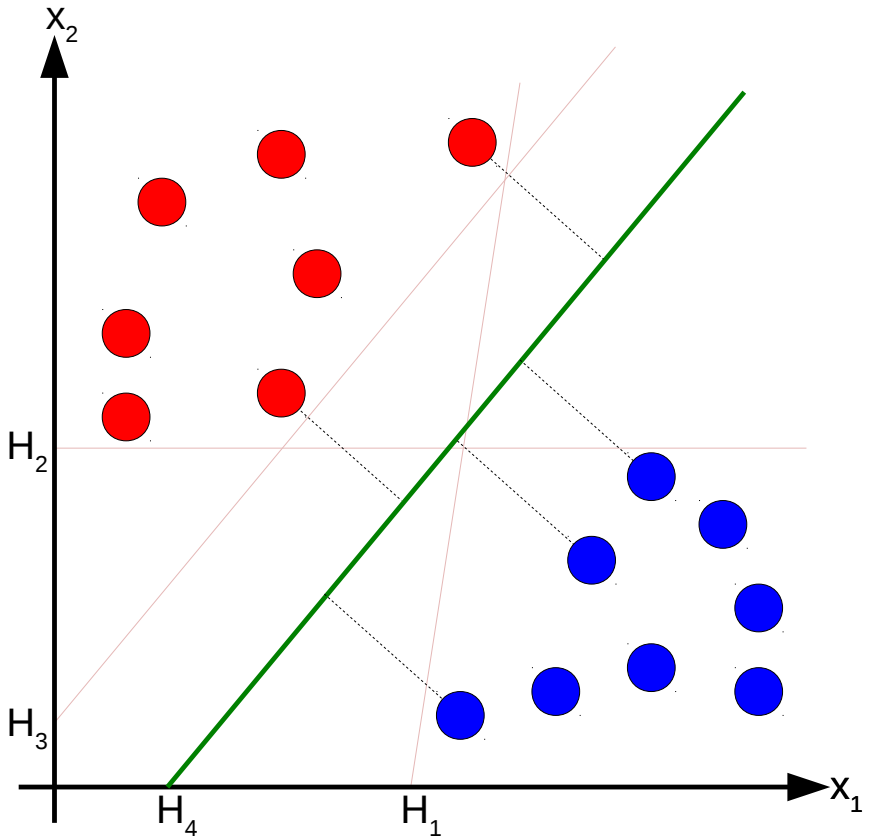
$$y = \text{sign}(\mathbf{w}^T\mathbf{x} + b) \qquad (2)$$

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



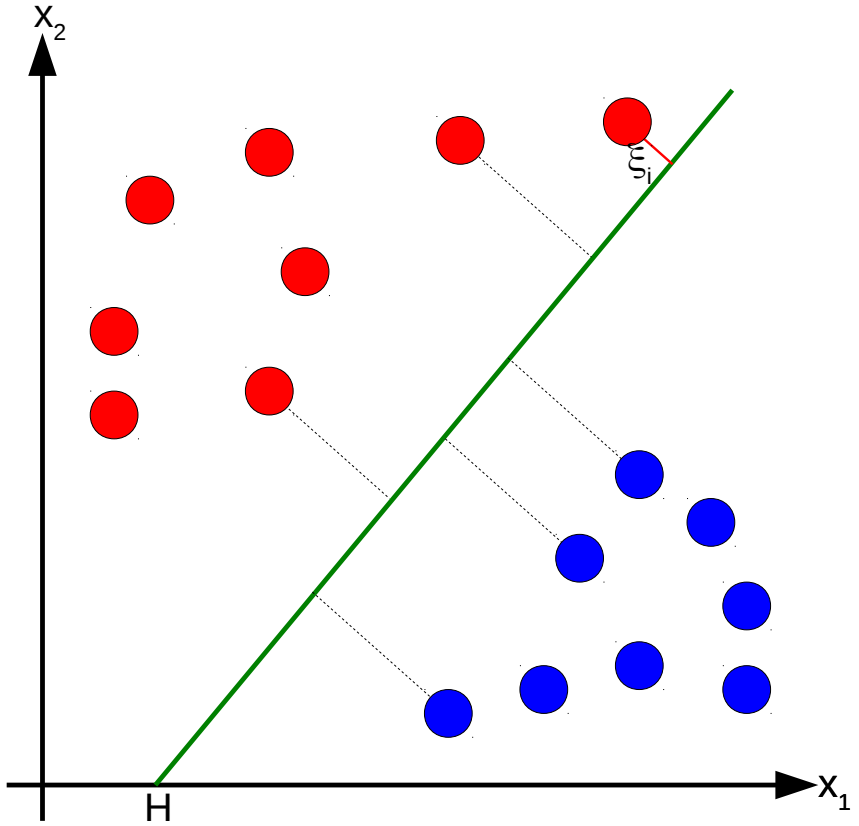
SVM

- 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 $\hat{y}_i(w^T\mathbf{x}_i - b) \geq 1 - \xi_i$
  $(\xi_i \geq 0)$

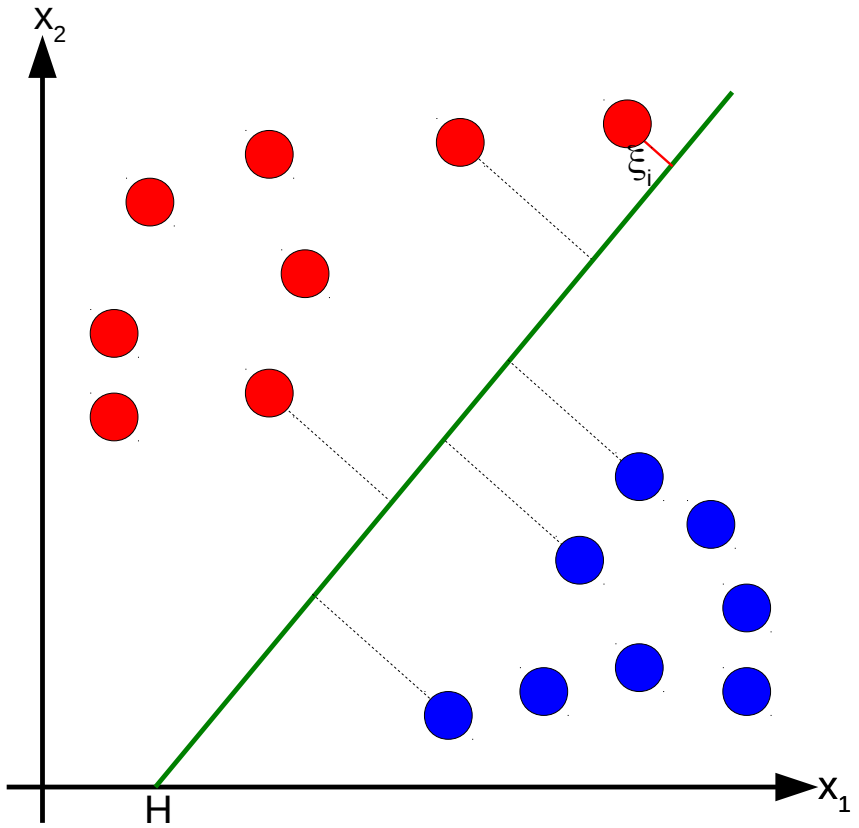


SVM

- 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 $\mathbf{w} = \sum_{i=1}^{n} \alpha_i \hat{y}_i \mathbf{x}_i$

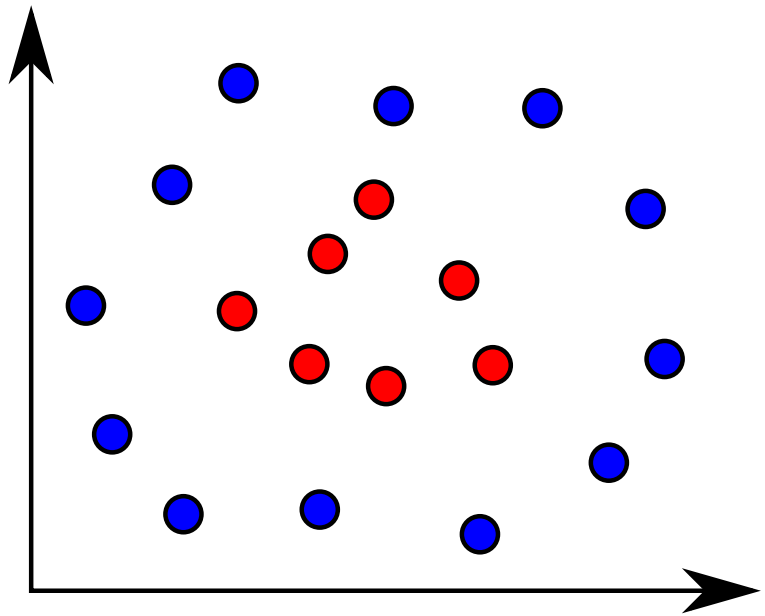


SVM

- 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?

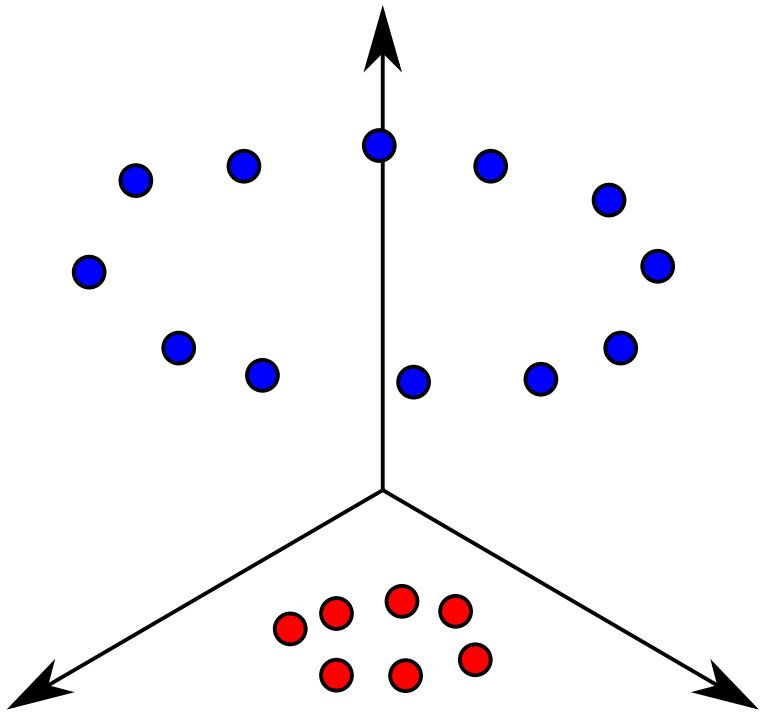


SVM

- 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 $\mathbf{x} \mapsto \phi(\mathbf{x})$

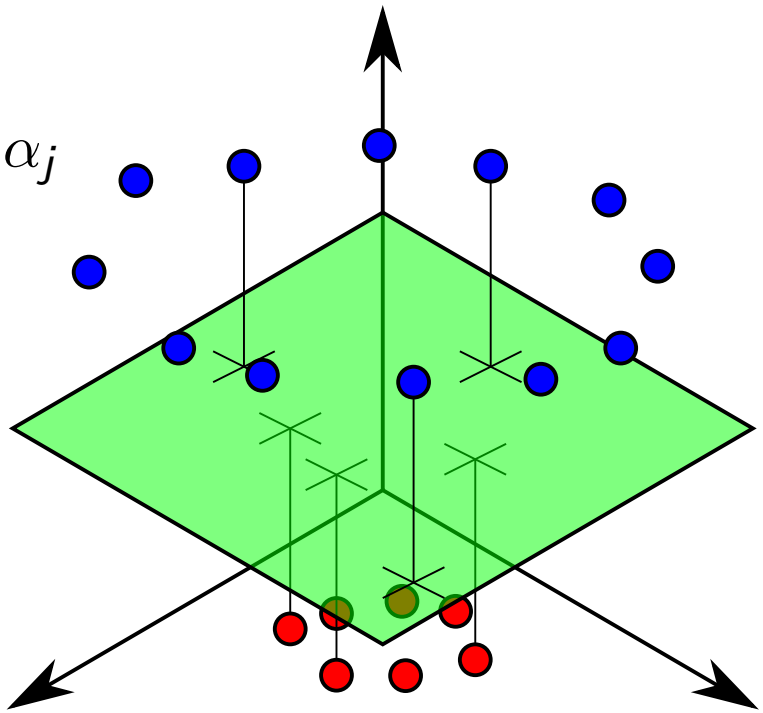


SVM

- 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 $\mathbf{x} \mapsto \phi(\mathbf{x})$

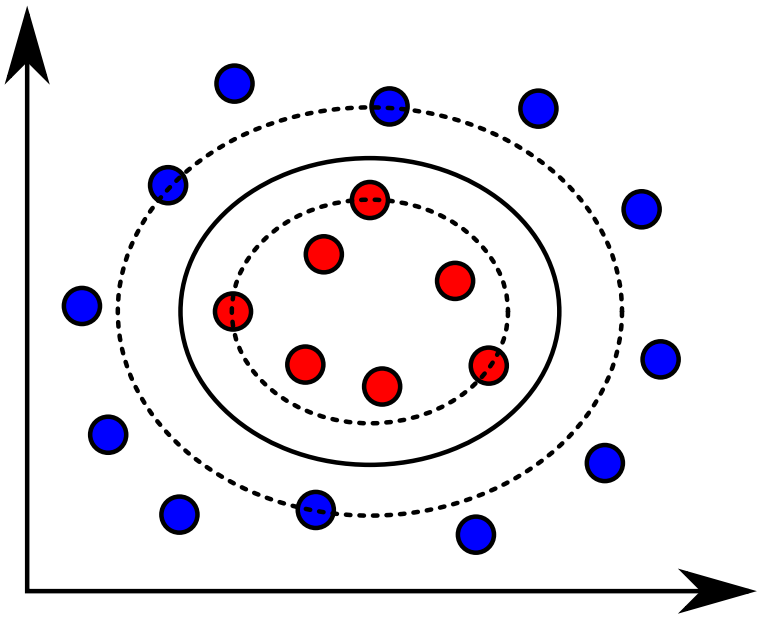


SVM

- 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

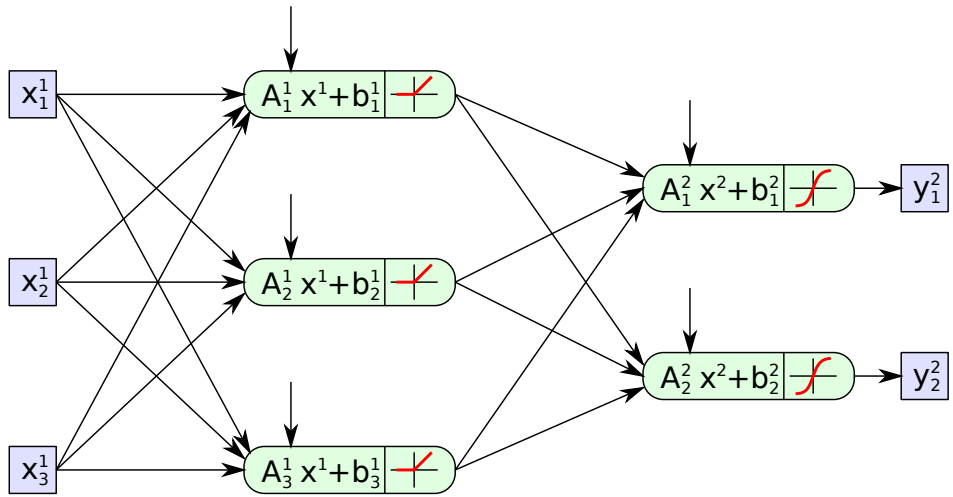
- Multiple kernels exist
  - Linear $k(\mathbf{x}_i, \mathbf{x}_j) = \mathbf{x}_i \cdot \mathbf{x}_j$
  - Polynomial $k(\mathbf{x}_i, \mathbf{x}_j) = (\mathbf{x}_i \cdot \mathbf{x}_j)^d$
  - RBF $k(\mathbf{x}_i, \mathbf{x}_j) = \exp(-\gamma||\mathbf{x}_i - \mathbf{x}_j)||^2$
  - Hyperbolic tangent $k(\mathbf{x}_i, \mathbf{x}_j) = \tanh(\kappa \cdot \mathbf{x}_i \cdot \mathbf{x}_j + c)$
- 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
- 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

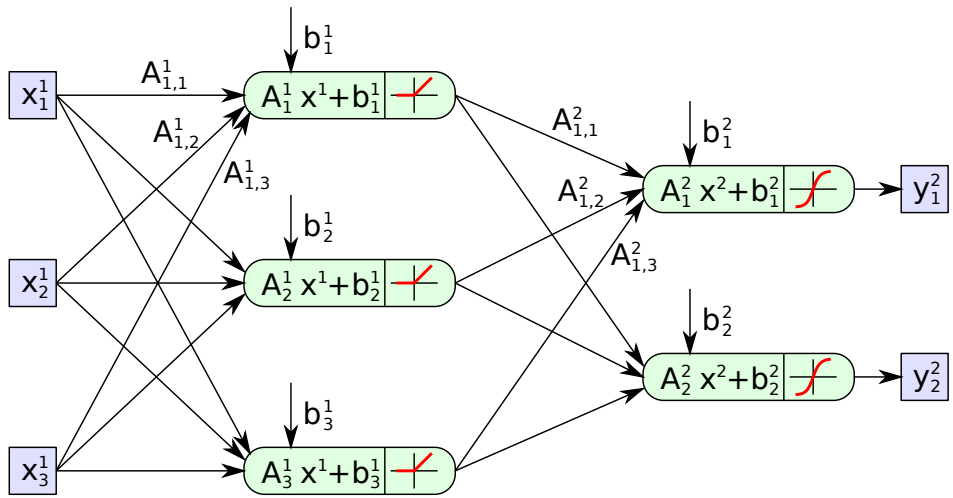
Models

1. Bayesian decision theory
2. Classification based on Features
   - Decision Boundary
   - Linear Decision Boundary
   - Non-linear Decision Boundary
3. Machine Learning Methods
   - Support Vector Machine (SVM)
   - Multi-Layer Perceptron (MLP)
   - Random Forest (RF)
   - Fusion
   - Node Tests
   - Interpretation
   - Application Tips
4. Feature extraction



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



Multi-Layer Perceptron

- You will learn more tomorrow ☺

Increasing Depth

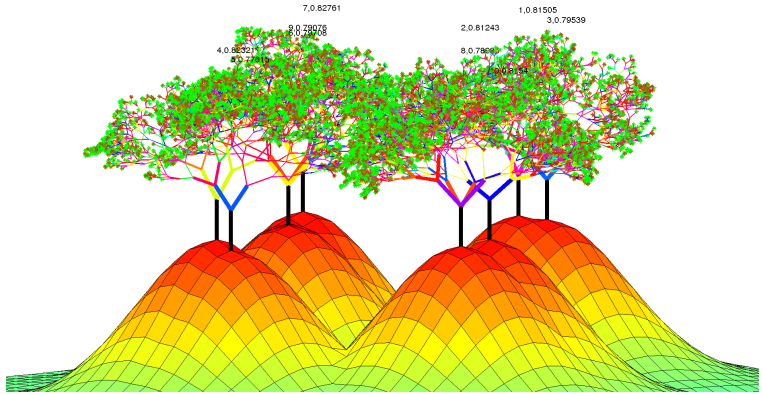
- Recent trend goes towards deeper networks
- Networks more powerful, but ...
- ... more difficult to train
  - Gradients collapse/explode/diffuse through the layers
- This is the book to read: Goodfellow et al. "Deep Learning," 2016

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

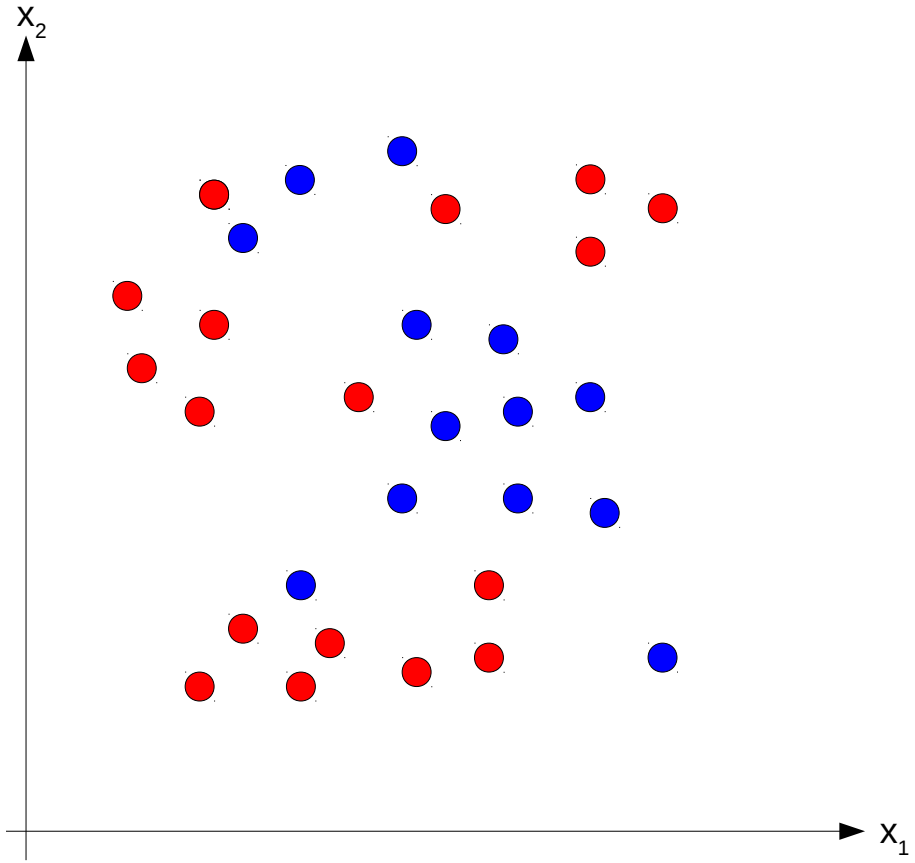
Models

1. Bayesian decision theory
2. Classification based on Features
   - Decision Boundary
   - Linear Decision Boundary
   - Non-linear Decision Boundary
3. Machine Learning Methods
   - Support Vector Machine (SVM)
   - Multi-Layer Perceptron (MLP)
   - Random Forest (RF)
   - Fusion
   - Node Tests
   - Interpretation
   - Application Tips
4. Feature extraction



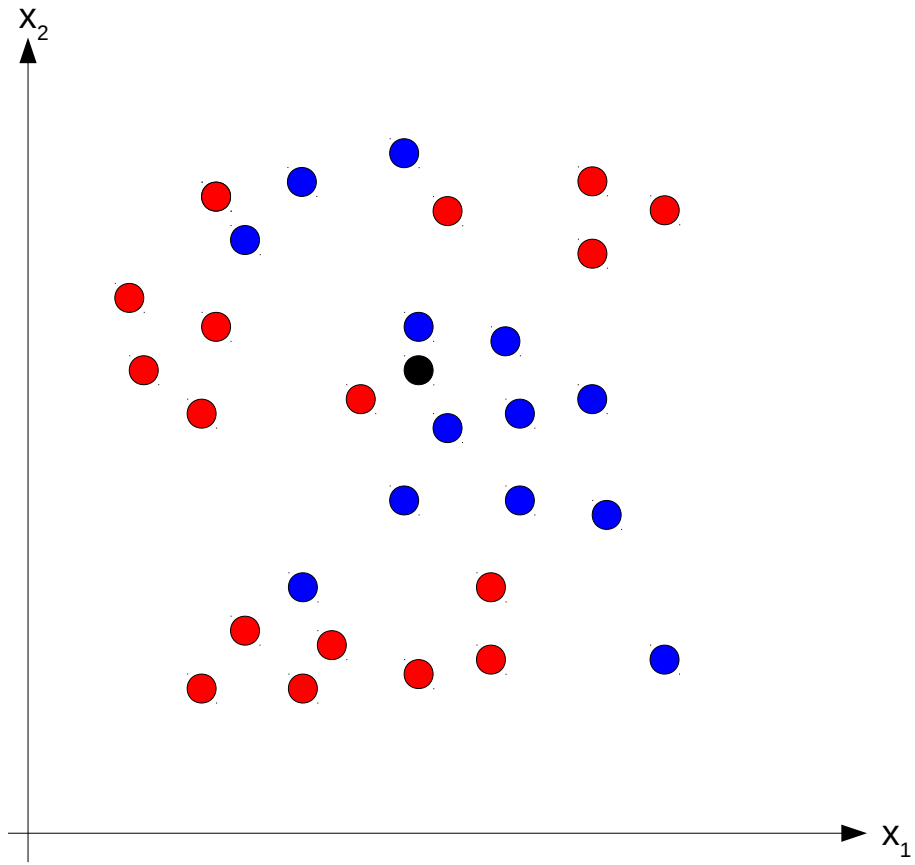
From kNN to Search Trees

- 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



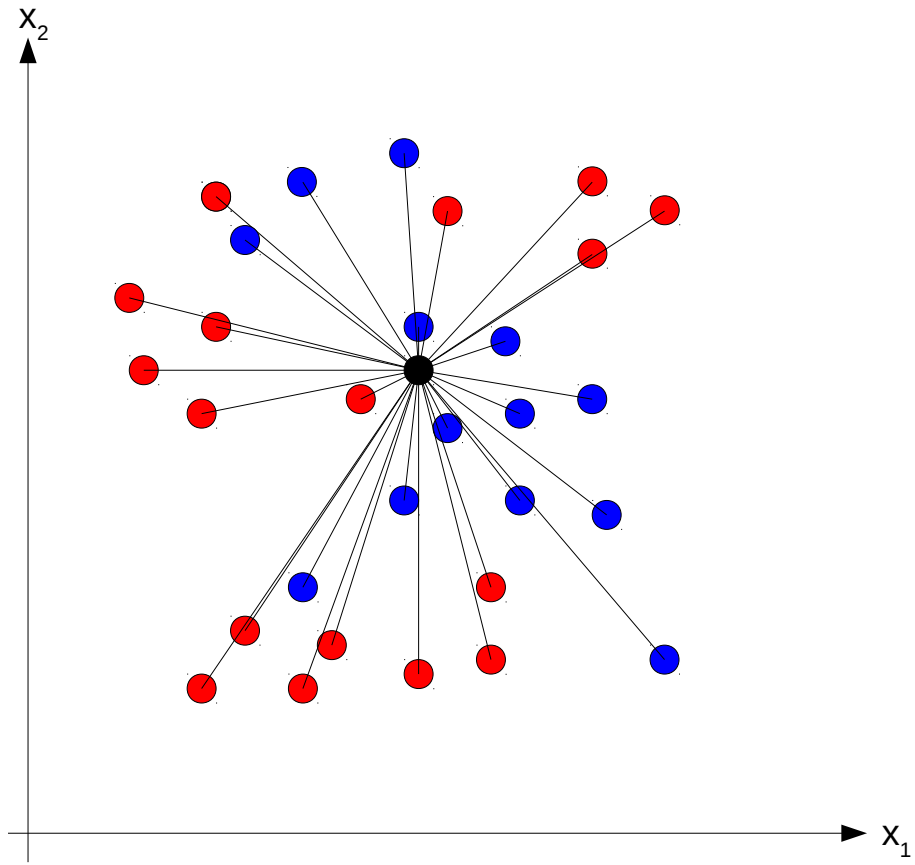
From kNN to Search Trees

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



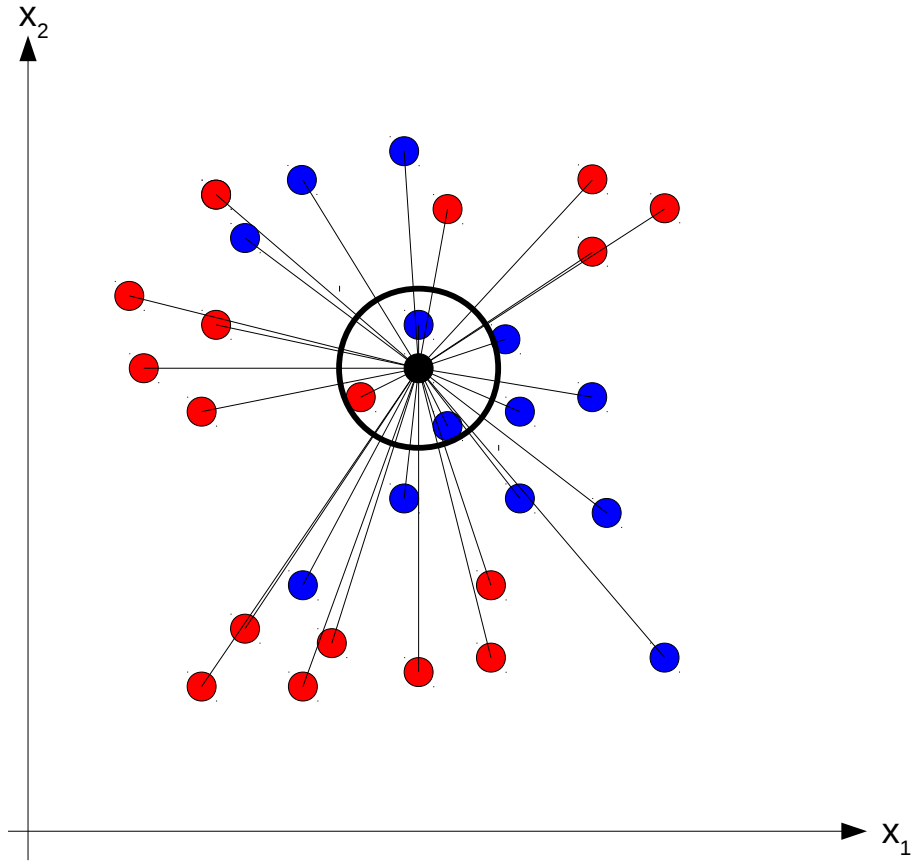
From kNN to Search Trees

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



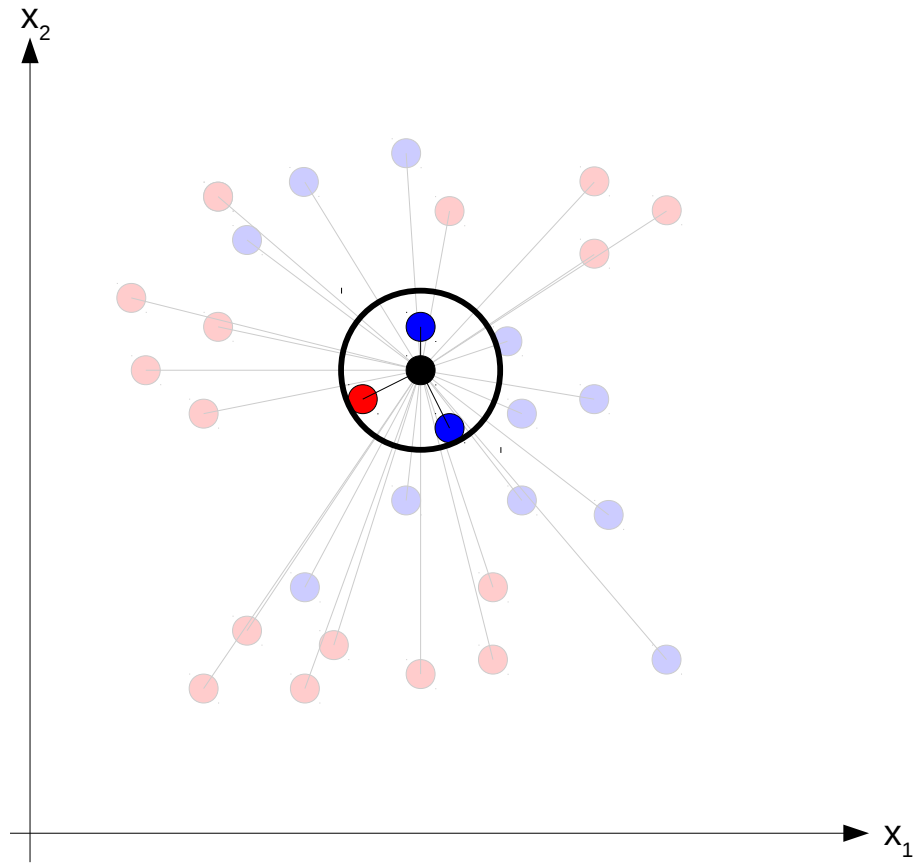
From kNN to Search Trees

- 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)



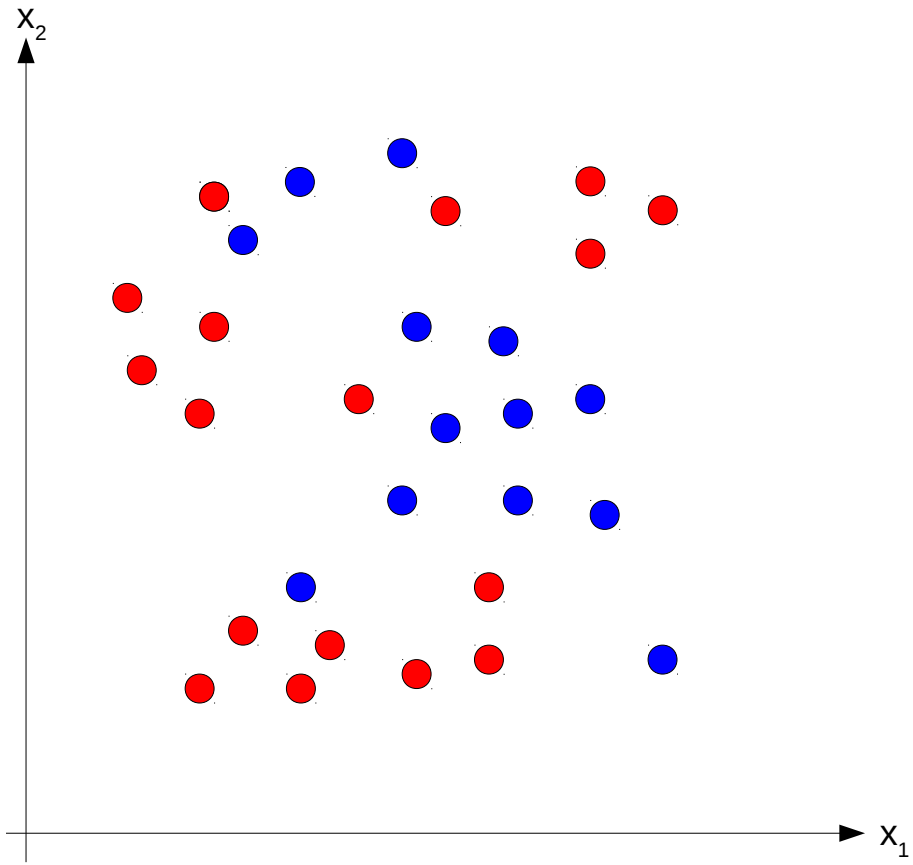
From kNN to Search Trees

- 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)



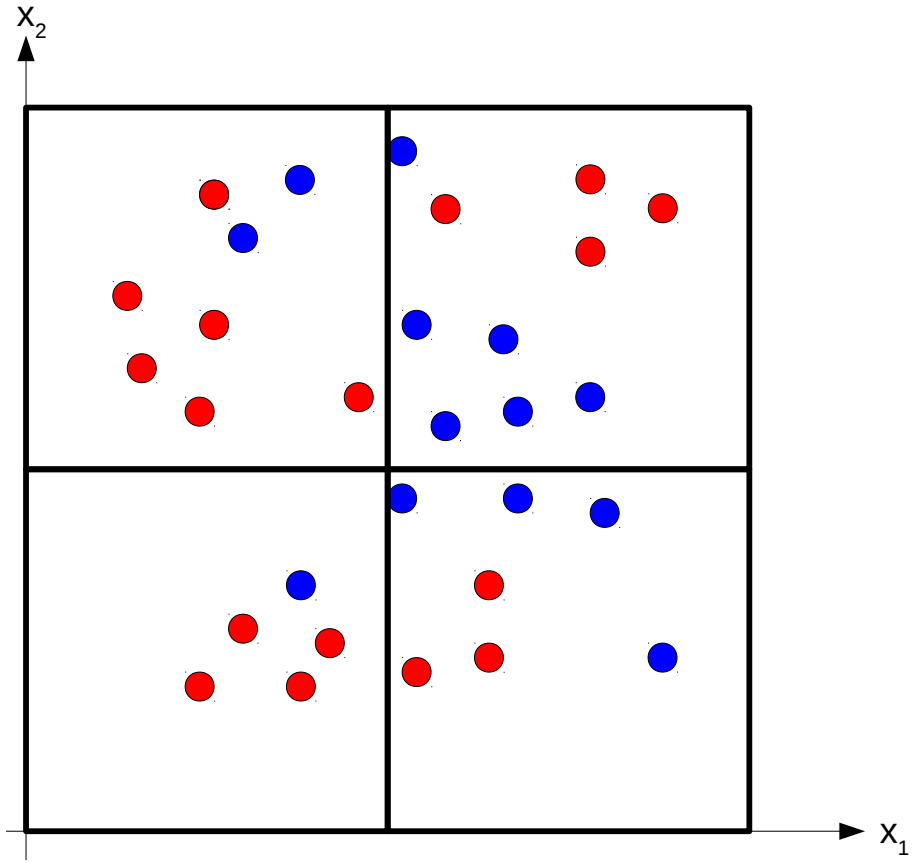
From kNN to Search Trees

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



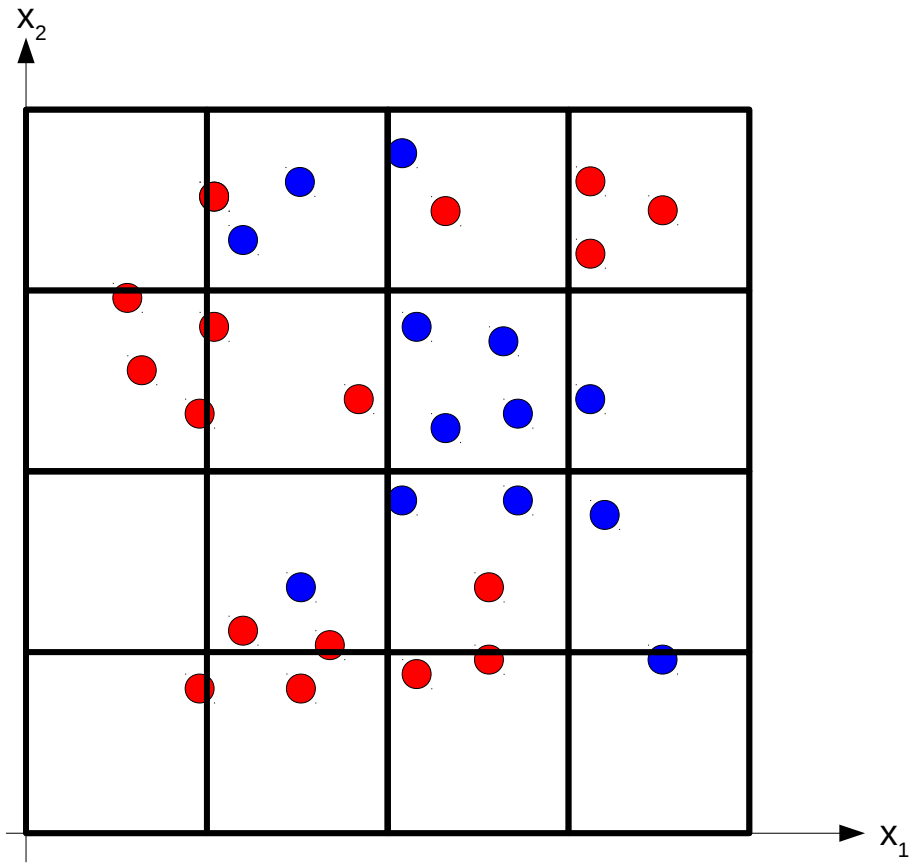
From kNN to Search Trees

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



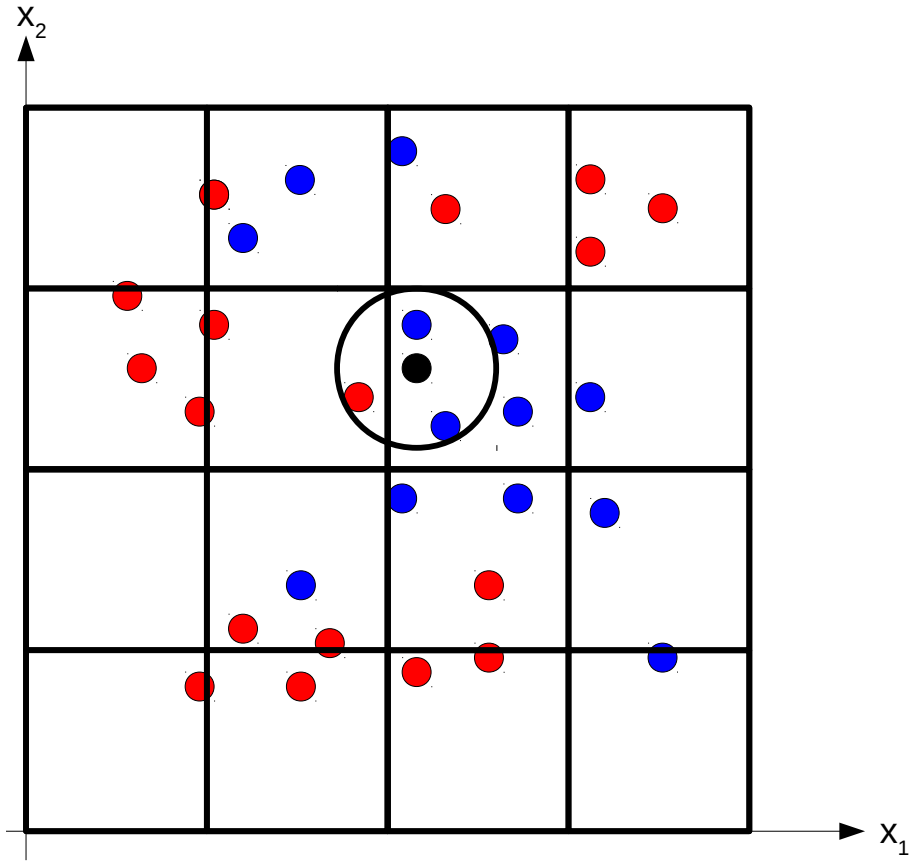
From kNN to Search Trees

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



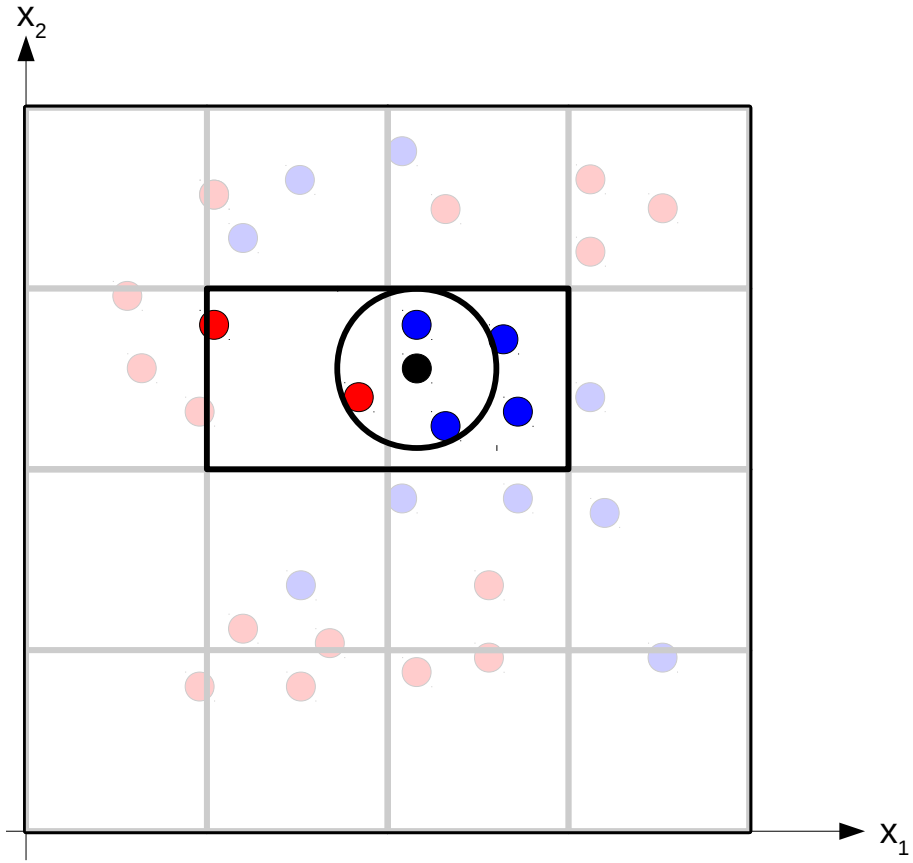
From kNN to Search Trees

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



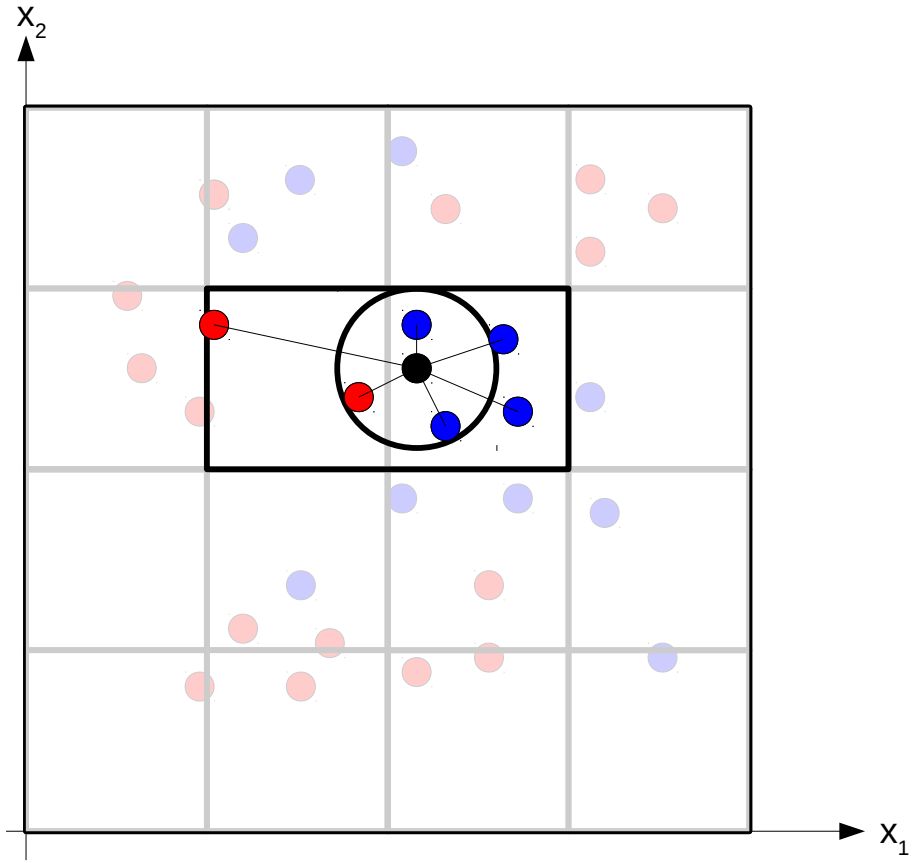
From kNN to Search Trees

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



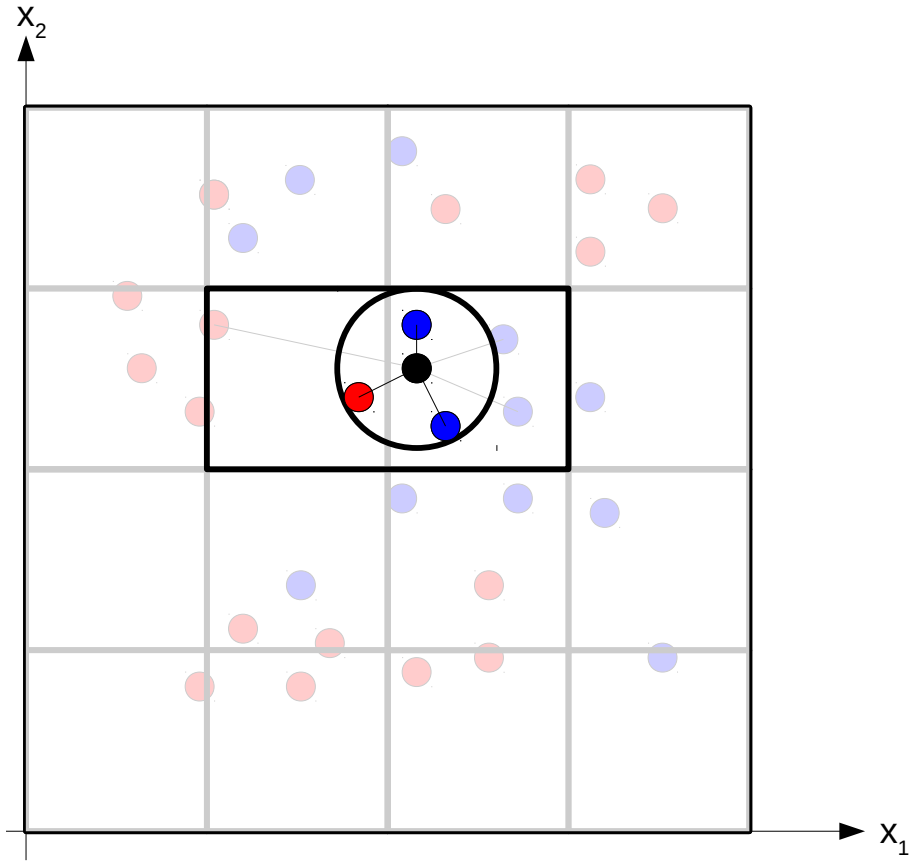
From kNN to Search Trees

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



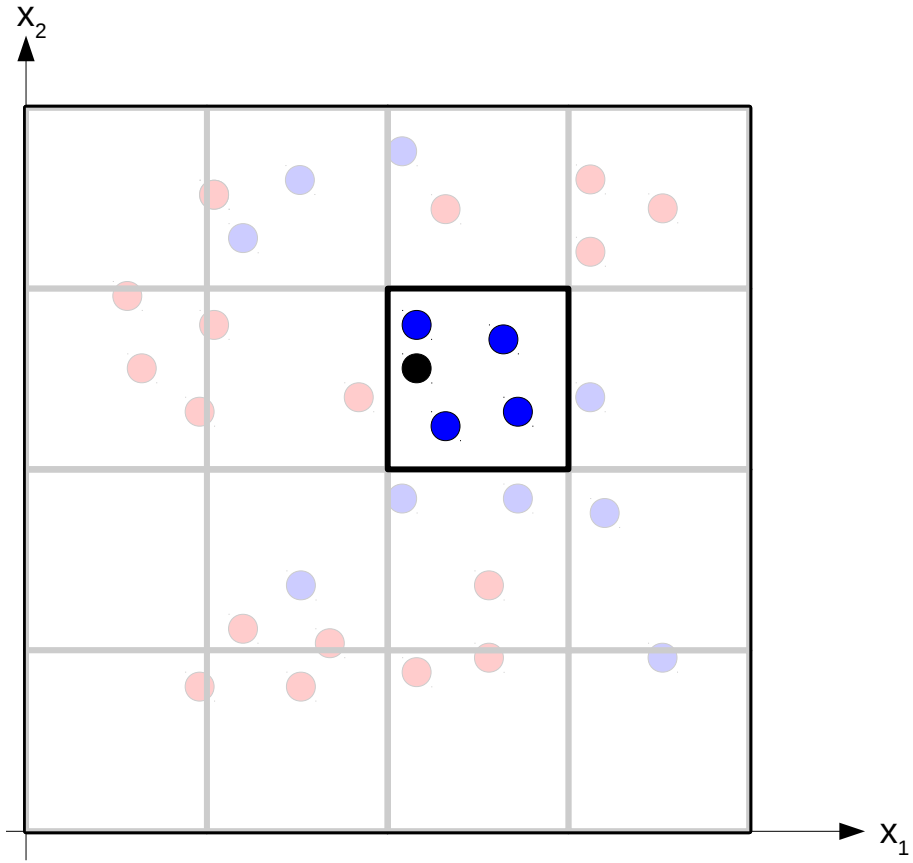
From kNN to Search Trees

- 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



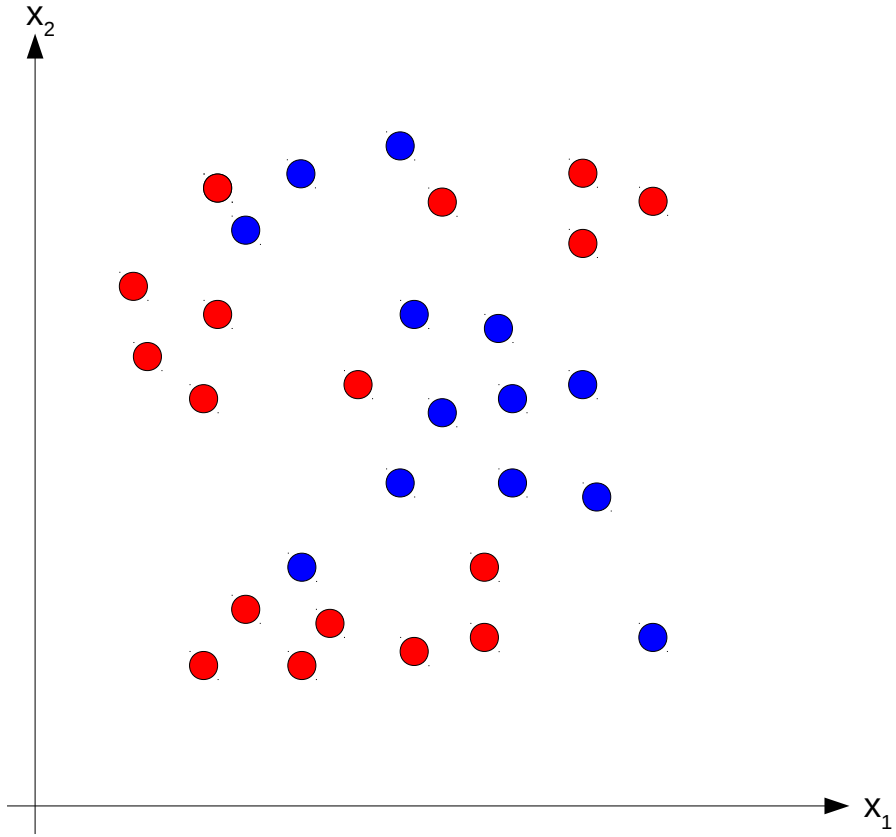
From kNN to Search Trees

- Search trees
  → 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



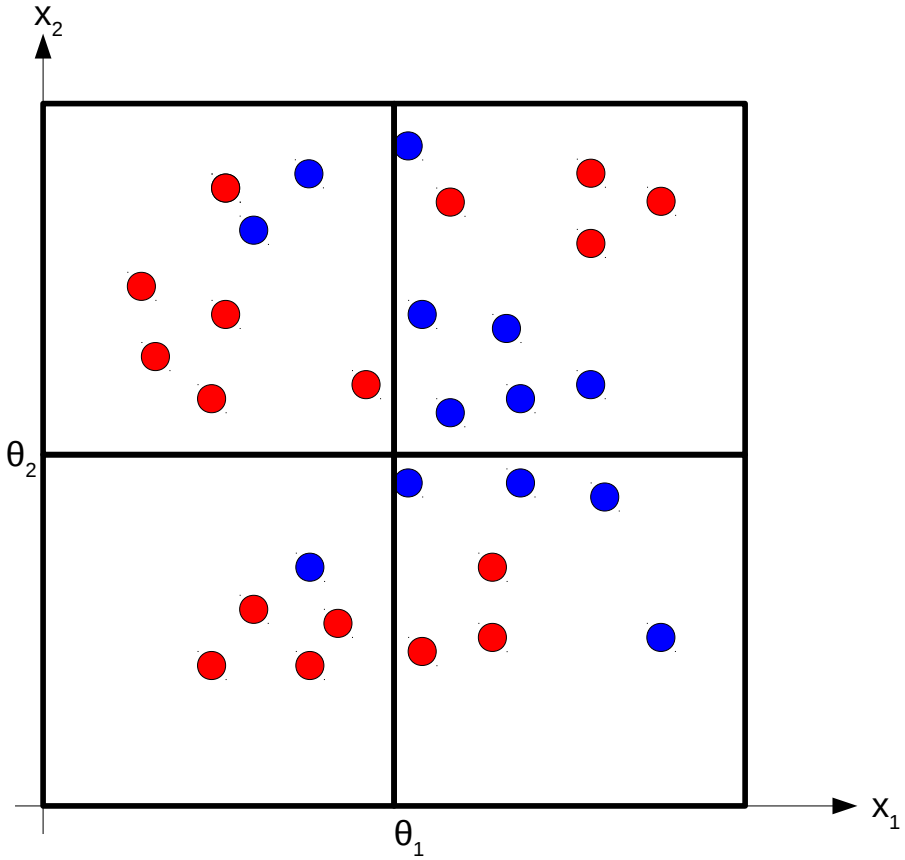
From Search Trees to (Random) Decision Trees

- Cell construction



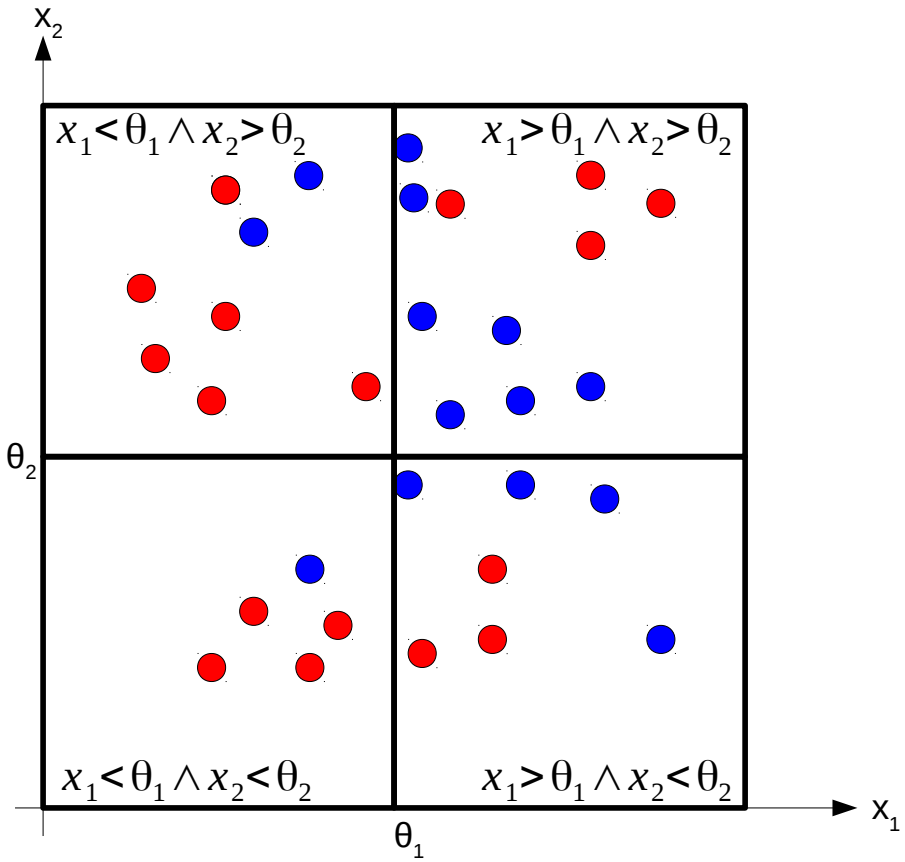
From Search Trees to (Random) Decision Trees

- Cell construction



From Search Trees to (Random) Decision Trees

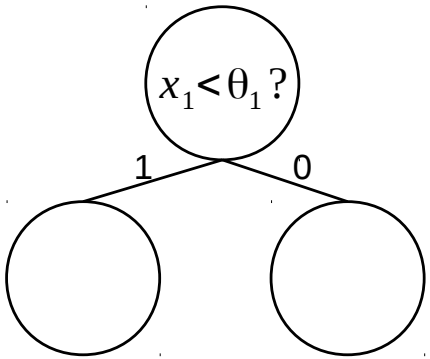
- Cell construction
  $\rightarrow$ Simple threshold operation
  $\rightarrow$ Different threshold definitions (e.g. equi-sized cells, threshold as data median) lead to different search tree variants (e.g. quad-tree, k-D tree).

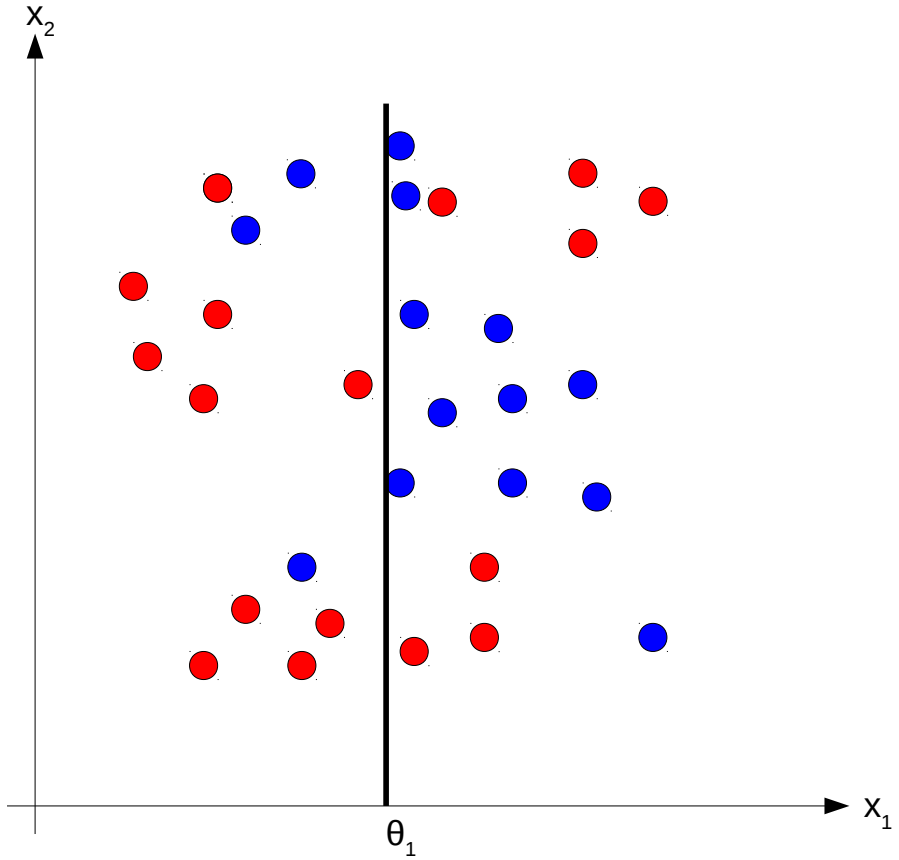


From Search Trees to (Random) Decision Trees

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

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

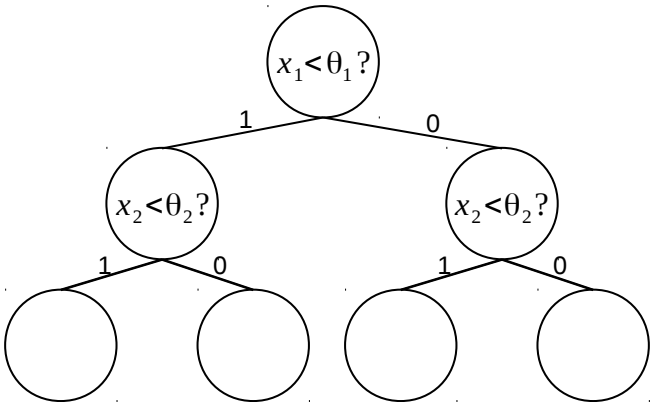


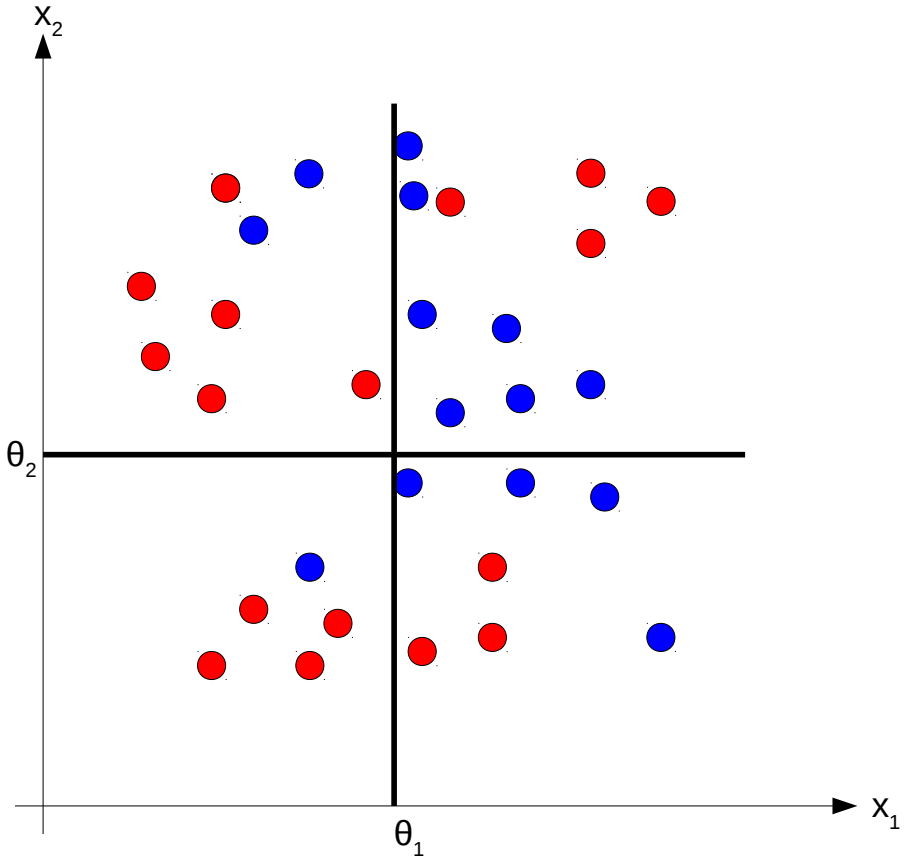


From Search Trees to (Random) Decision Trees

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

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

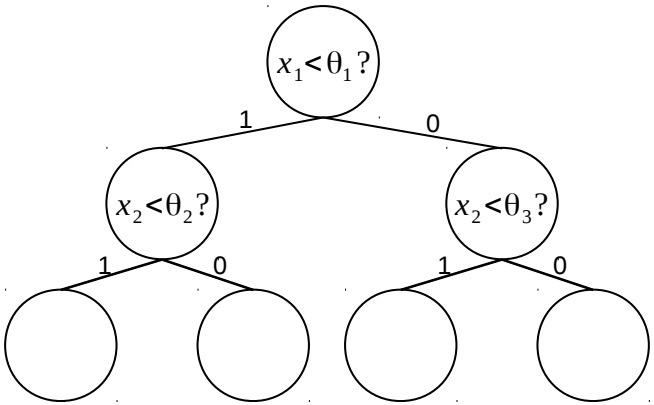


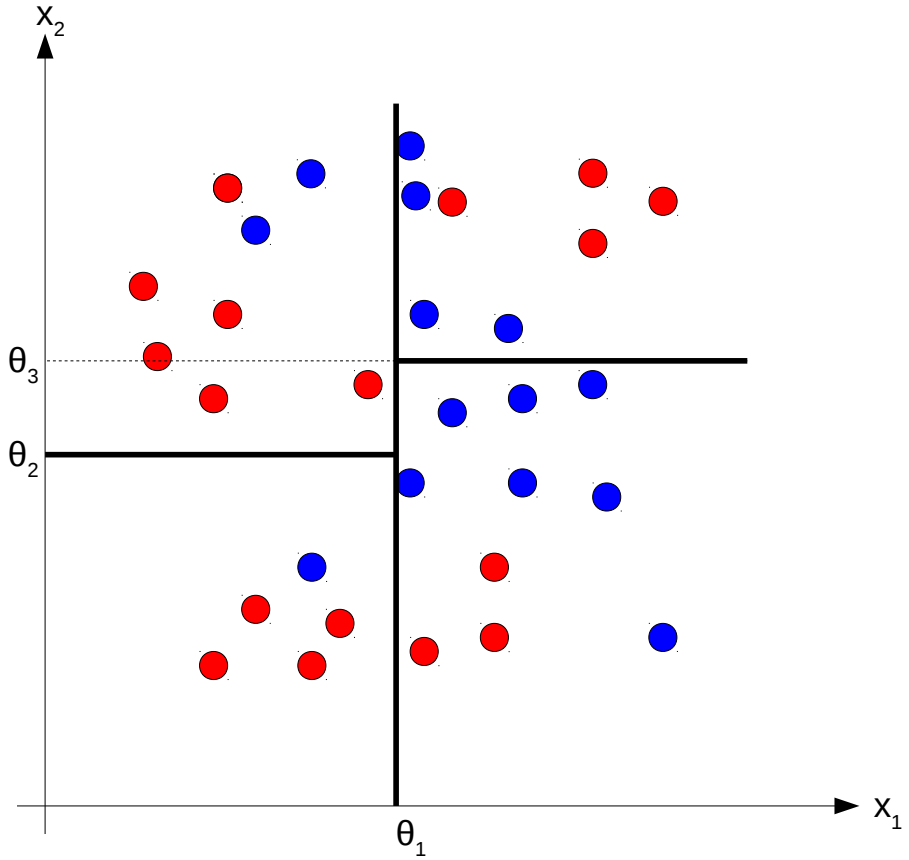


From Search Trees to (Random) Decision Trees

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

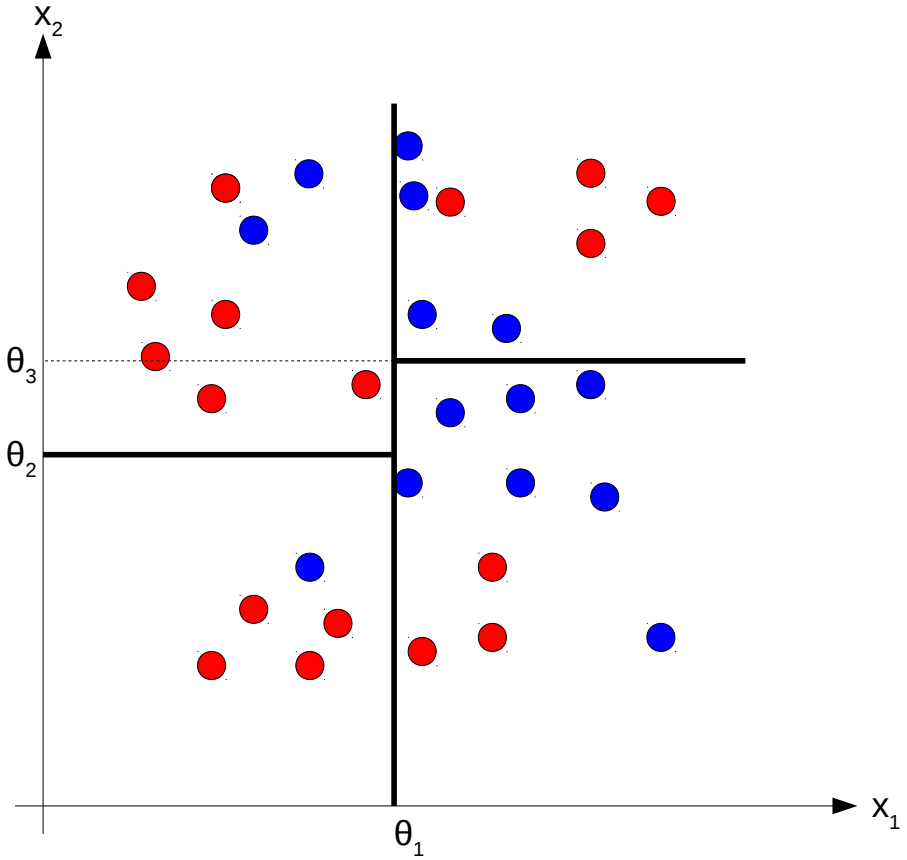
$$t(\mathbf{x}) = \begin{cases} 0 & \text{if } x_1 < \theta_1 \\ 1 & \text{otherwise.} \end{cases}$$



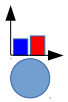


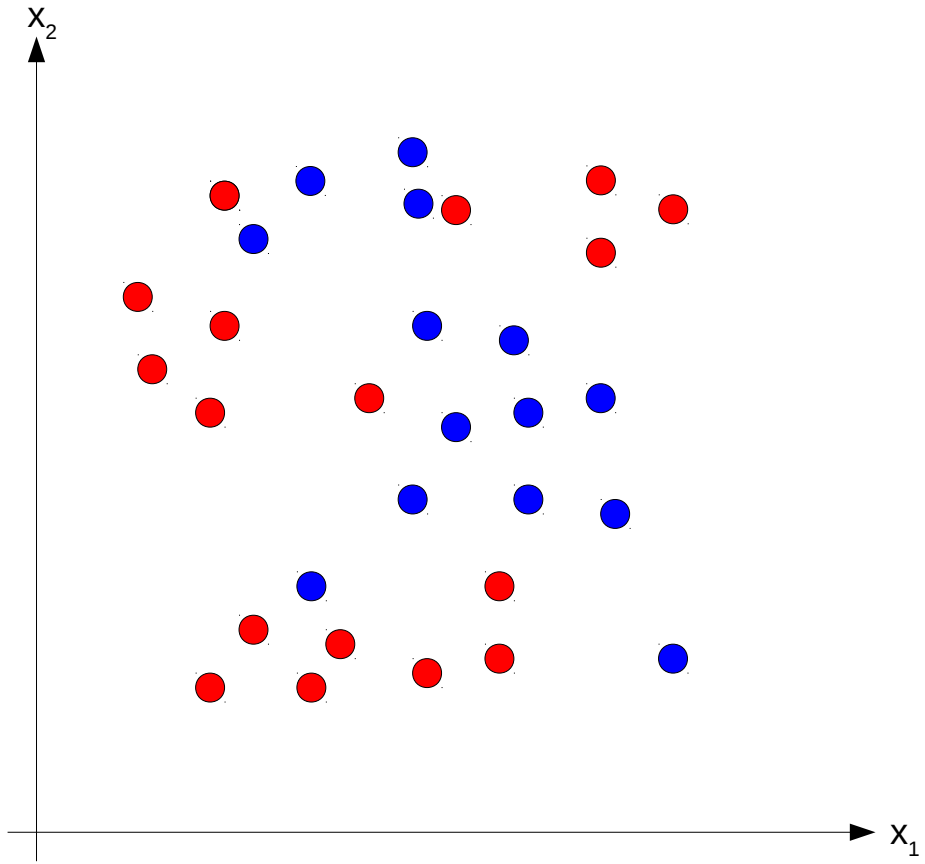
From Search Trees to (Random) Decision Trees

- 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

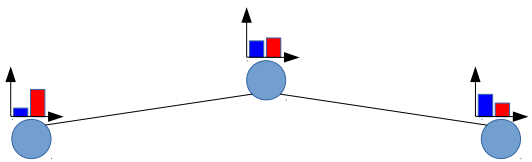


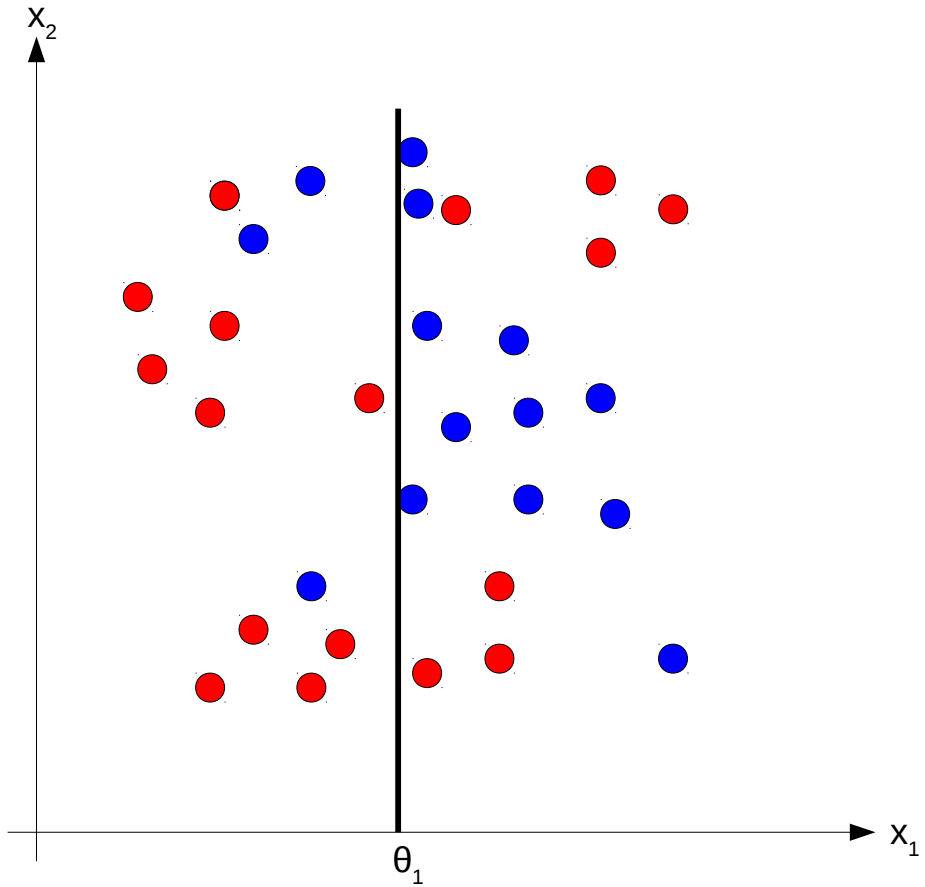
From Search Trees to (Random) Decision Trees



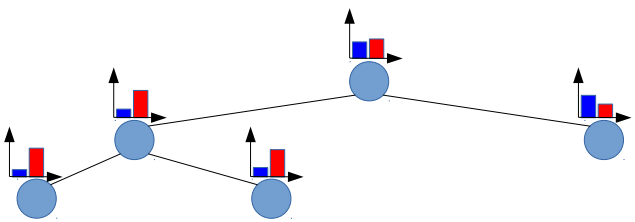


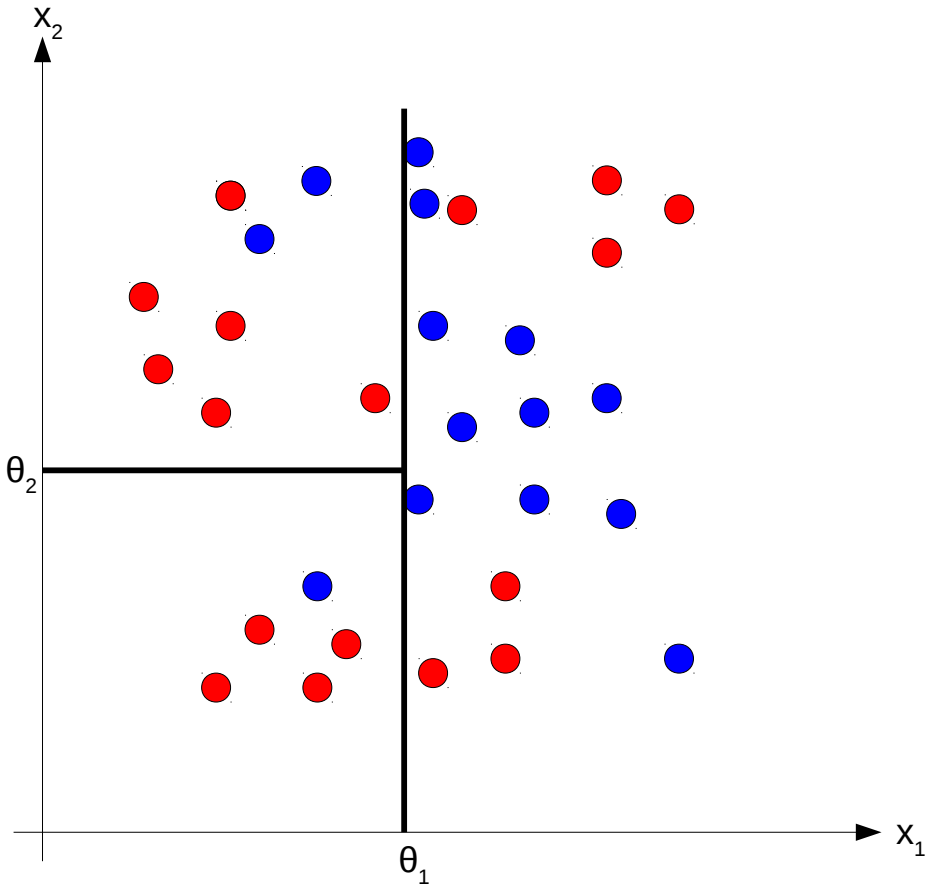
From Search Trees to (Random) Decision Trees



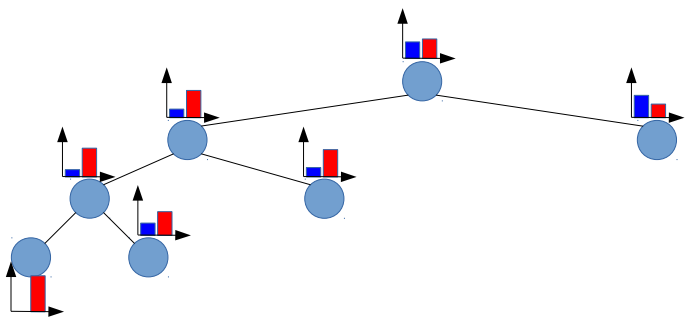


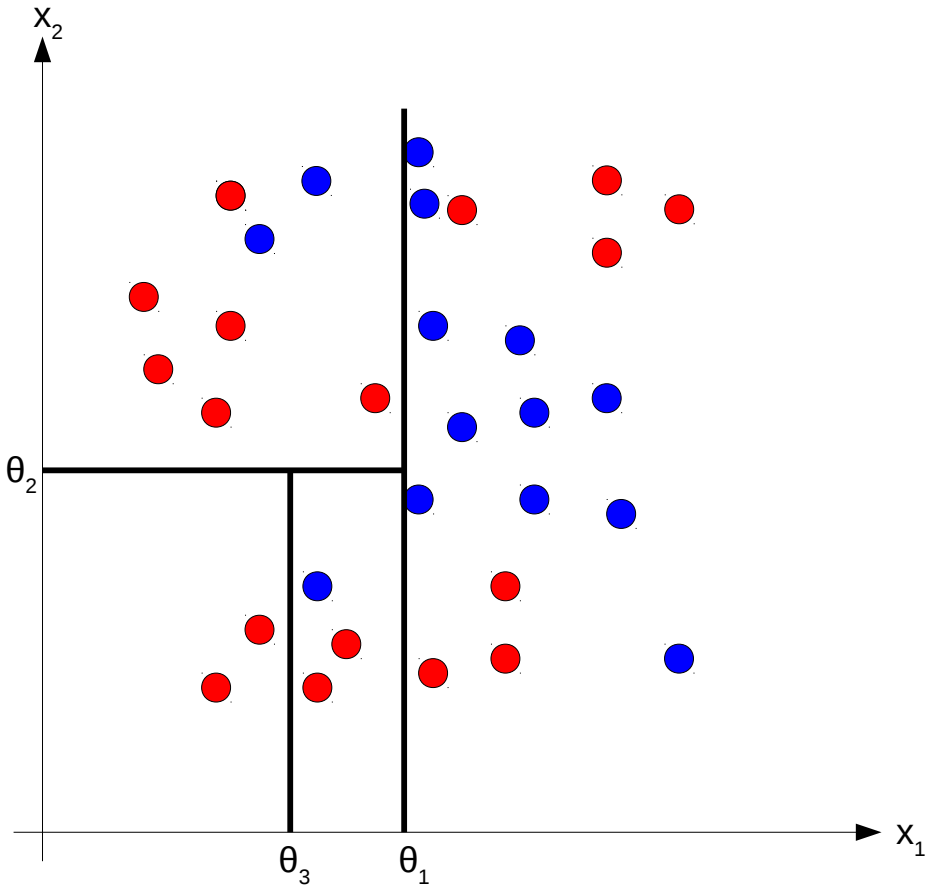
From Search Trees to (Random) Decision Trees



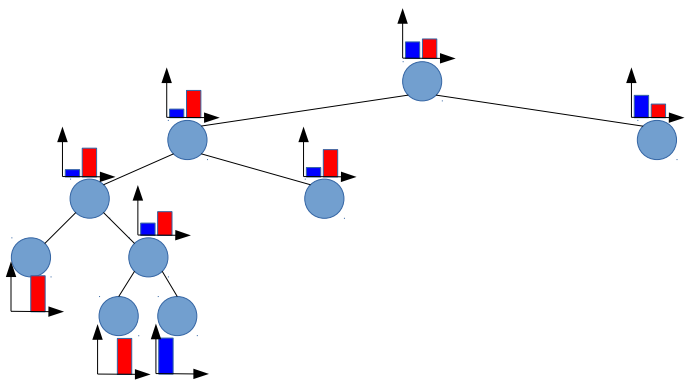


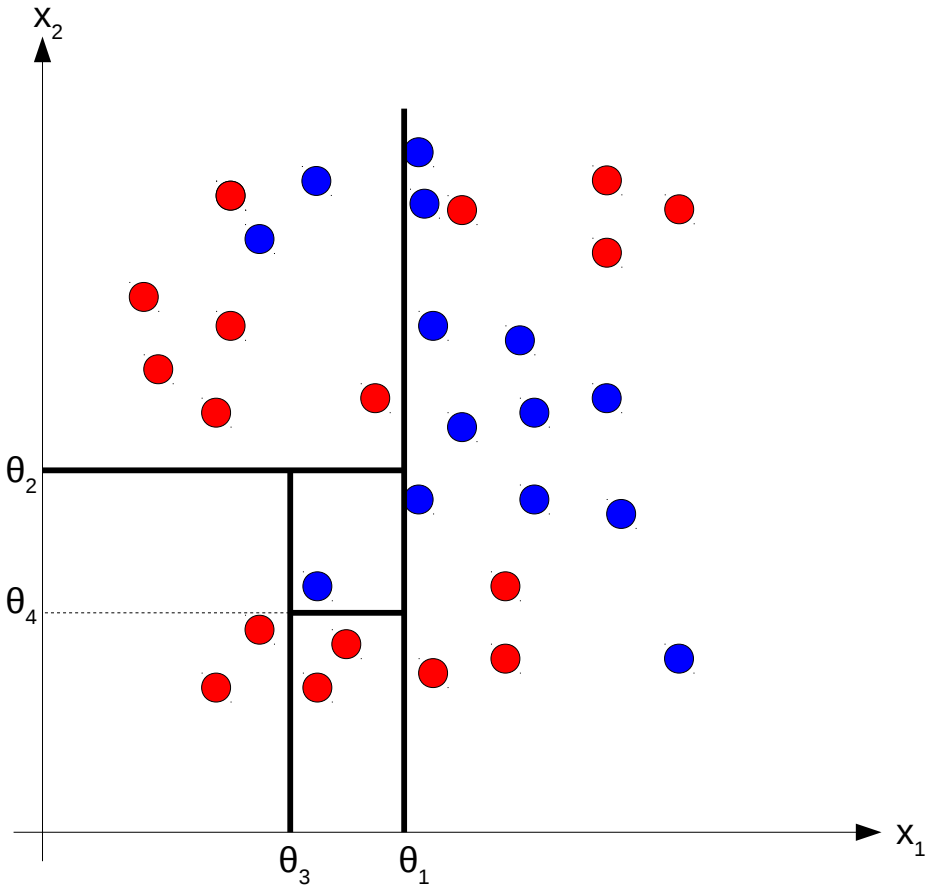
From Search Trees to (Random) Decision Trees



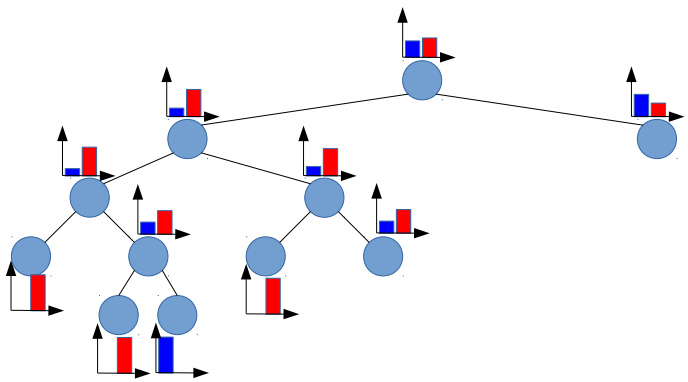


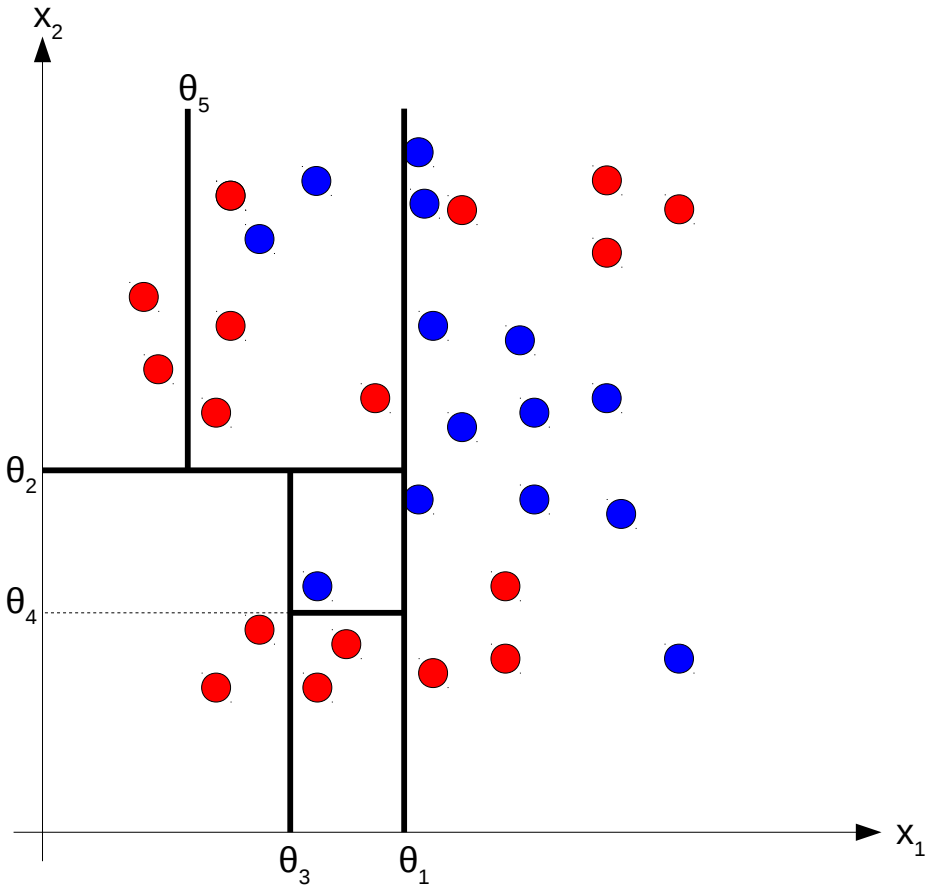
From Search Trees to (Random) Decision Trees



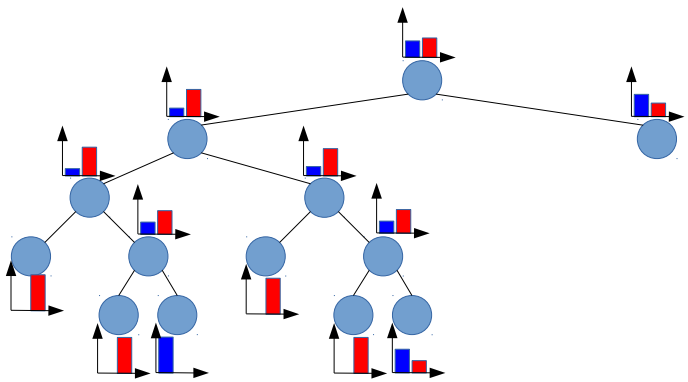


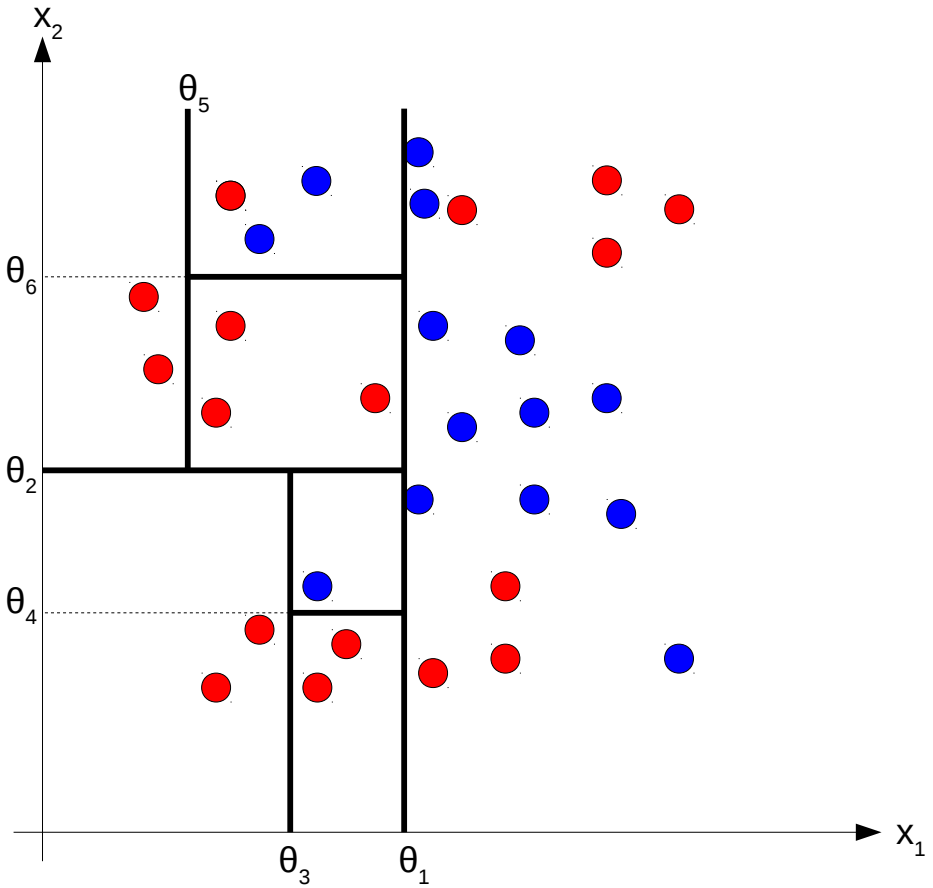
From Search Trees to (Random) Decision Trees



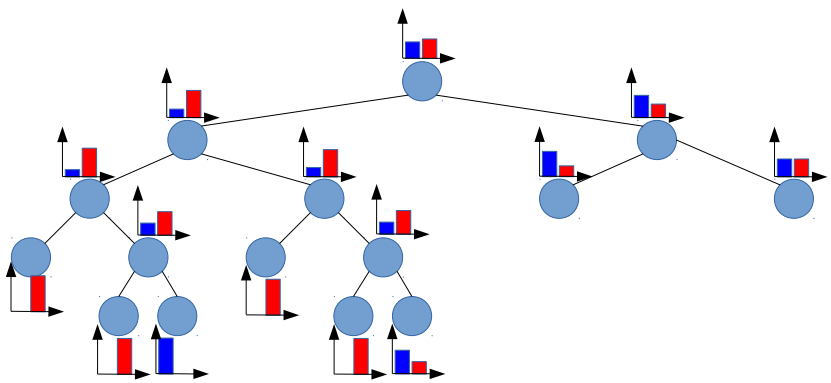


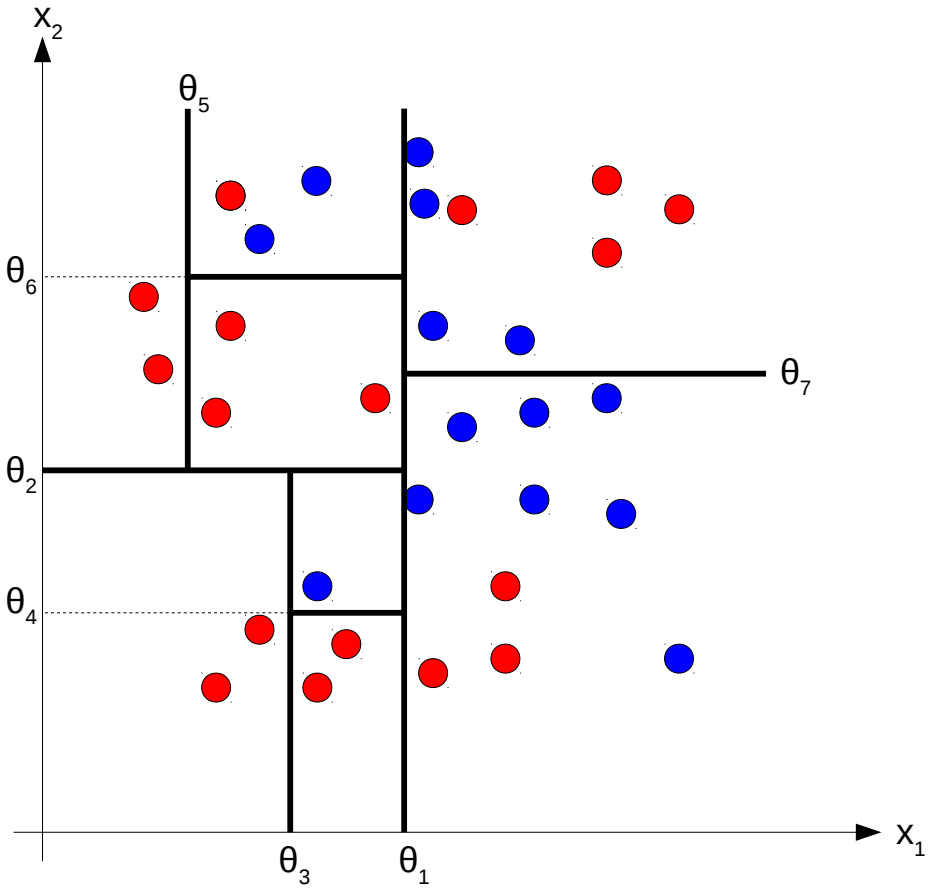
From Search Trees to (Random) Decision Trees



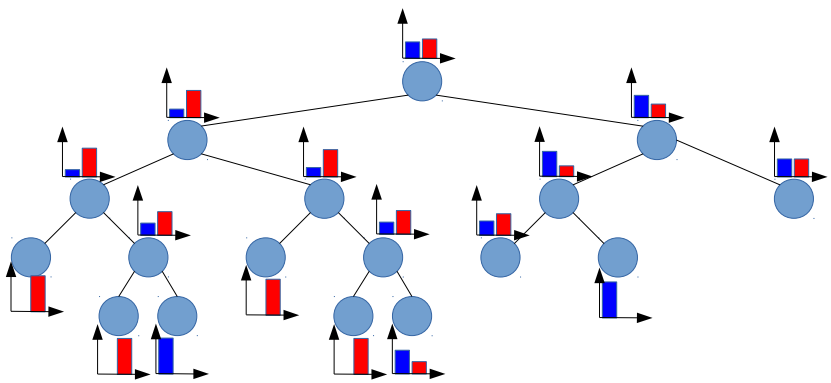


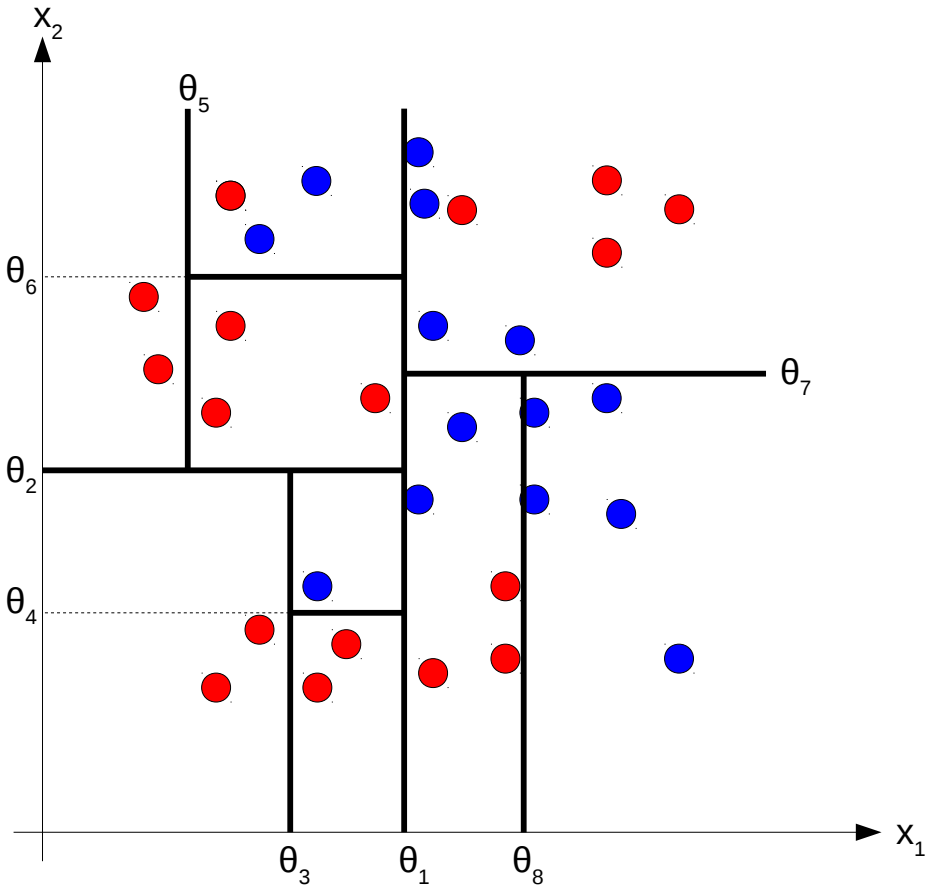
From Search Trees to (Random) Decision Trees



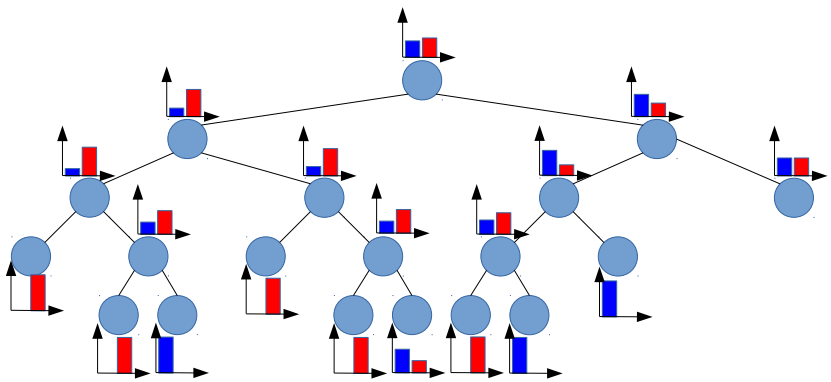


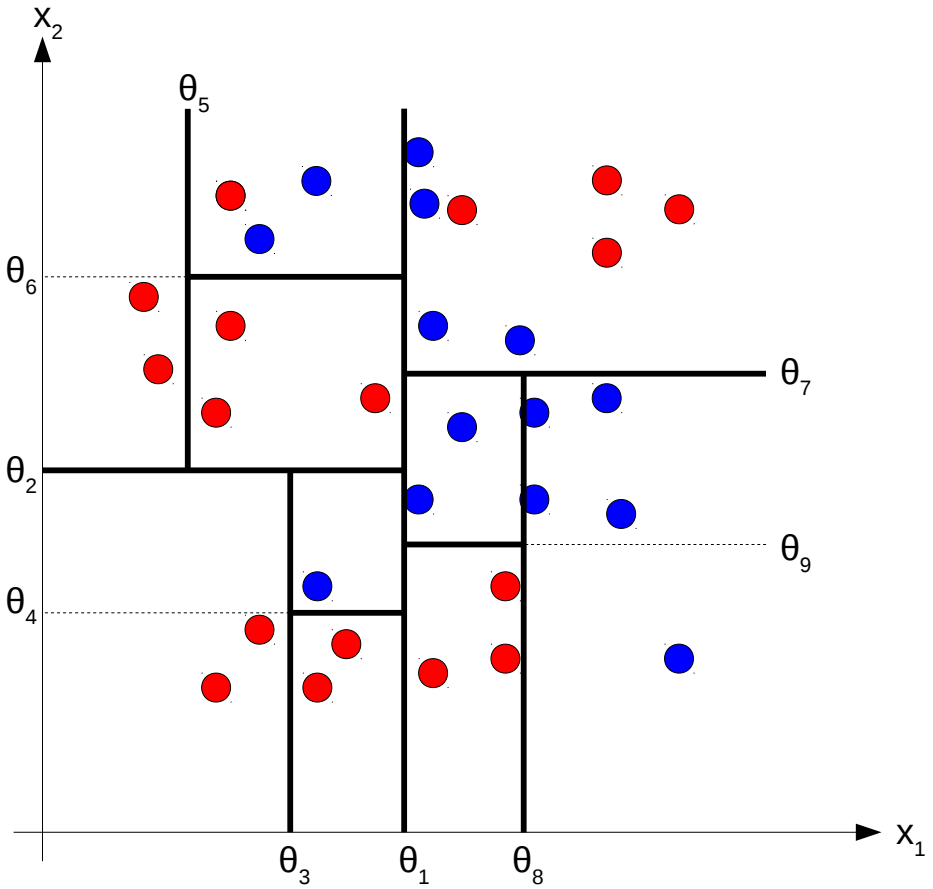
From Search Trees to (Random) Decision Trees



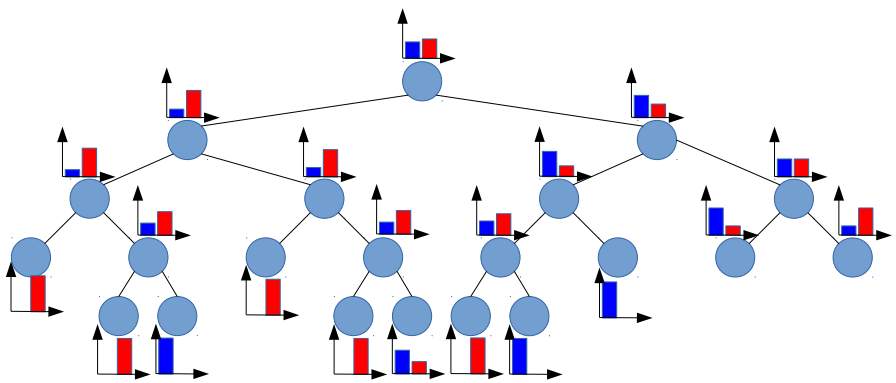


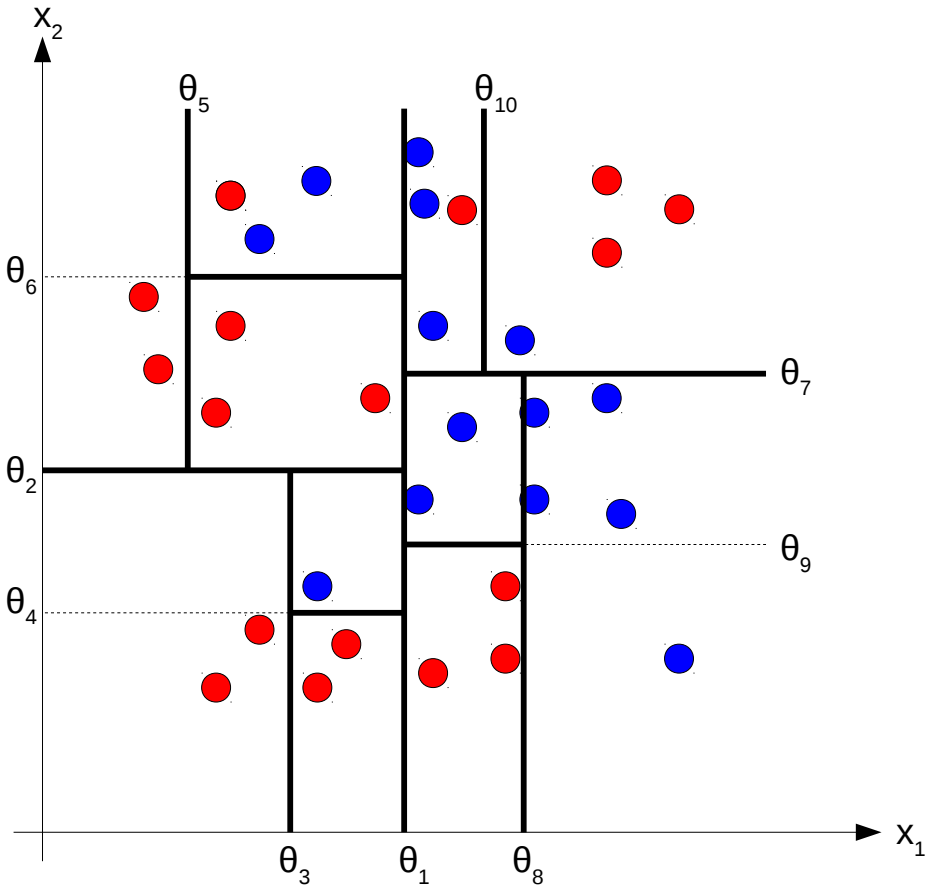
From Search Trees to (Random) Decision Trees



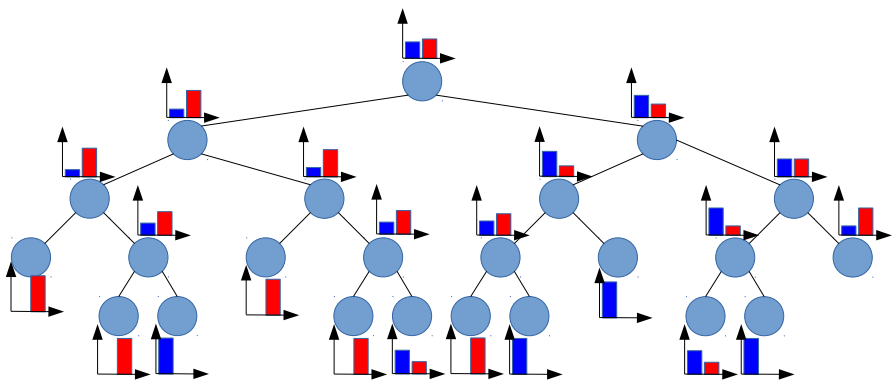


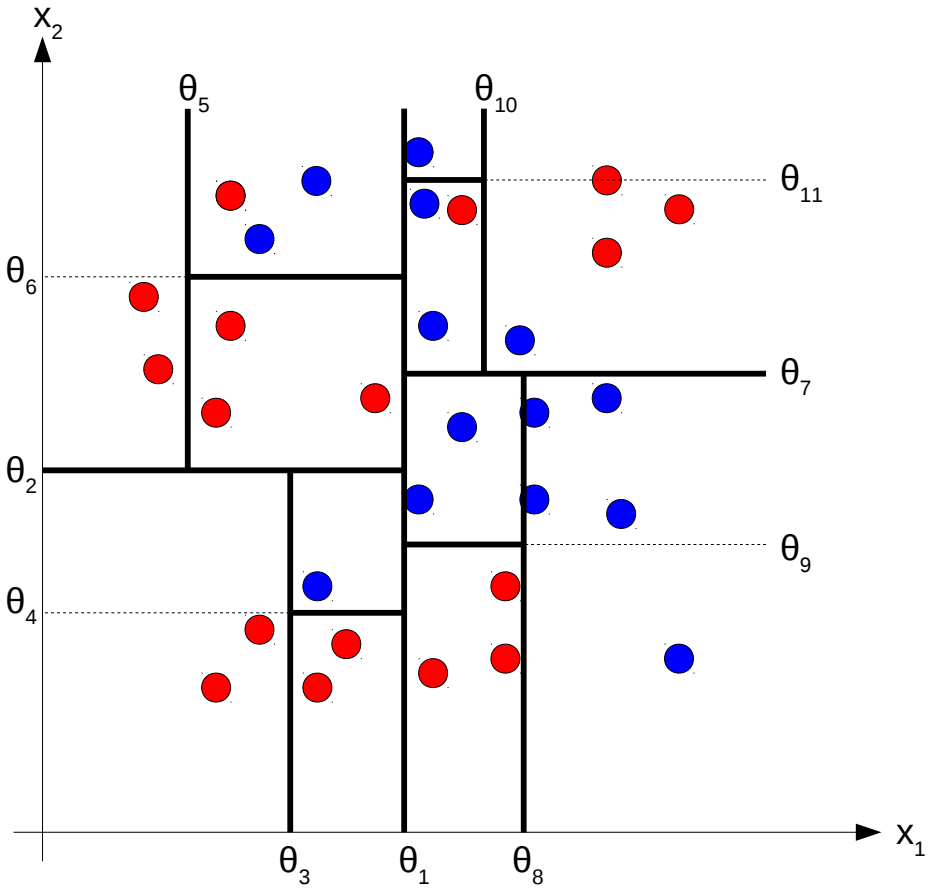
From Search Trees to (Random) Decision Trees



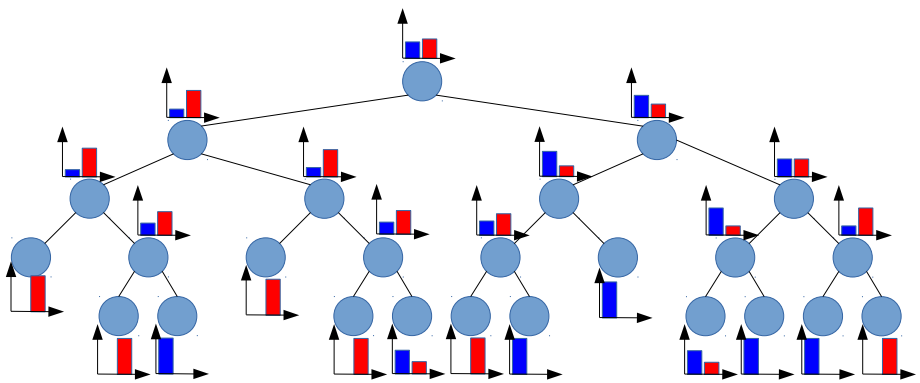


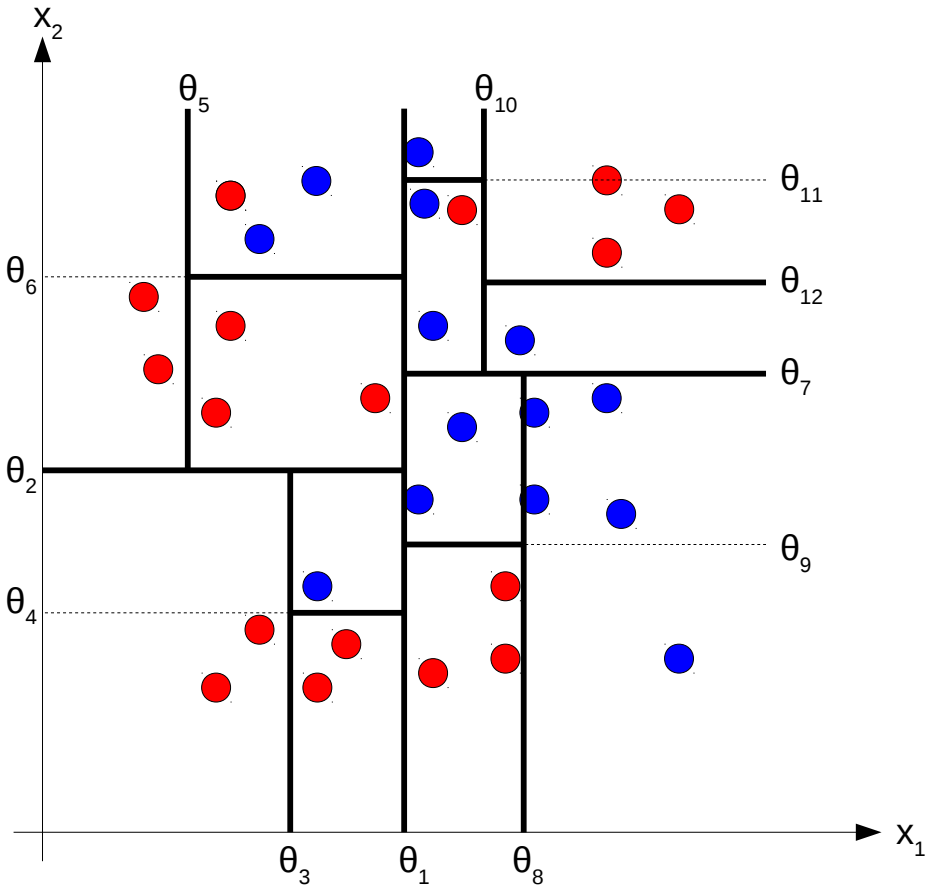
From Search Trees to (Random) Decision Trees





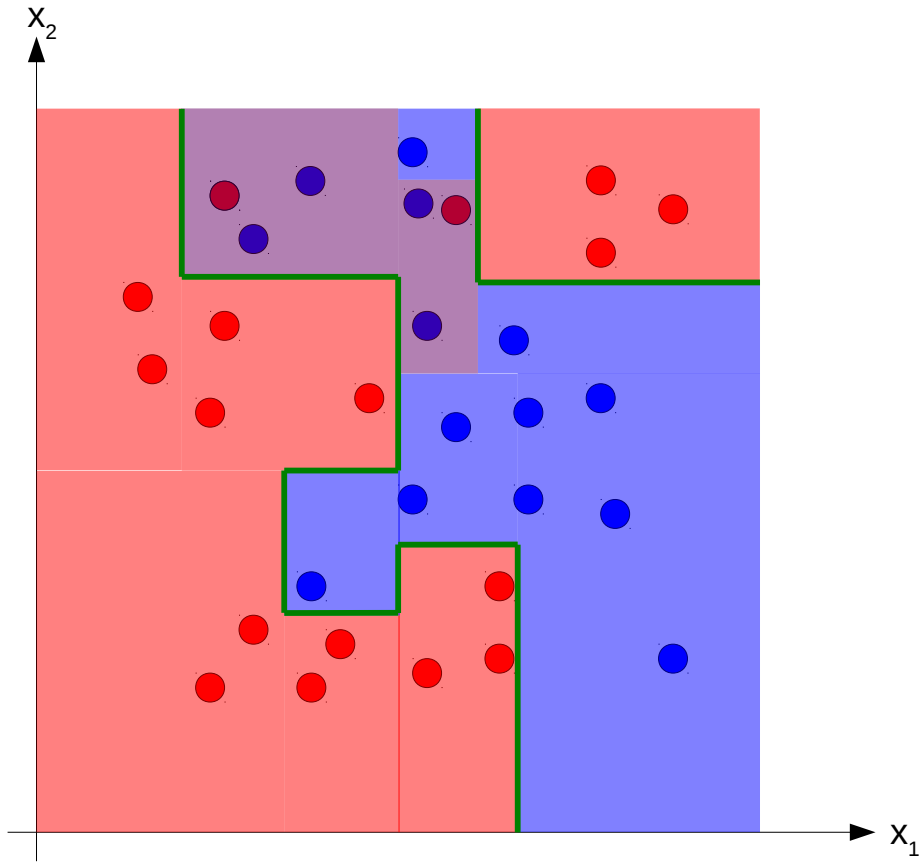
From Search Trees to (Random) Decision Trees





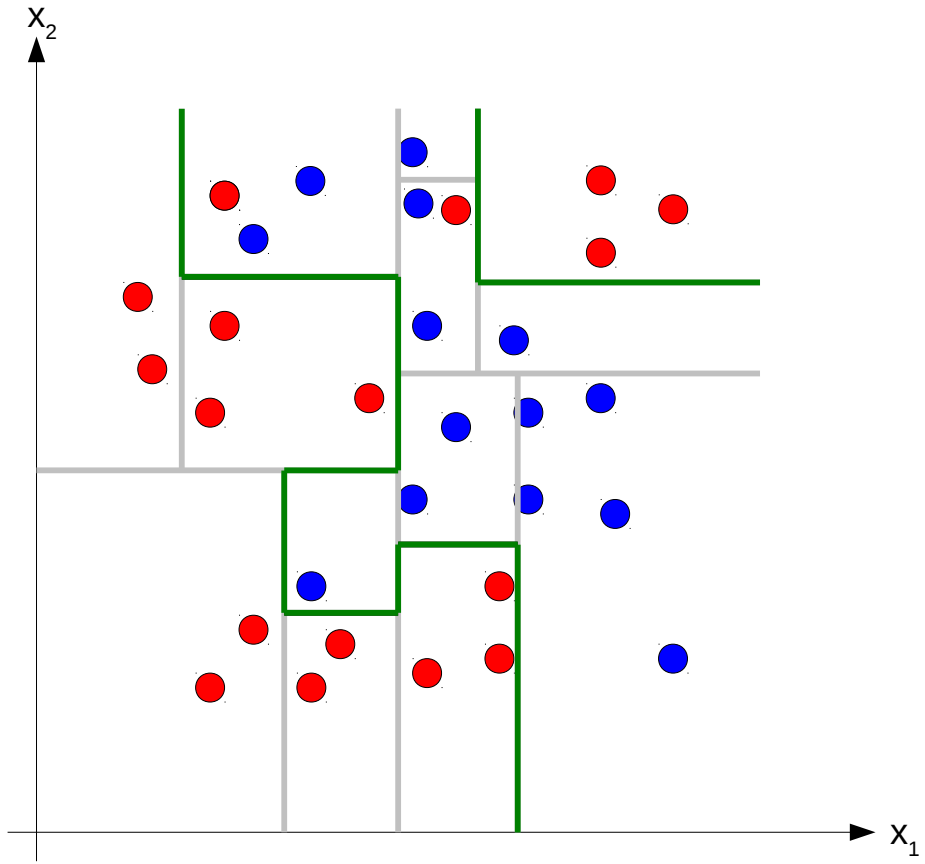
From Search Trees to (Random) Decision Trees

- 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



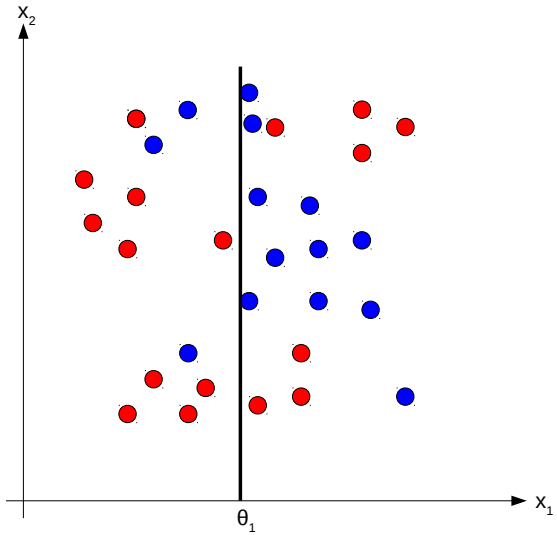
From Search Trees to (Random) Decision Trees

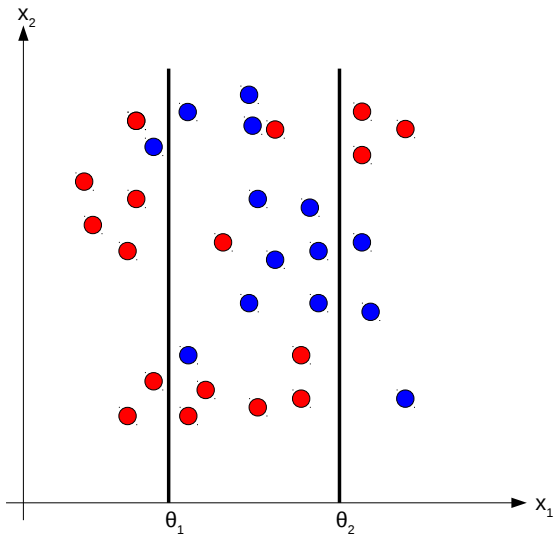
Other node tests are possible:

- Axis-aligned:

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

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





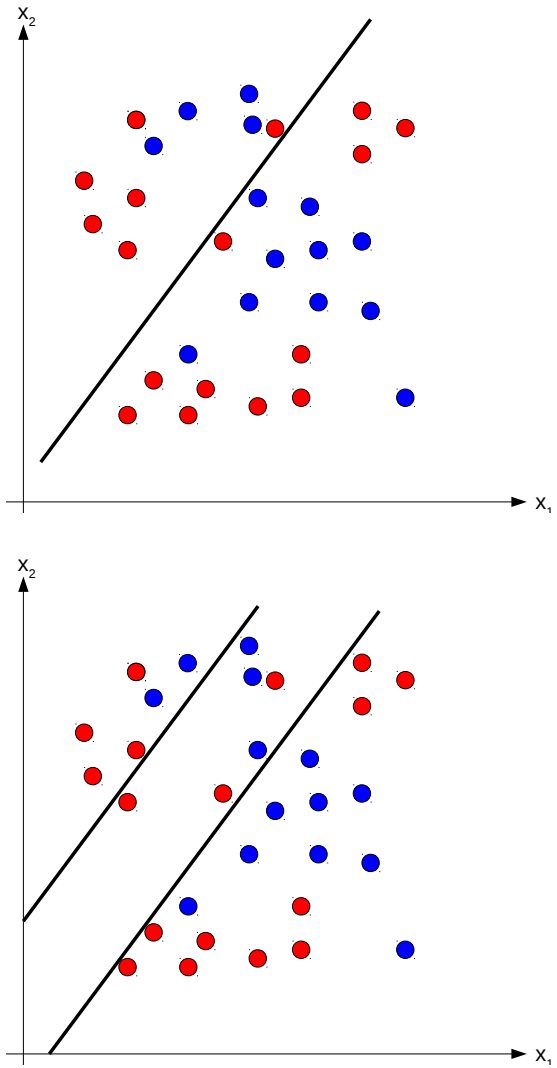
From Search Trees to (Random) Decision Trees

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

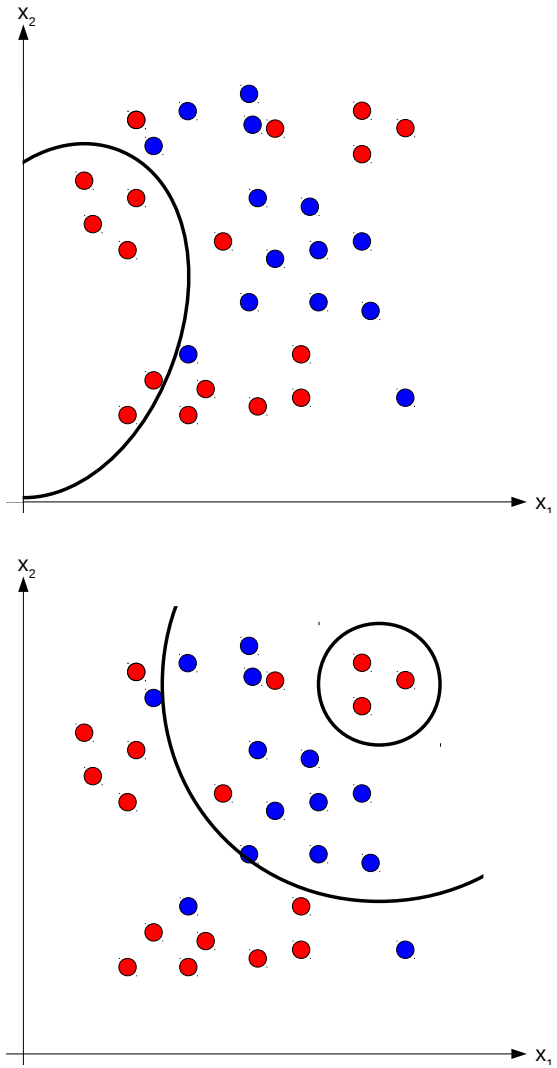
Other node tests are possible:

- Axis-aligned
- Linear
- Conic section:

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

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

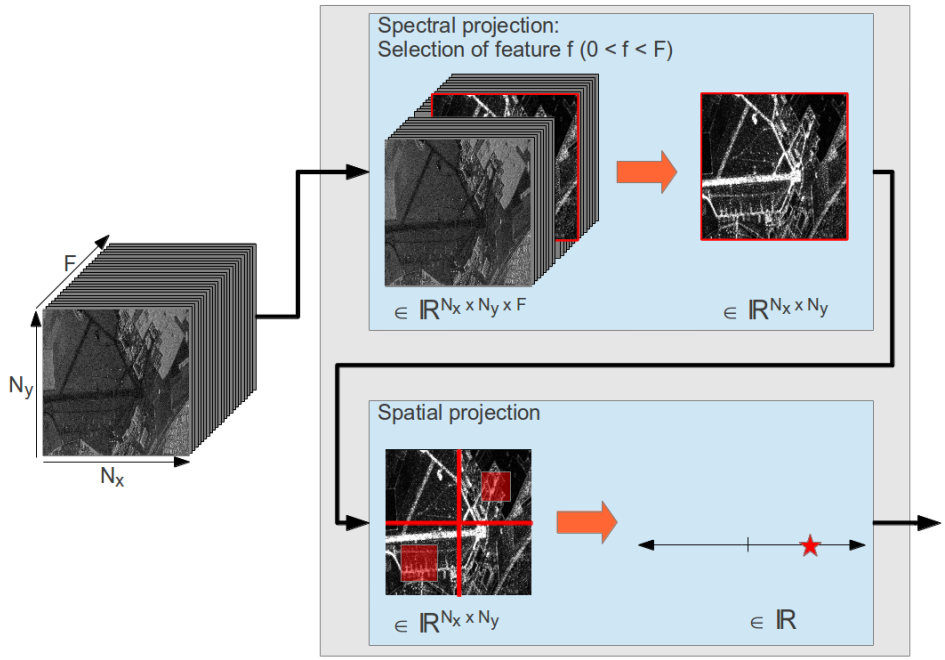
$$t(\mathbf{x}) = \begin{cases} 0 & \text{if } \theta_1 < \tilde{\mathbf{x}}^T \psi \tilde{\mathbf{x}} < \theta_2 \\ 1 & \text{otherwise.} \end{cases}$$



From Search Trees to (Random) Decision Trees

Other node tests are possible:

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



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.

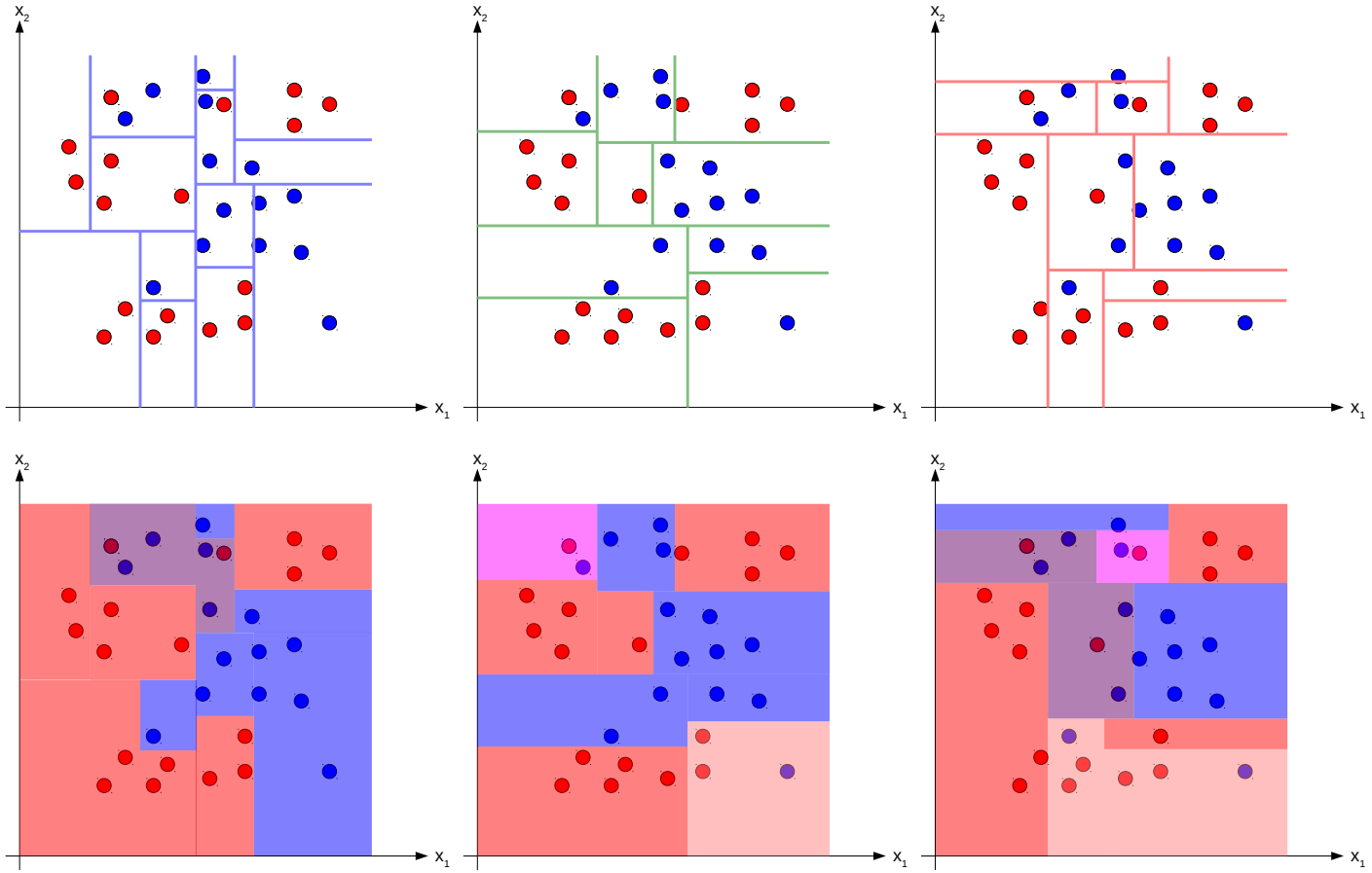
From (Random) Decision Trees to Random Forests

Disadvantages

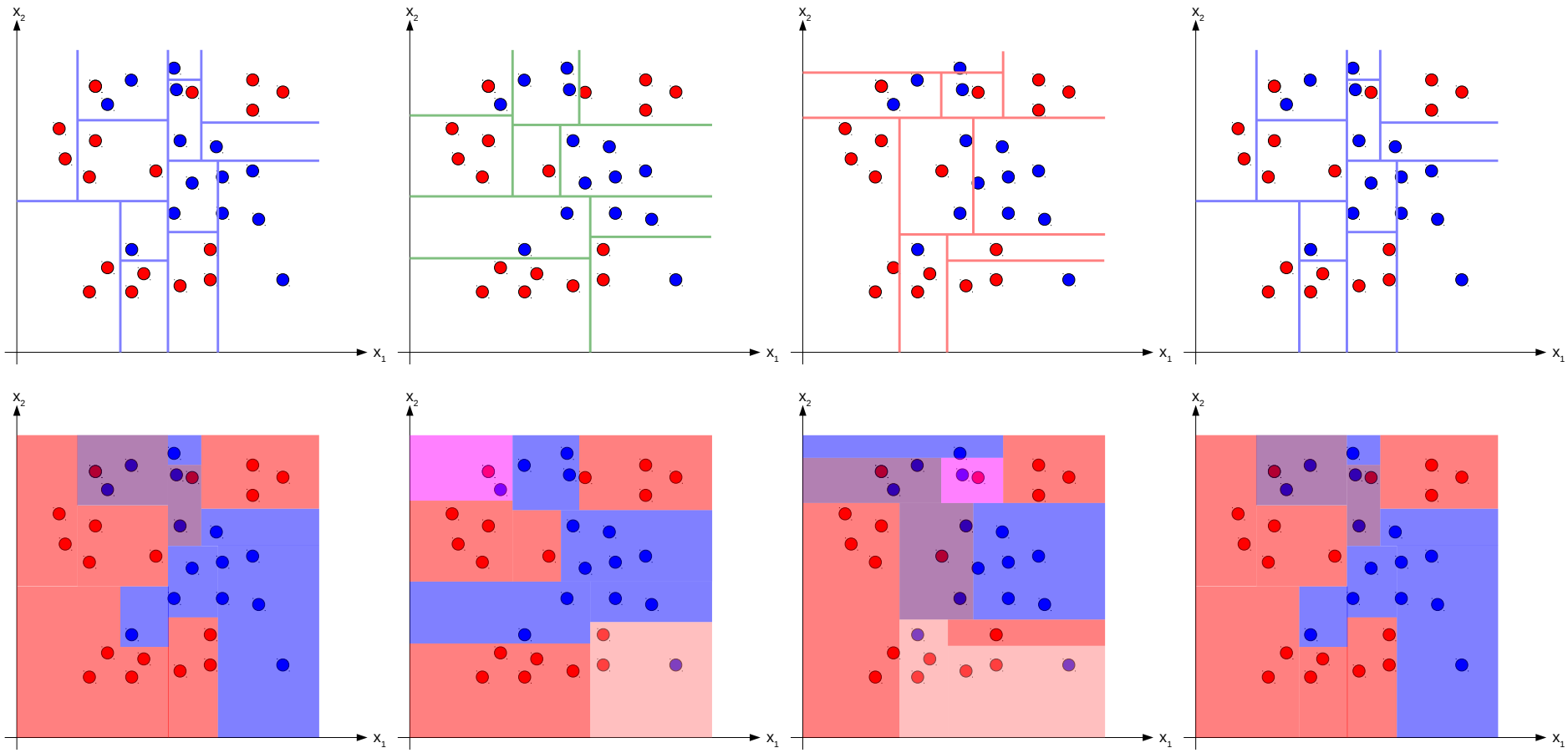
- Optimized by greedy algorithms
  → A chain of individually optimal decisions, might not lead to an overall optimum
- The optimal solution (i.e. decision boundary) might not be part of the model class (e.g. piece-wise linear and axis-aligned functions)
- Prone to overfitting
- Model capacity depends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

How to
→ keep (most) of the advantages
→ getting rid of (most) disadvantages?

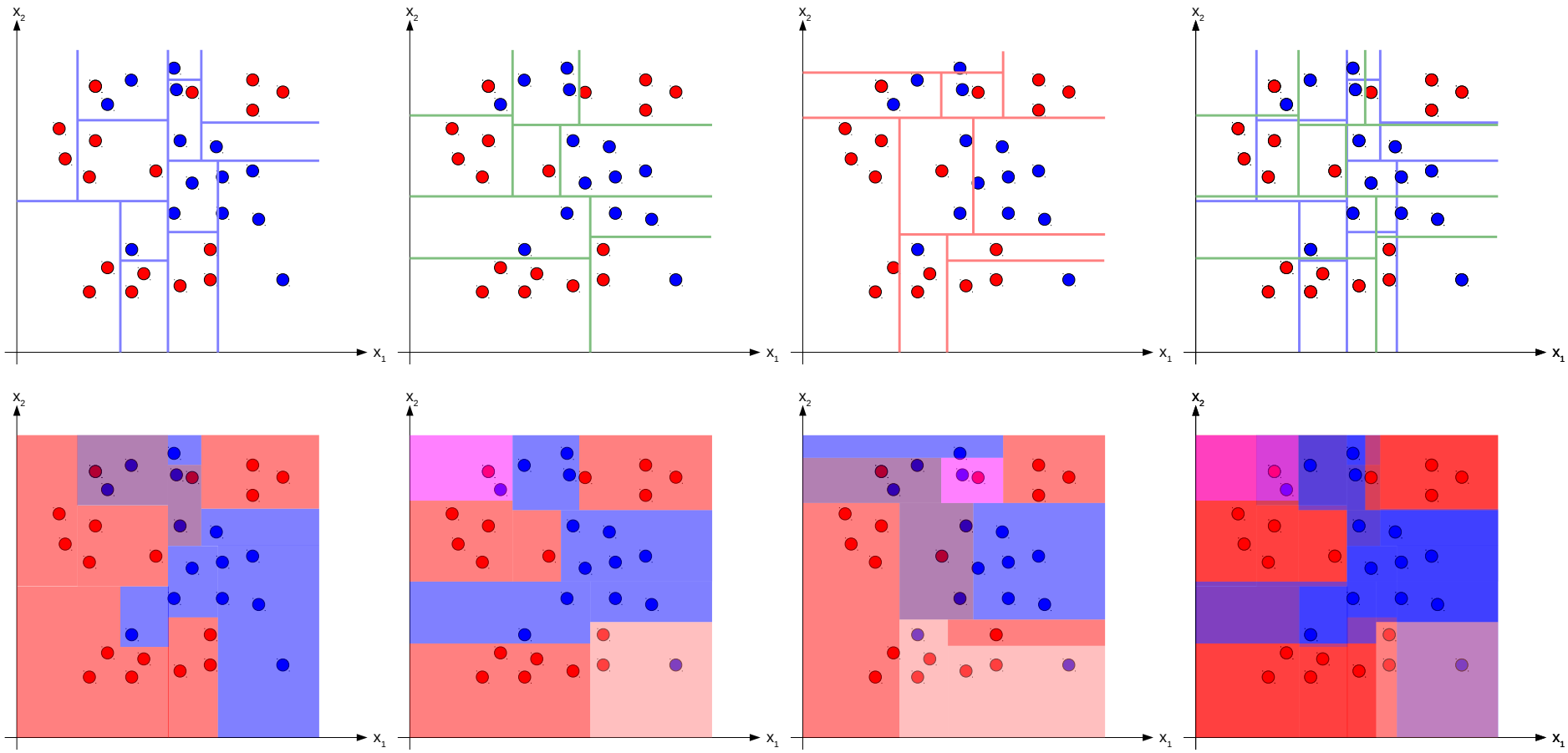
From (Random) Decision Trees to Random Forests



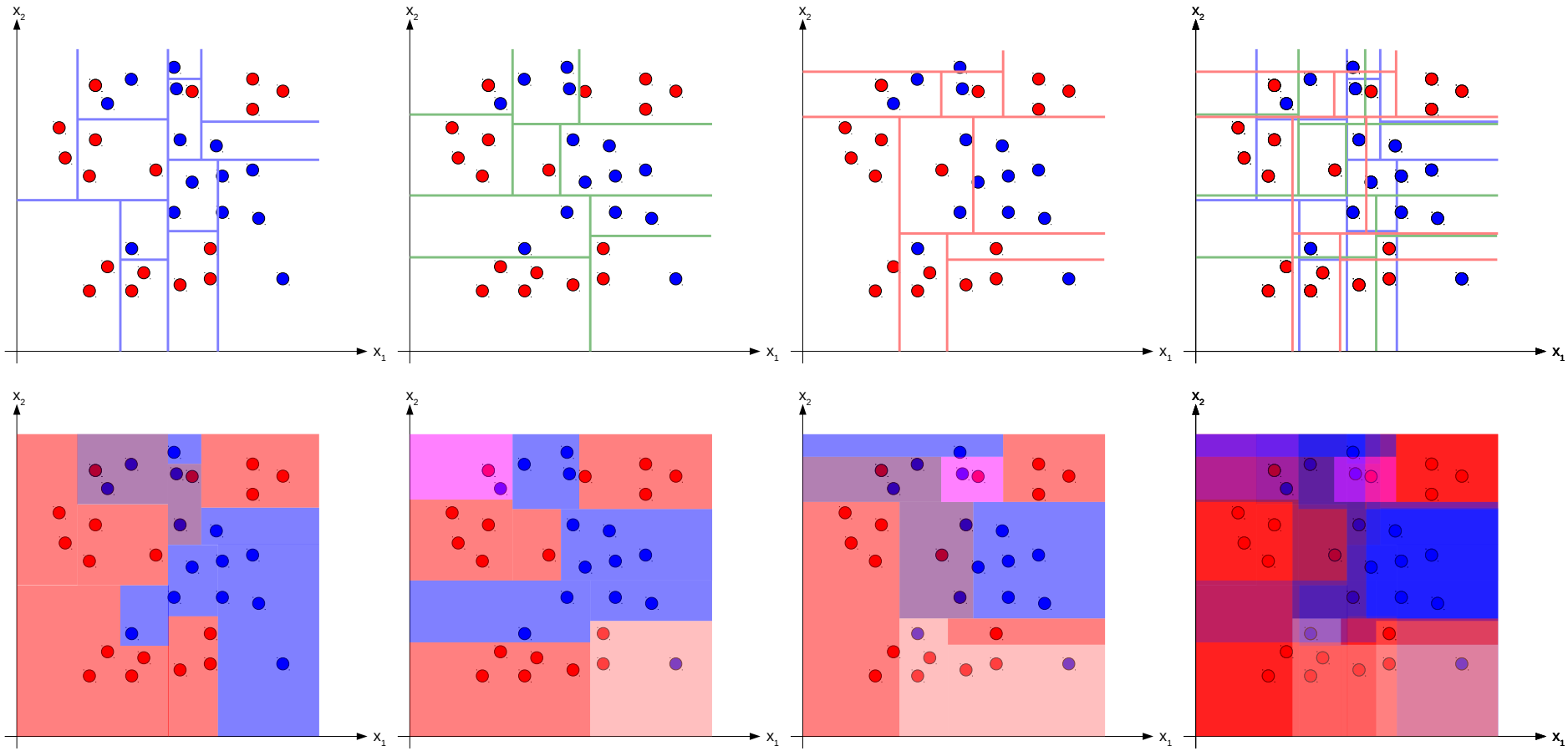
From (Random) Decision Trees to Random Forests



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

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

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

$\phi : \mathbb{D} \rightarrow \mathbb{R}^n \equiv$ Implicit feature extraction

e.g. $x \in \mathbb{R}^n : \ \phi : \mathbb{R}^n \rightarrow \mathbb{R}^2, x \mapsto (x_i, x_j)^T$

$\psi : \mathbb{R}^n \rightarrow \mathbb{R} \equiv$ Feature fusion

e.g. $\phi(x) \in \mathbb{R}^2 : \ \psi : \mathbb{R}^2 \rightarrow \mathbb{R}, \phi(x) \mapsto [\psi_i, \psi_j] \cdot \phi(x)$

$\xi : \mathbb{R} \rightarrow \{0, 1\} \equiv$ Child node assignment

e.g. thresholding

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

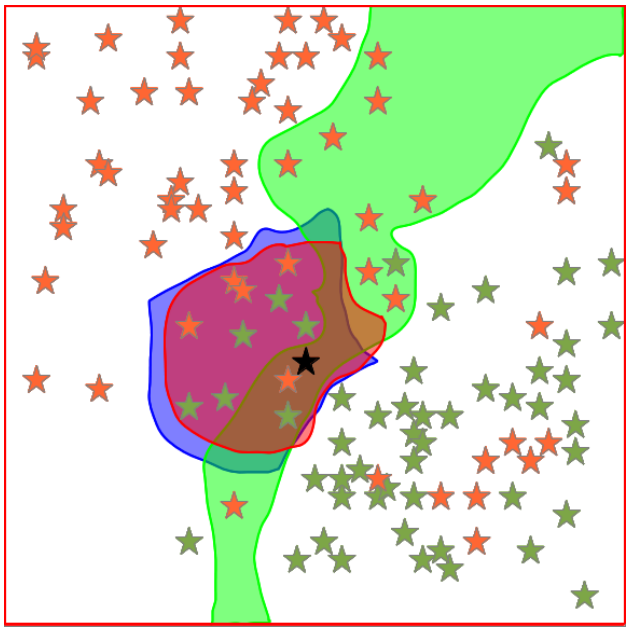
- What kind of node tests?
  $\rightarrow$ For images, for other data spaces than $\mathbb{R}^n$
- How to select node tests?
  $\rightarrow$ How to measure good tests?
- How to limit model capacity (tree height, tree number)?
  $\rightarrow$ The more the better? What about overfitting?
- How to fuse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

Tree Fusion

$t = 1$ $t = 2$ ... $t = T$

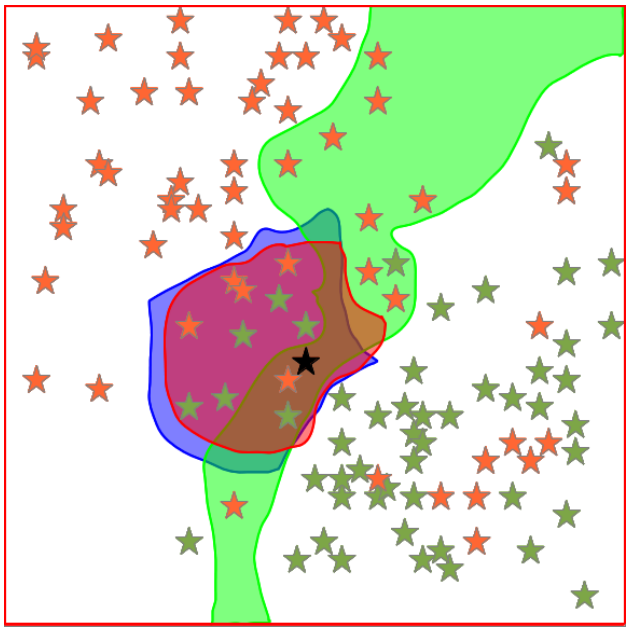
- Query sample is propagated through all trees
- Reaches exactly one leaf in each tree
- Information about target variable assigned to these leafs need to be combined

Tree Fusion



How to combine information assigned to individual leafs?

Tree Fusion

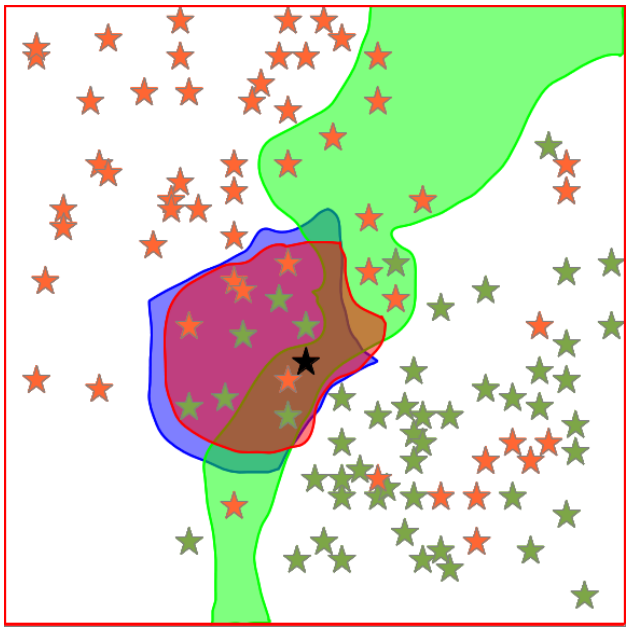


Random Forests

$$P(c|x) = \frac{1}{T}\sum_{t=1}^{T}\delta(M(n_t), c)$$

$M(n)$ is dominant class
of samples in leaf node n

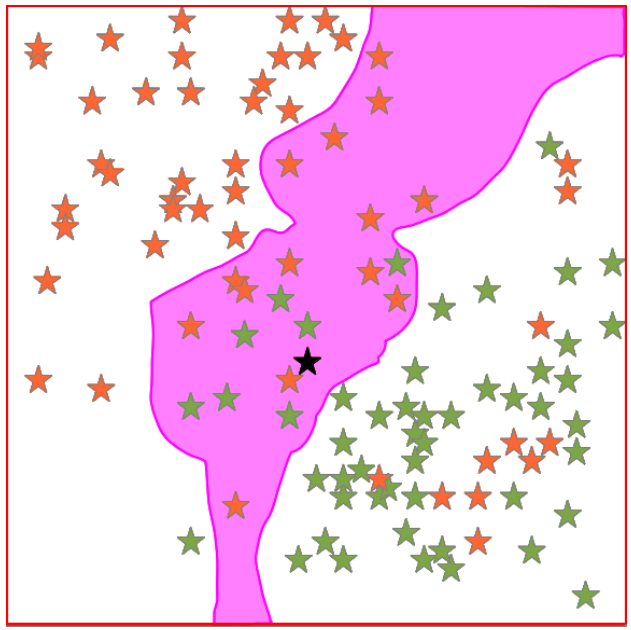
Tree Fusion



Randomized Trees

$$P(c|x) = \frac{1}{T}\sum_{t=1}^{T} P_t(c|x)$$

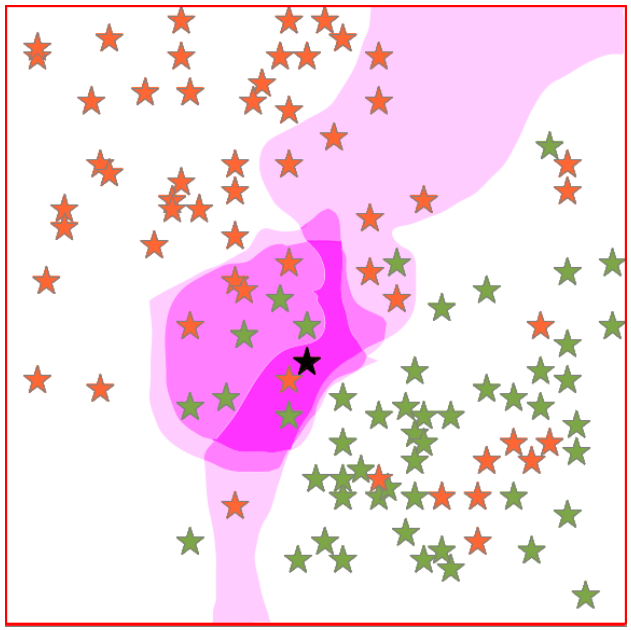
$P_t(c|x)$ is class posterior in leaf node n_t



Fuse before estimation:

$$D_{L_x} = \bigcup_{t=1}^{T} D_{n_t}$$

(Multi-set)



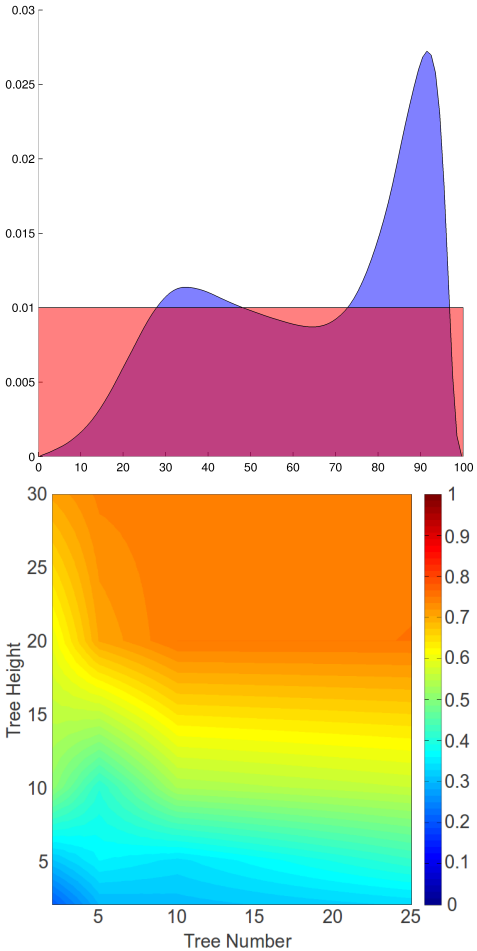
Weighted fusion.

$$P(c|x) = \sum_{t=1}^{T} w_t P_t(c|x)$$

Random Forests - Split point selection

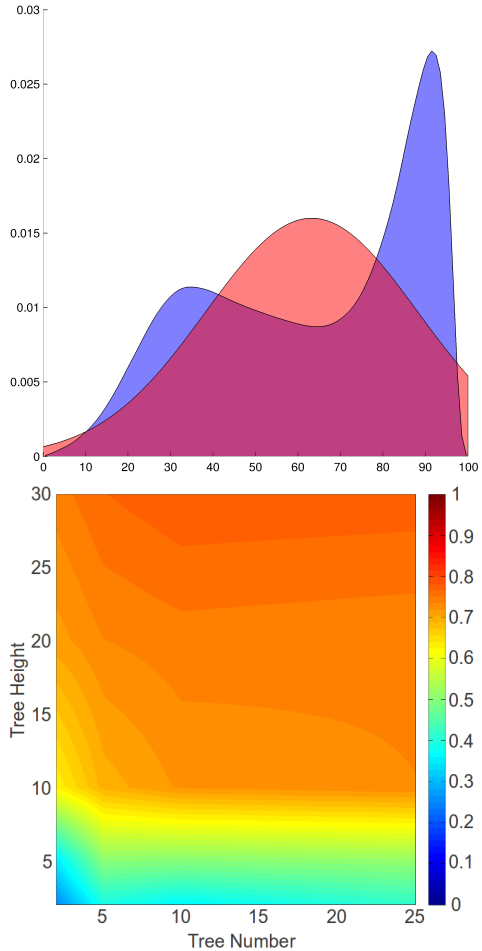
Uniform sampled

$\theta \sim U(\min(\hat{D}), \max(\hat{D}))$



Gaussian sampled

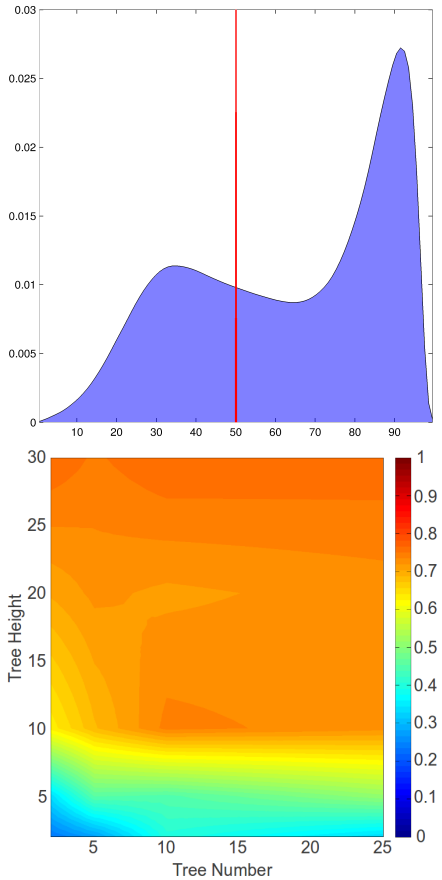
$\theta \sim N(\mu(\hat{D}), \sigma(\hat{D}))$



Random Forests - Split point selection

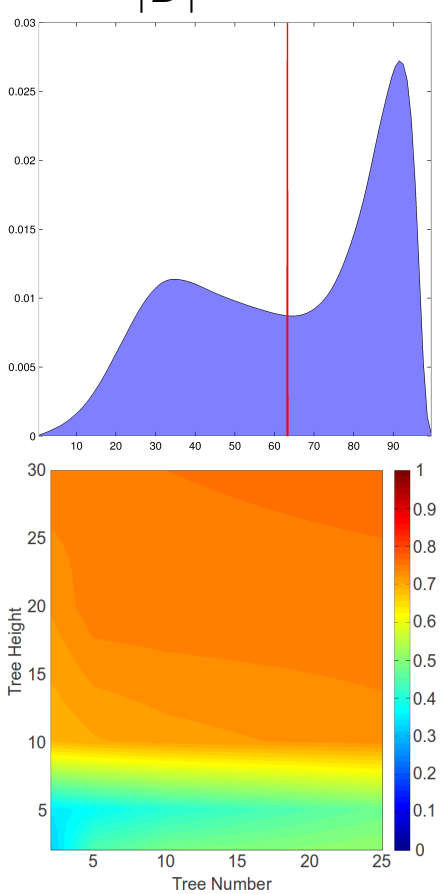
Interval center

$$\theta = \frac{min(\hat{D}) + max(\hat{D})}{2}$$



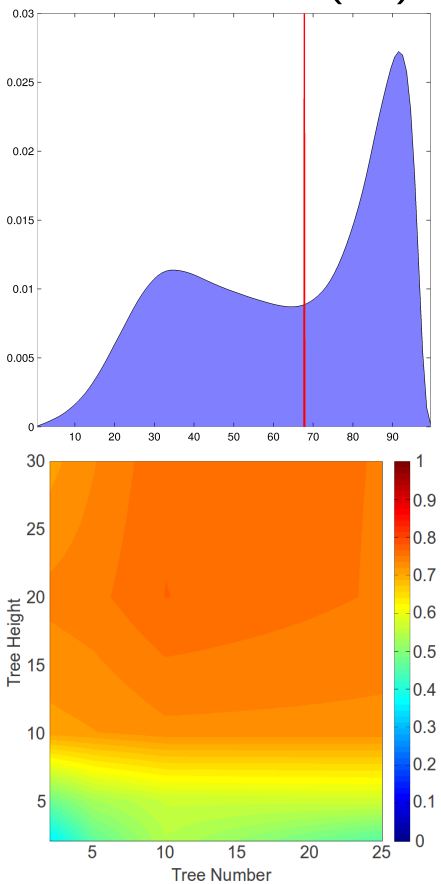
Mean value

$$\theta = \frac{1}{|\hat{D}|} \sum_{x \in \hat{D}} \hat{x}_i$$



Median value

$$\theta = median(\hat{D})$$

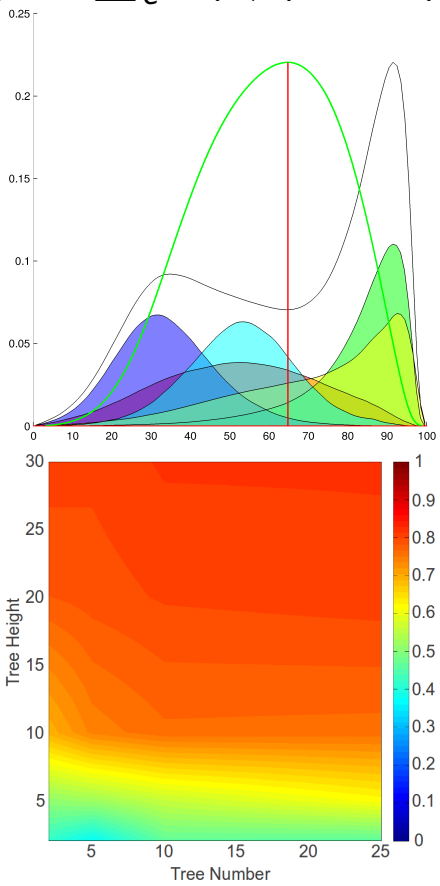


Random Forests - Split point selection

Max. drop of impurity $\theta = \arg\min_{\hat{\theta}} [I(n) - P_L I(n_L) - P_R I(n_R)]$

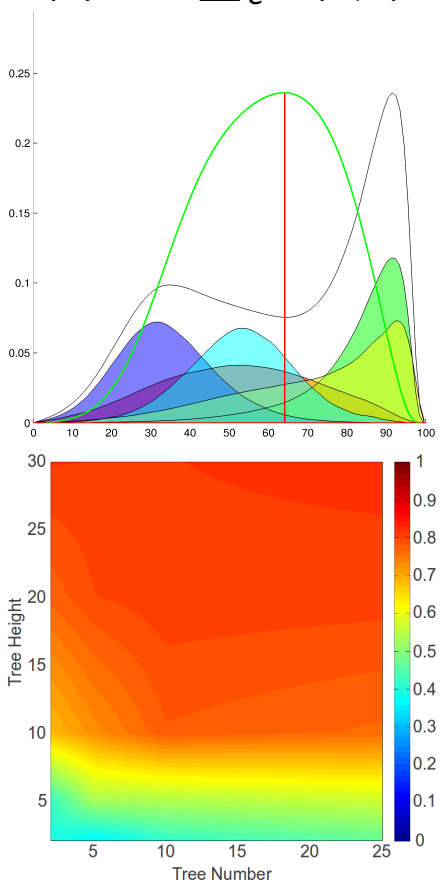
Entropy

$I(n) = -\sum_c P(c|n) \cdot \log P(c|n)$



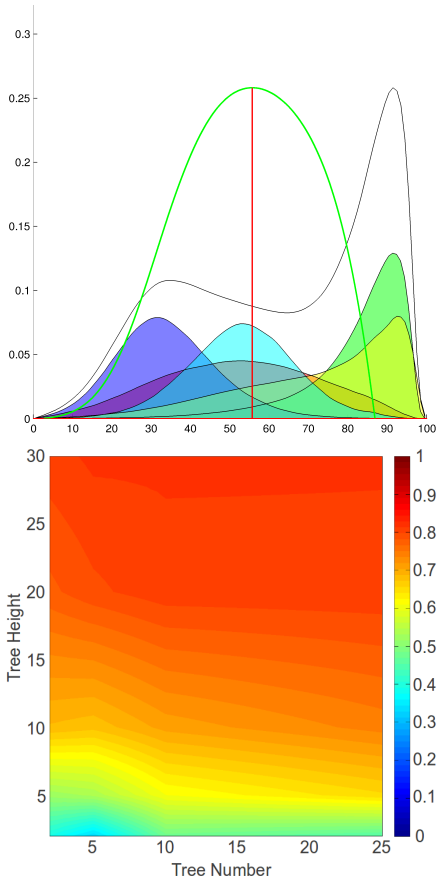
Gini

$I(n) = 1 - \sum_c P(c|n)^2$



Misclassification

$I(n) = 1 - \max_c P(c|n)$



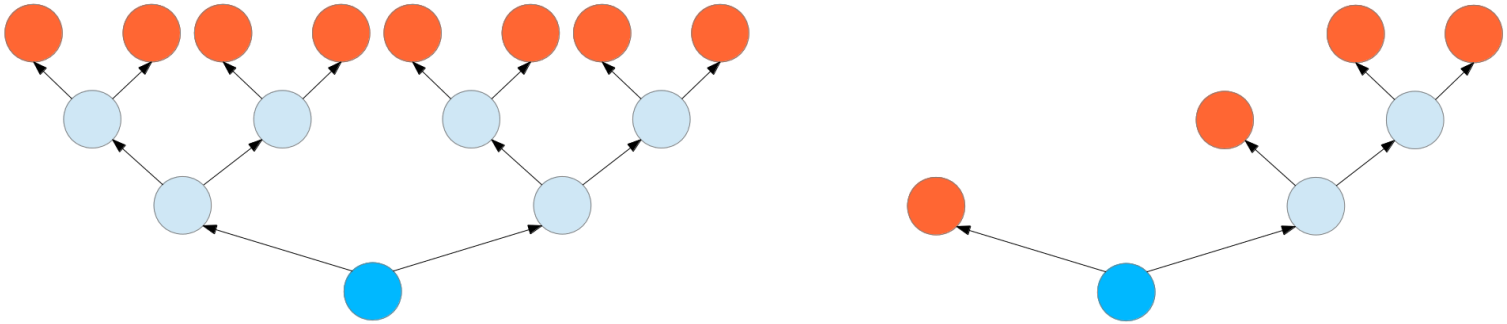
Random Forests - Node optimization

- Generate m split candidates
  $\rightarrow$ "Traditionally": $m = \sqrt{d}$, where d is data dimension
  $\rightarrow$ "Modern" approaches: $m \approx 10^5$
  $\rightarrow$ Usually even $m = 2$ leads to performance increase
  $\rightarrow$ Trade-off between high performance and high correlation
- Select best split, reject all others
- Measure optimality of a split
  - Classification: "Purity" of child nodes (e.g. Gini, entropy, etc.)
  - Regression: e.g. variance
  - In general: How much better is the estimation of the child nodes (as a weighted average) than parent nodes?

Random Forests - Structured prediction

- Image data is structured
- Exploited already during structured projection within node tests
- Target variable can be structured as well
  $\rightarrow$ Offset vectors, label patch
- Enriched spatial estimate for image labeling
- Disadvantage: Increased memory footprint

Random Forests - Interpretation



- Is maximum tree height reached?
- How balanced is a tree? $\frac{\#\text{nodes}}{2^{H+1}-1}$
- How large is largest leaf? $1 - \frac{\max_{n_t} |D_{n_t}|}{|D|}$
- How pure is largest leaf? $I(n^*)$ with $n^* = \arg\max_{n_t} |D_{n_t}|$
- Out-of-bag estimate for generalization error
- Out-of-bag estimate for correlation

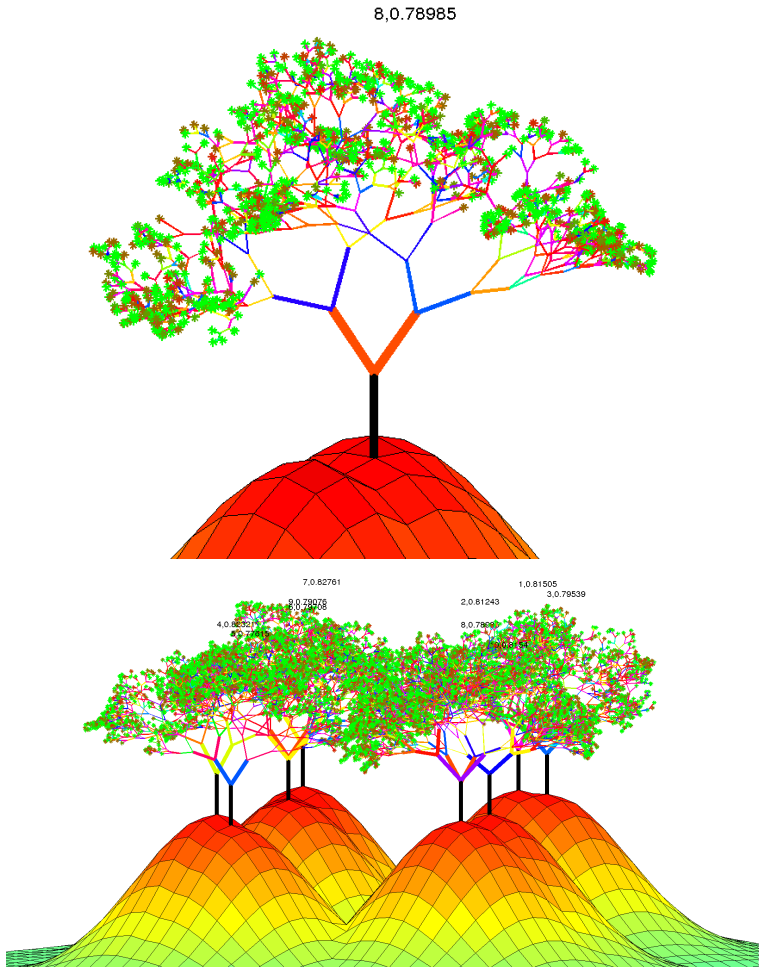
Random Forests - Visualization

Important Characteristics of a Random Forests

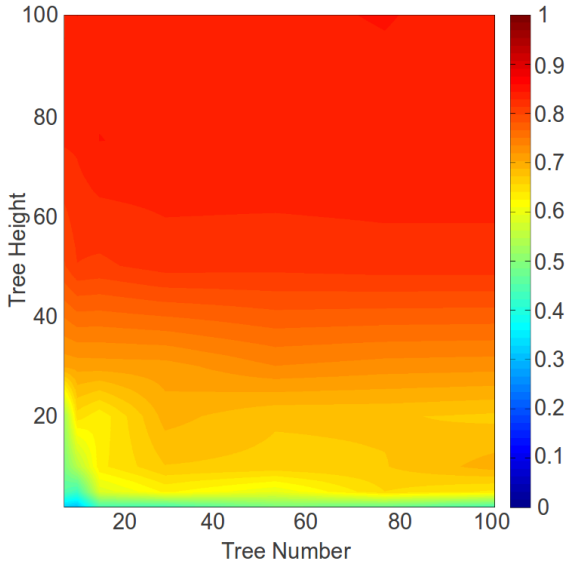
- Forest Level
  - Strength of the whole forest
    → e.g. classification accuracy
  - Correlation between trees
    → e.g. correlation of classification maps
- Tree Level
  - Strength of the individual tree
    → e.g. based on out-of-bag error
  - Structural layout of individual trees
    → e.g. balanced vs. degenerated chain
- Node Level
  - Node features
    → e.g. size, split dimension, drop of impurity, leaf impurity, etc.

Random Forests - Visualization

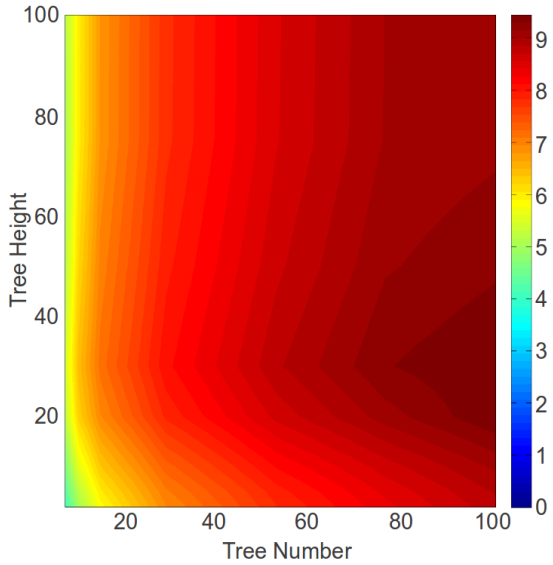
- Branch
  - Color:
    (e.g.) split dimension
  - Thickness:
    Number of samples
  - Length:
    $l_{h+1}^{L/R} = l_h \cdot \kappa_2 \cdot ((f_{\max} - f_{\min}) + f_{\min})$
  - Orientation:
    $(\alpha, \beta)_{h+1}^{L/R} = (\alpha, \beta)_h \pm (30°, \kappa_1 \cdot 45°)$
- Leaf
  - Color: Leaf impurity
  - Size: Leaf size
- 2D position
  Based on pairwise correlation



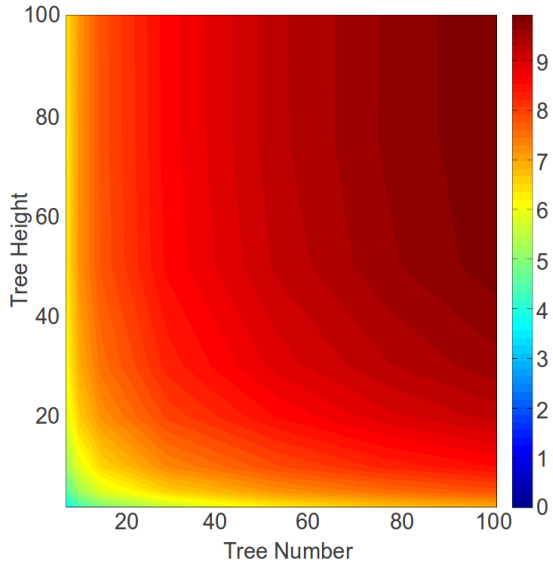
Random Forests - Best Practice



Accuracy



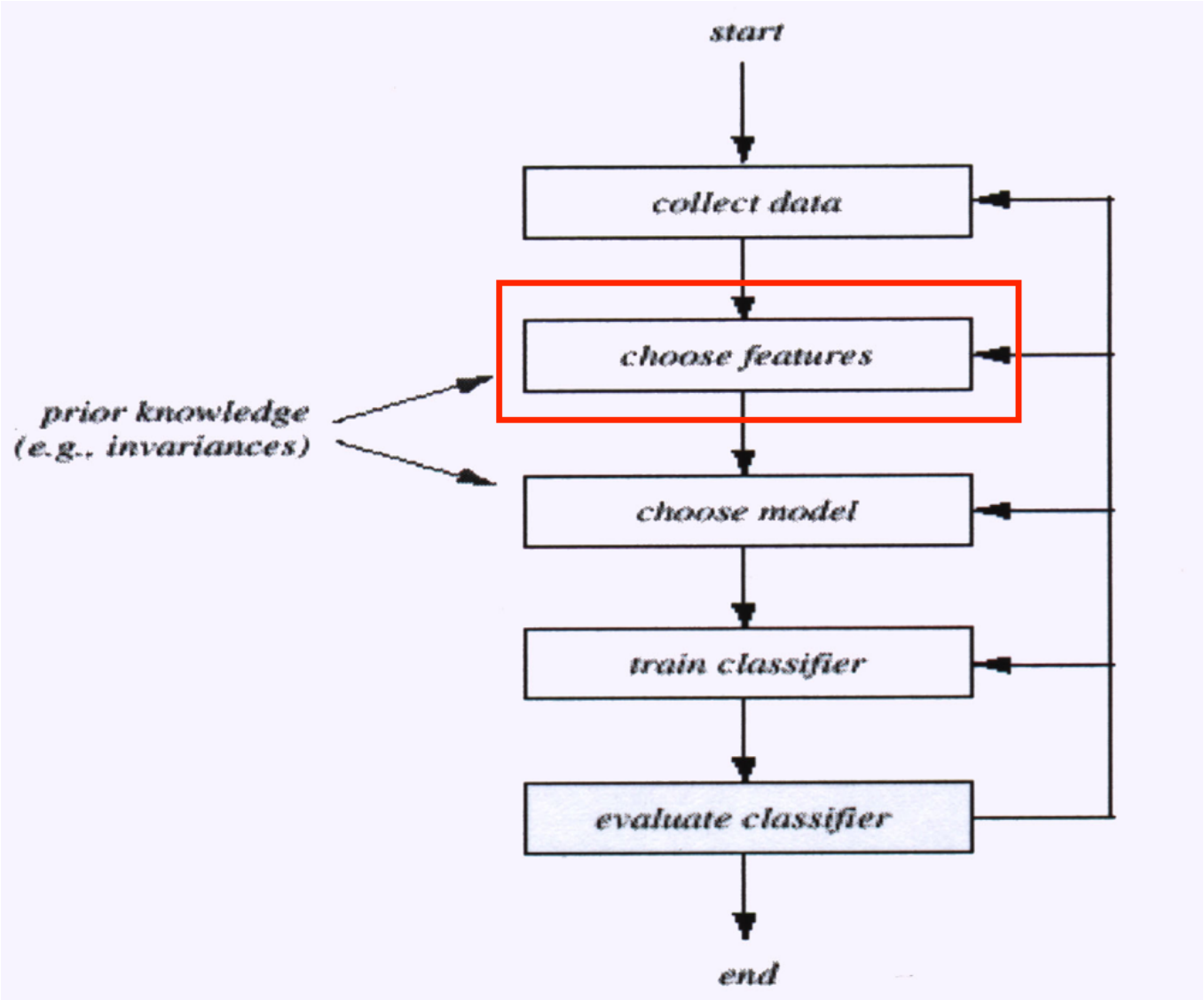
log-Training time



log-Prediction time

- Accuracy usually grows faster with tree height than tree number
- But: Tree height limited by amount of training data
- Use projections / features that are as diverse as possible
- Use simple split point definitions in combination with node optimization (i.e. selection)
- Check tree properties / visualize

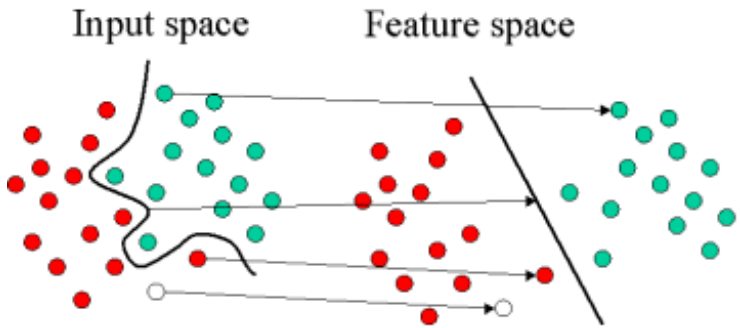
Why do we need feature extraction?



Motivation

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

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

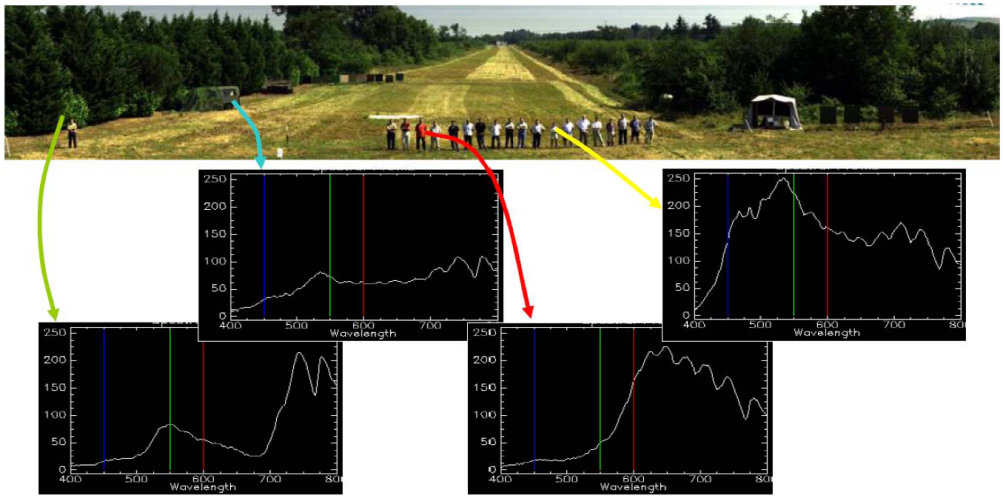


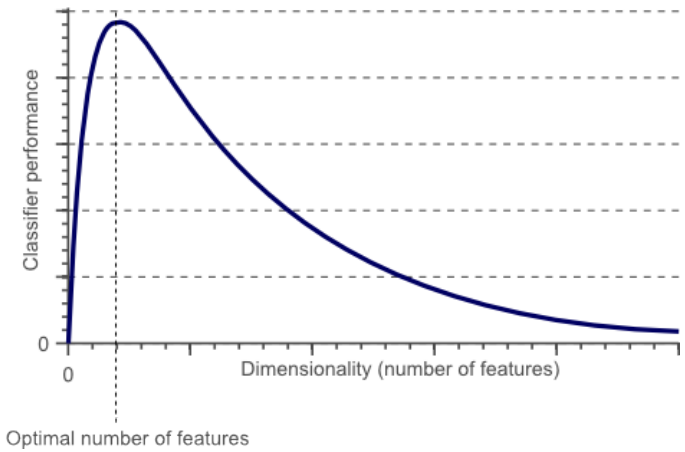
Objectives:

- 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





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

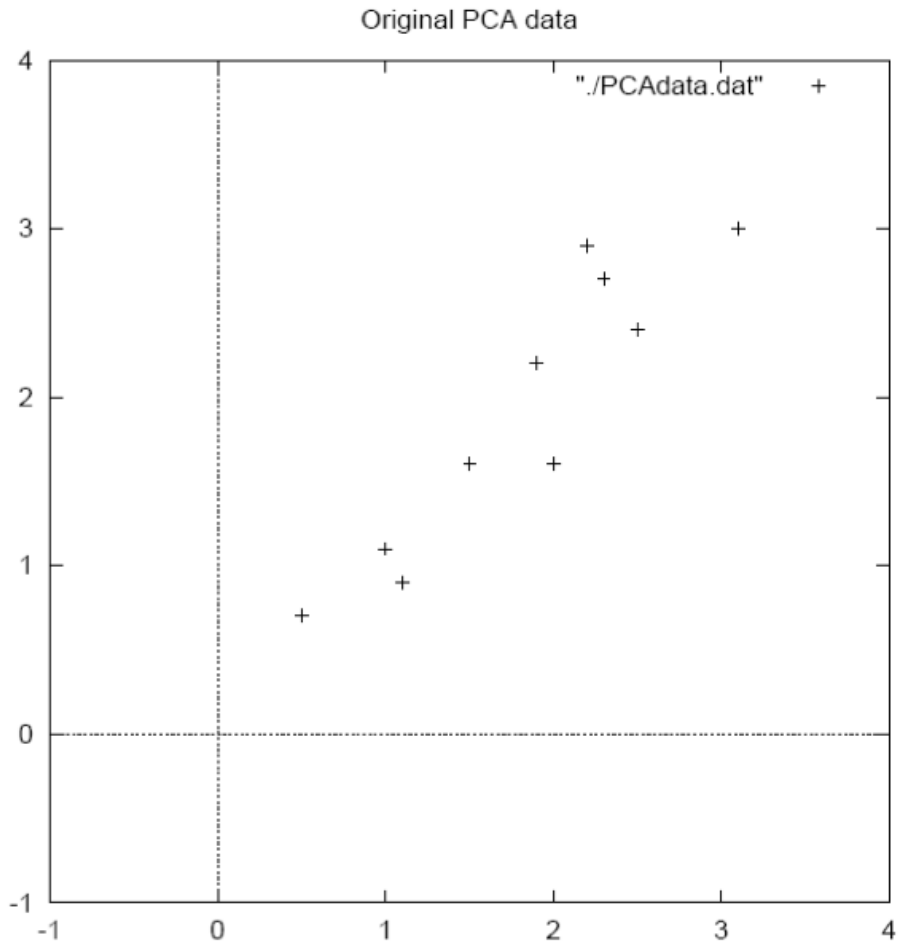
- **Goal**: represent data is a space that best describes the variation in a sum-squared error sense
- Projection onto eigenvectors that correspond to the first few largest eigenvalues of the covariance matrix
  - d-dimensional data are represented in a lower-dimensional space
  - Reduces the space and 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

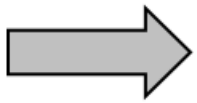


Principal component analysis

- **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 =

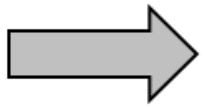
x	y
.69	.49
-1.31	-1.21
.39	.99
.09	.29
1.29	1.09
.49	.79
.19	-.31
-.81	-.81
-.31	-.31
-.71	-1.01

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



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

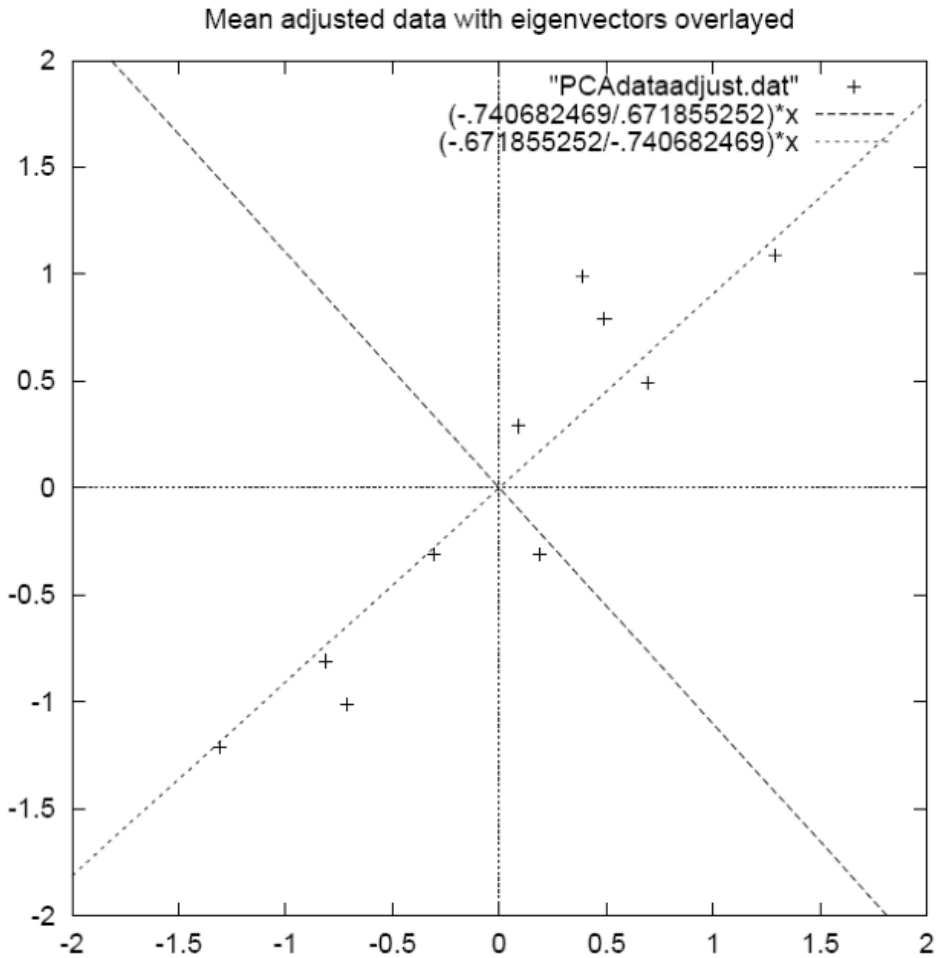
Principal component analysis

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

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

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

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



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

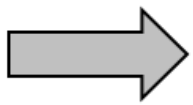
- **Step 5**: Choose components and form a feature vector
  - Order eigenvectors by eigenvalues
    - This gives the components in order of significance
    - You can decide to ignore the components of lesser significance $\Rightarrow$ final data will have less dimensions ($p < d$)
  - Form a feature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

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

- 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

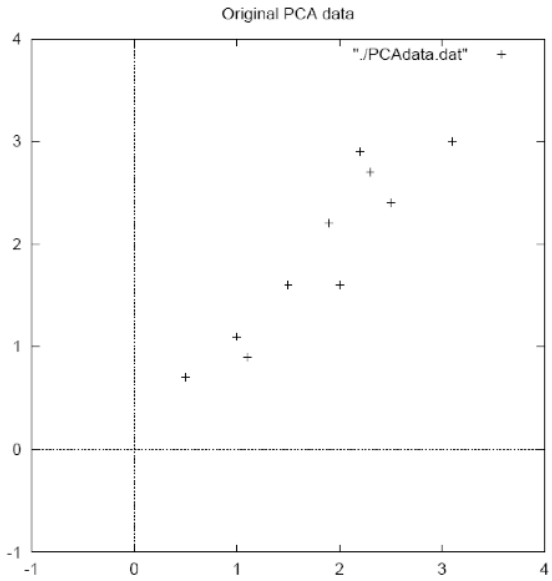
Principal component analysis

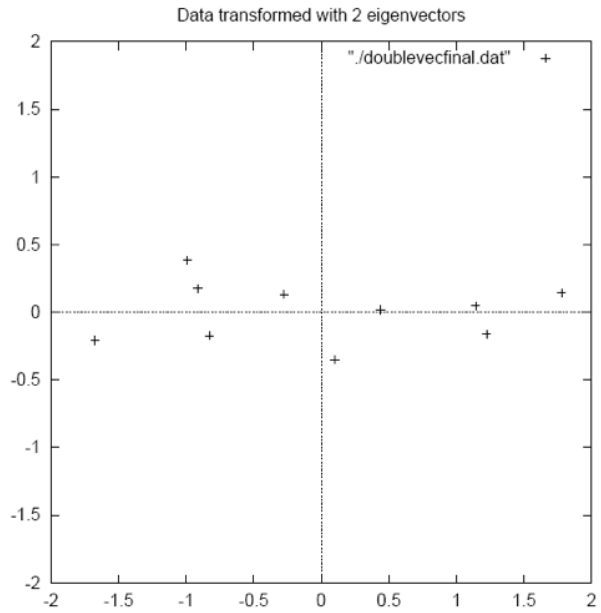
Data =

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



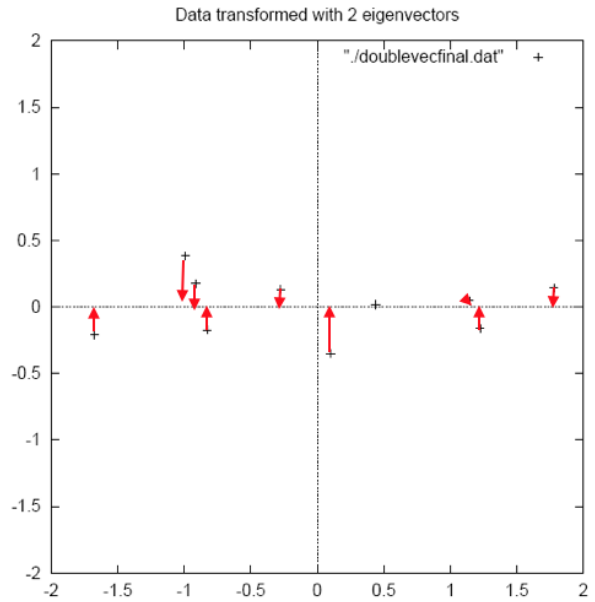


Principal component analysis (PCA)

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

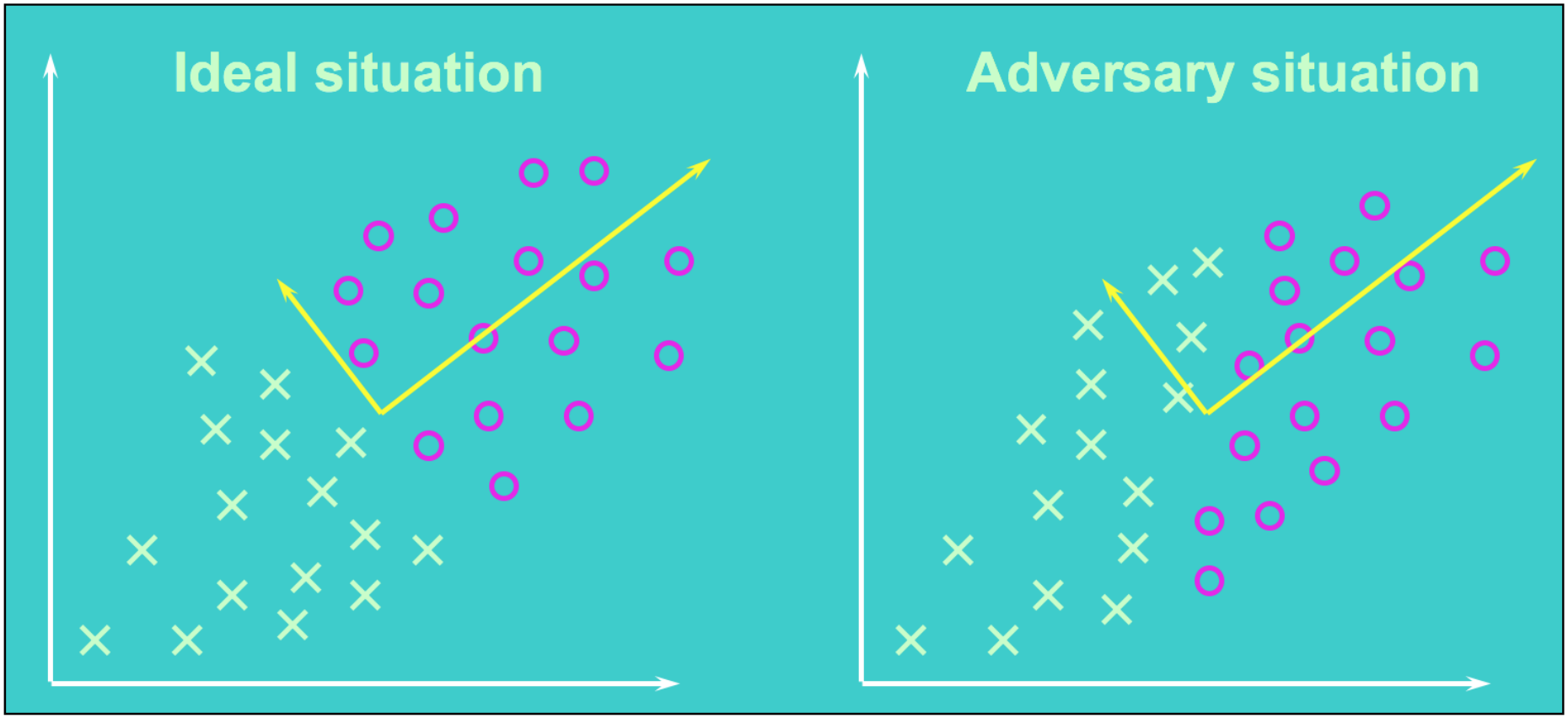
Transformed Data=

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

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

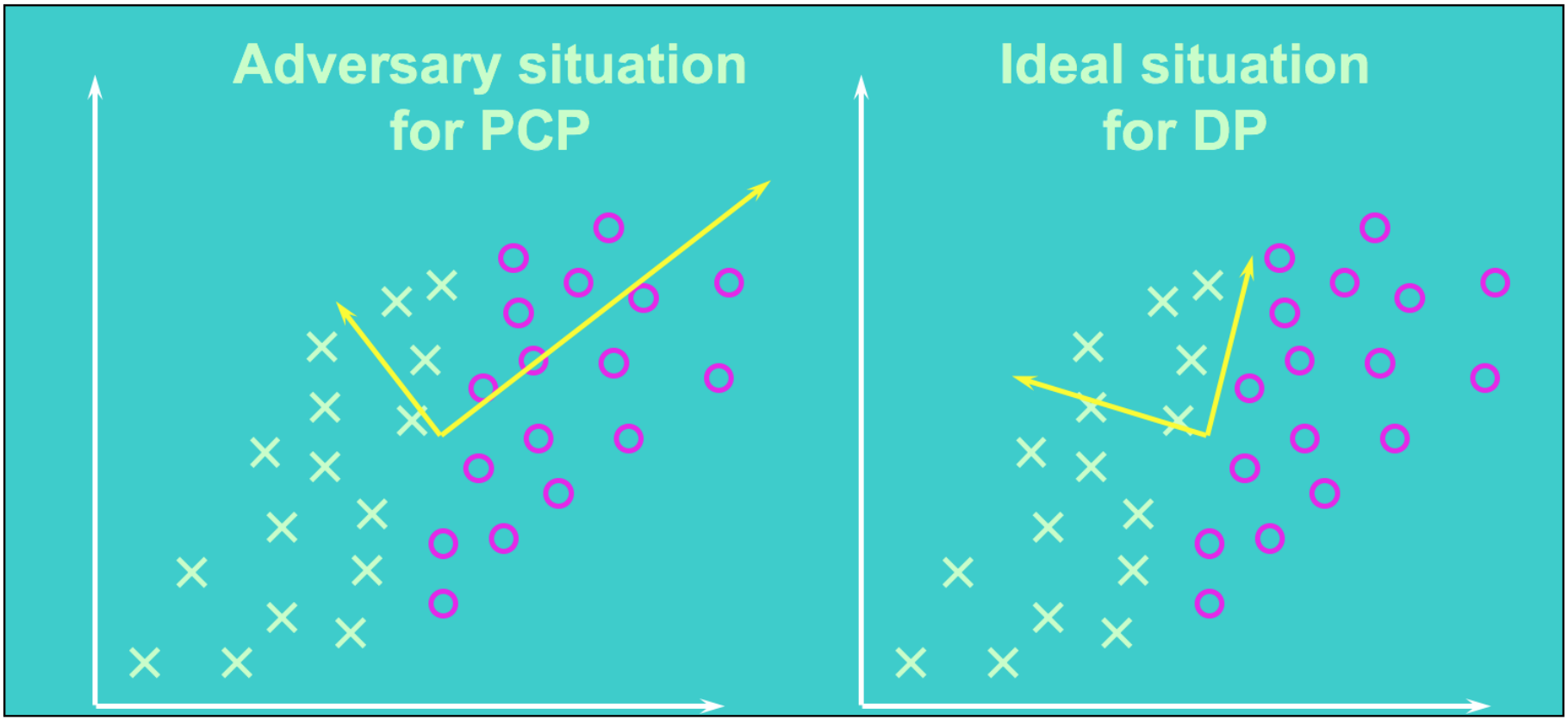


Discriminant analysis

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

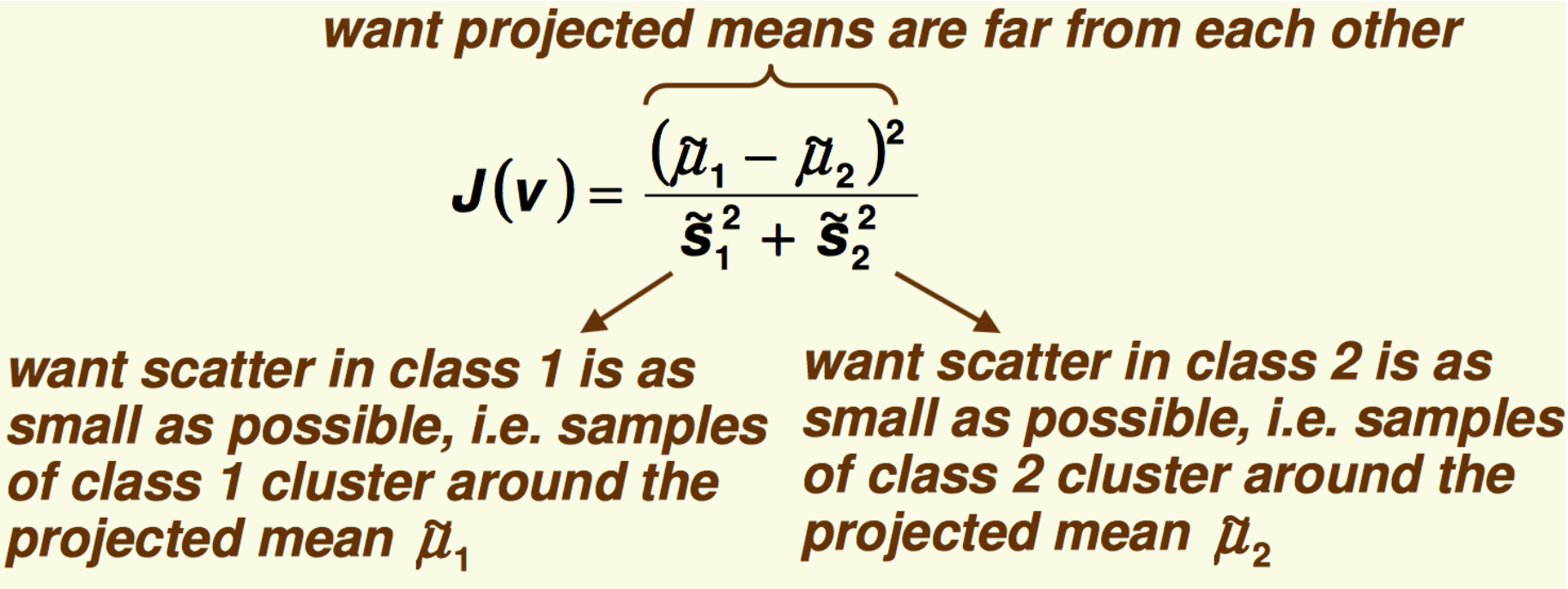
Discriminant analysis

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



Discriminant analysis

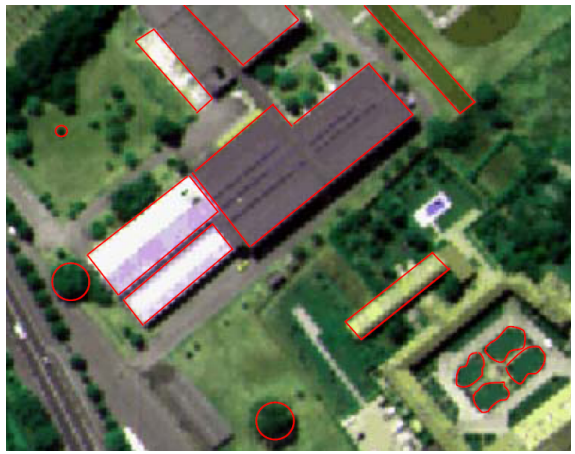
- Theory of Fisher linear discriminant: `http://www.csd.uwo.ca/~olga/Courses//CS434a_541a//Lecture8.pdf`
- Project on line in the direction v which maximizes:



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

How to include spatial information for image classification?

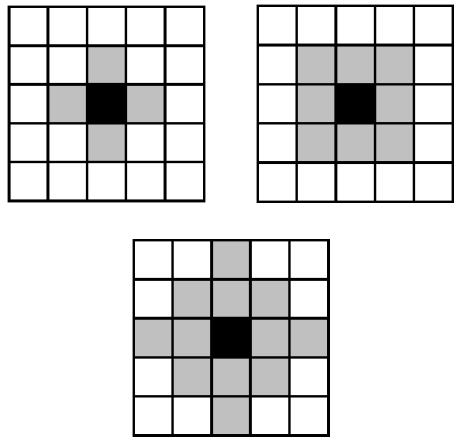
- By simply looking at a grey pixel, we cannot say if it belongs to a *building* or a *road*
- We guess a category by considering spatial/contextual information
- How can a classifier consider this rich source of information?



Approaches to extract spatial info

1. Closest fixed neighborhoods
   - Markov Random Field [Pony00, Jackson02, Farag05]
   - Contextual features [Camps-Valls06]
     - Spectral content +
     - Spatial content (e.g. mean or standard deviation per spectral band)

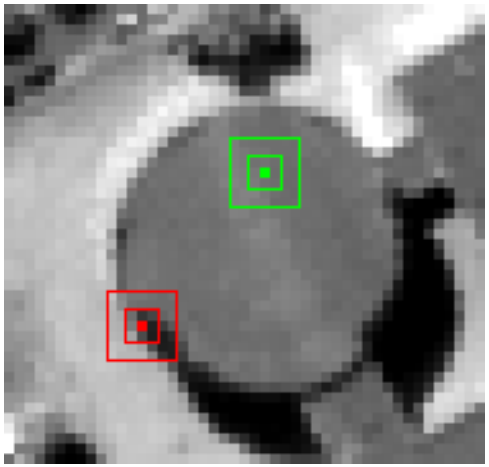
   + Simplicity
   − Imprecision at the border of regions



Approaches to extract spatial info

1. Closest fixed neighborhoods
   - Markov Random Field [Pony00, Jackson02, Farag05]
   - Contextual features [Camps-Valls06]
     - Spectral content +
     - Spatial content (e.g. mean or standard deviation per spectral band)

   \+ Simplicity
   − Imprecision at the border of regions

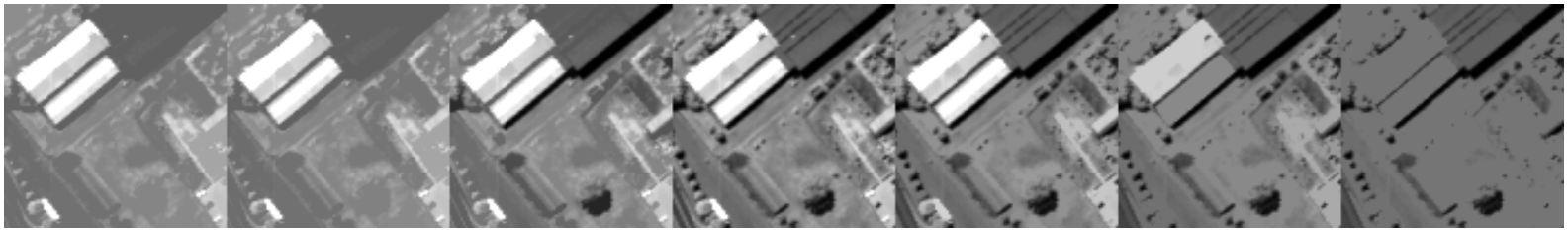


Approaches to extract spatial info

2. Morphological and area filtering

   - Morphological profiles [Pesaresi01, Dell'Acqua04, Benediktsson05]
   - Self-complementary area filtering [Fauvel07]
   - Attribute profiles [Ghamisi15, Cavallaro17]

   \+ Neighborhoods are adapted to the structures
   \+ Non-linear operators $\Rightarrow$ do not blur the edges as convolutions do
   – Neighborhoods are scale dependent

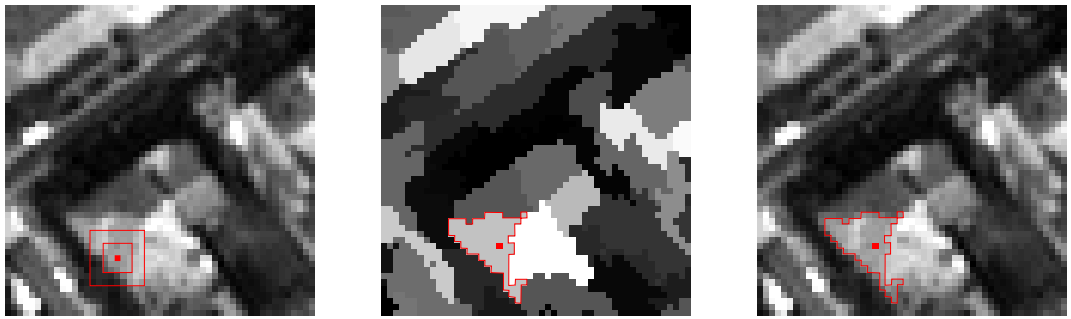


Course on mathematical morphology:
`http://www-sop.inria.fr/members/Yuliya.Tarabalka/teaching.htm`

Approaches to extract spatial info

2. Morphological and area filtering
   - Morphological profiles [Pesaresi01, Dell'Acqua04, Benediktsson05]
   - Self-complementary area filtering [Fauvel07]
   - Attribute profiles [Ghamisi15, Cavallaro17]

\+ Neighborhoods are adapted to the structures
\+ Non-linear operators $\Rightarrow$ do not blur the edges as convolutions do
– Neighborhoods are scale dependent



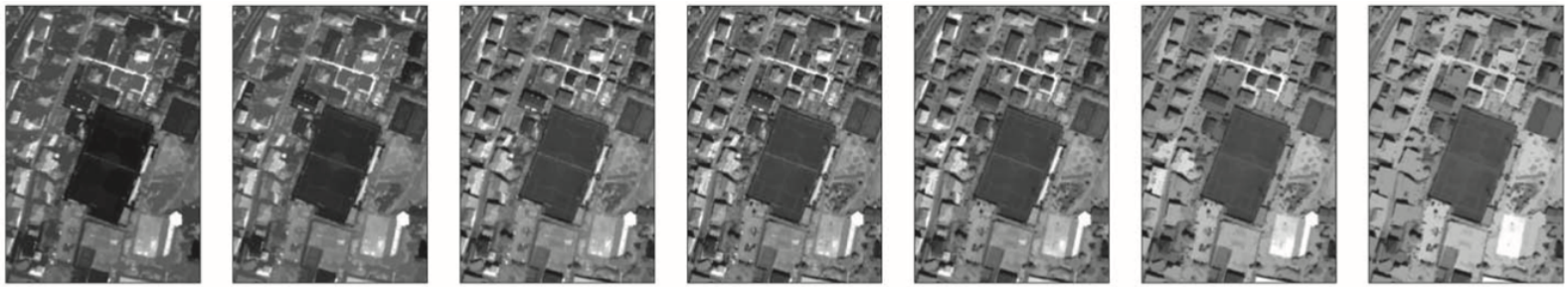
Course on mathematical morphology:
`http://www-sop.inria.fr/members/Yuliya.Tarabalka/teaching.htm`

Approaches to extract spatial info

2. Morphological and area filtering

   - Morphological profiles [Pesaresi01, Dell'Acqua04, Benediktsson05]
   - Self-complementary area filtering [Fauvel07]
   - Attribute profiles [Ghamisi15, Cavallaro17]

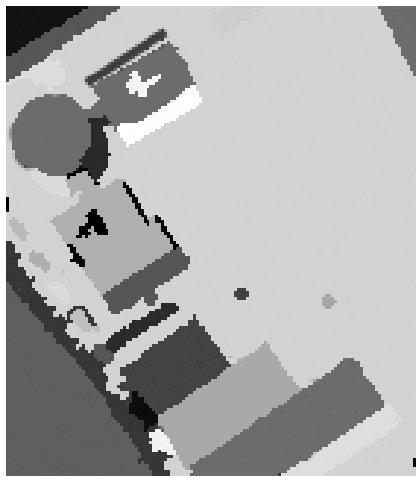
   \+ Neighborhoods are adapted to the structures
   \+ Non-linear operators $\Rightarrow$ do not blur the edges as convolutions do
   – Neighborhoods are scale dependent



Course on mathematical morphology:
`http://www-sop.inria.fr/members/Yuliya.Tarabalka/teaching.htm`

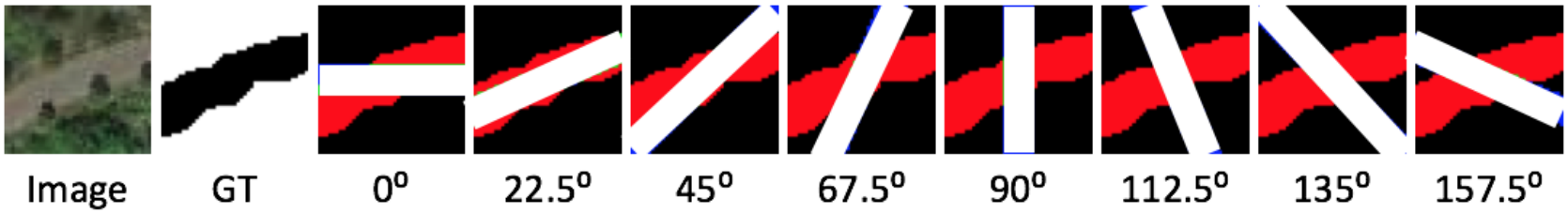
Approaches to extract spatial info

3. Superpixels derived from segmentation
   - Extraction and Classification of Homogeneous Objects [Kettig76]
   - ...
   - Multiscale segmentation, then features are derived from the regions [Linden07, Huang09]
     + Flexible
     – Computationally demanding
     – Difficult to scale/parallelize



Approaches to extract spatial info

4. Features handcrafted for a particular application
   - Example 1: Line templates with different orientations for road detection [Jeong15]
   - Example 2: Rectangular templates for building detection [Garcin01]
     + Can model complex shape
     – Lack of genericity
     – Computationally demanding



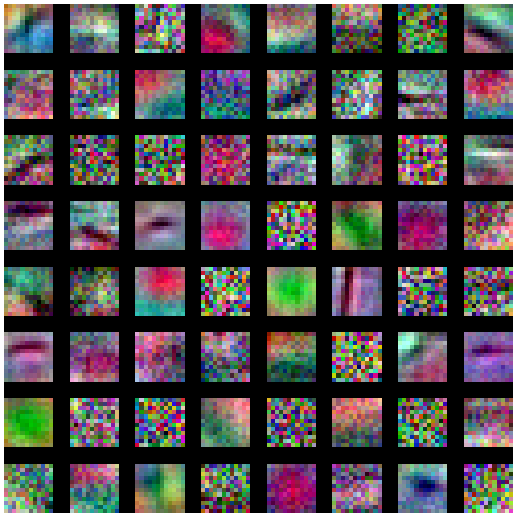
Approaches to extract spatial info

4. Features handcrafted for a particular application
   - Example 1: Line templates with different orientations for road detection [Jeong15]
   - Example 2: Rectangular templates for building detection [Garcin01]
     + Can model complex shape
     – Lack of genericity
     – Computationally demanding

Modern trend & Conclusions

Deep learning:

- Automatically learn features if a lot of training data are available



Advice:

- If for the considered application it is easy to hand-craft class-separable features, no need to learn them
- If it is not easy to discriminate between categories, learning features often helps