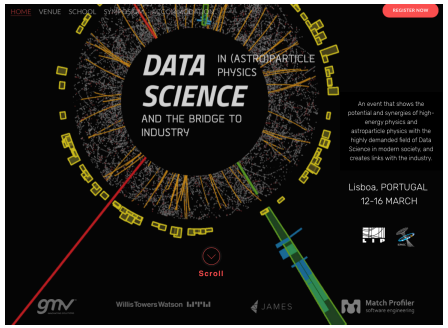
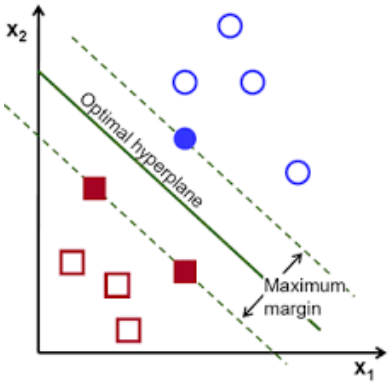
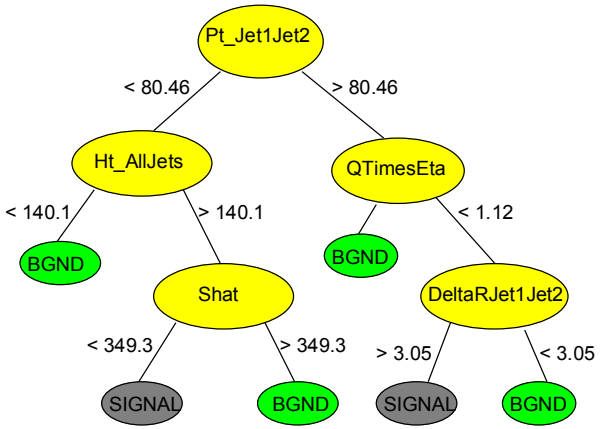


Machine Learning (Lecture 2)





Lorenzo Moneta
CERN - EP-SFT
Lorenzo.Moneta@cern.ch

LIP Data Science School / 12-14 March 2018

Outline

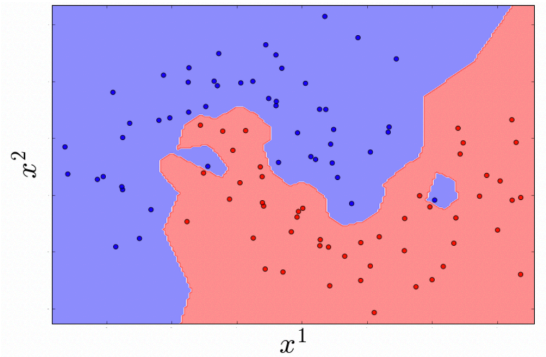
Lecture 1 (yesterday)

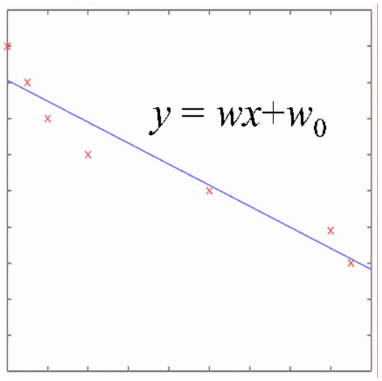
- Introduction to Machine Learning
- Supervised Learning
- Linear Models
  - Regression
  - Classification
- Hypothesis Tests and ROC curve
- Overfittig and Regularization
- Cross-Validation
- Machine Learning Software
  - Introduction to ROOT/TMVA

Mathematical Modeling

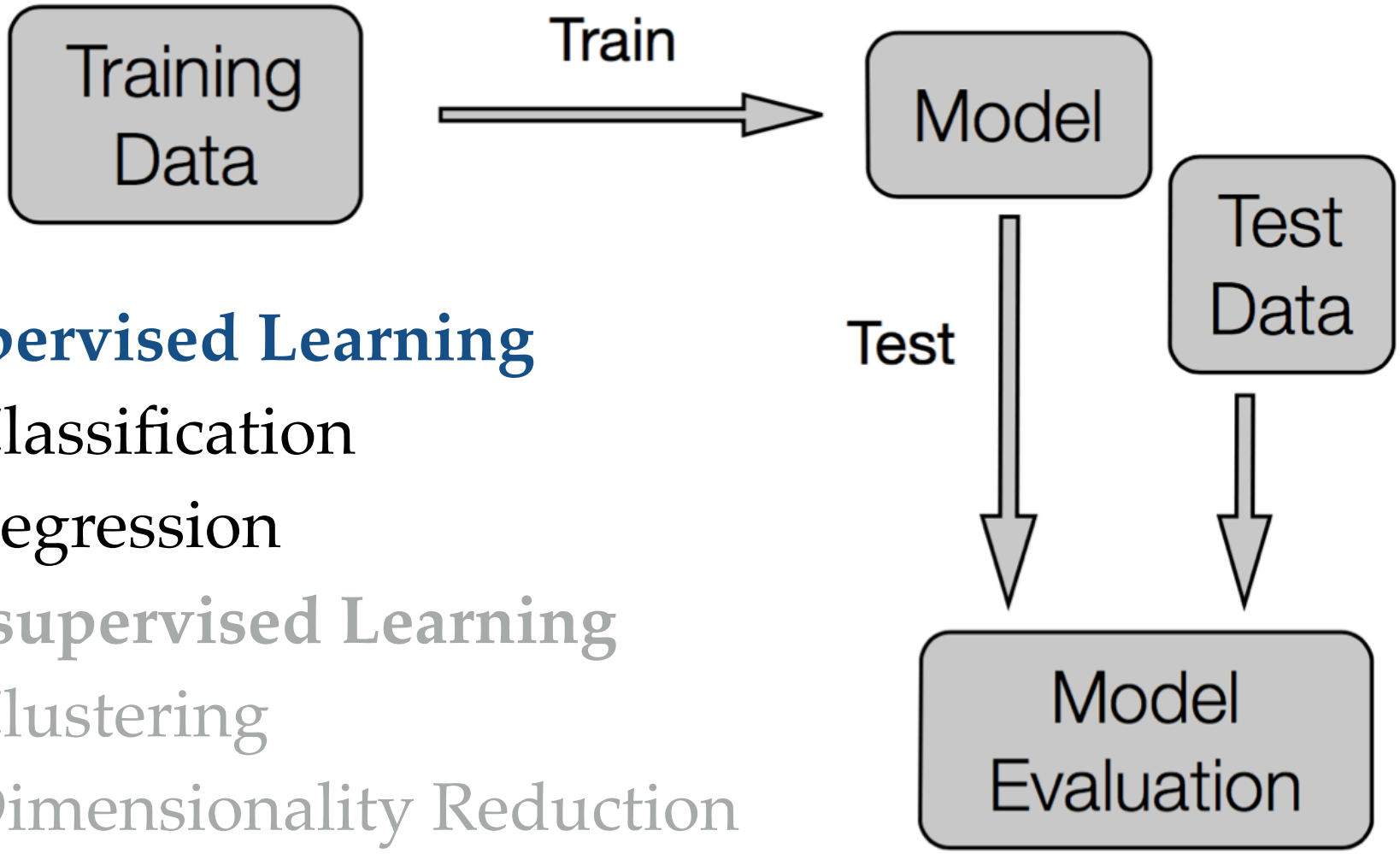
Key element is a mathematical model

- **Learning**
  - Estimate statistical model from the data
- **Prediction and Inference**
  - use the statistical model to make predictions on new data points and infer properties of system(s)





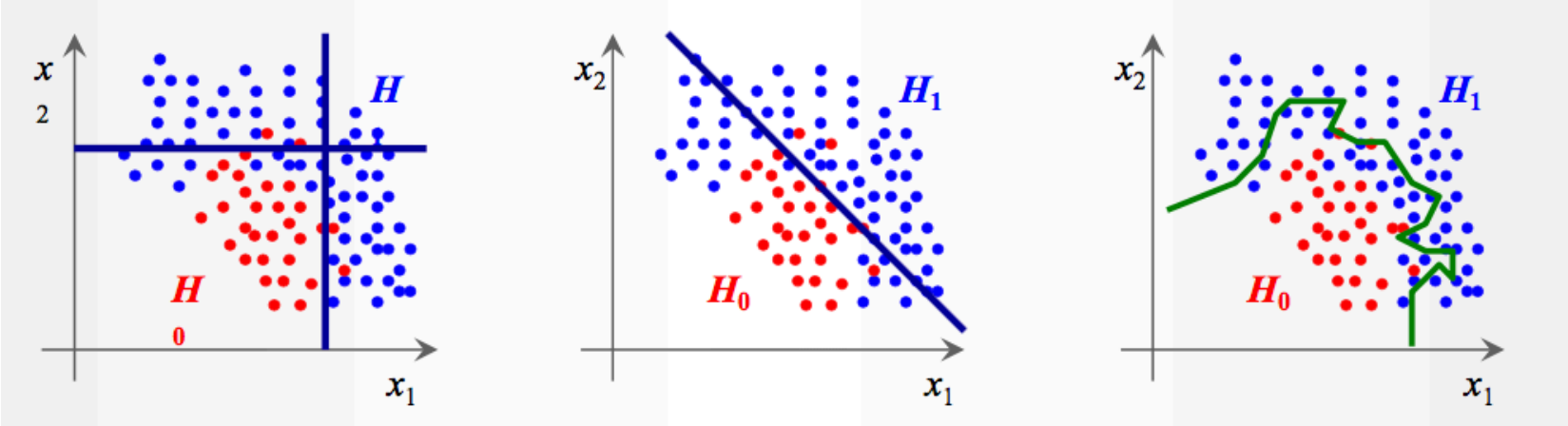
Learning



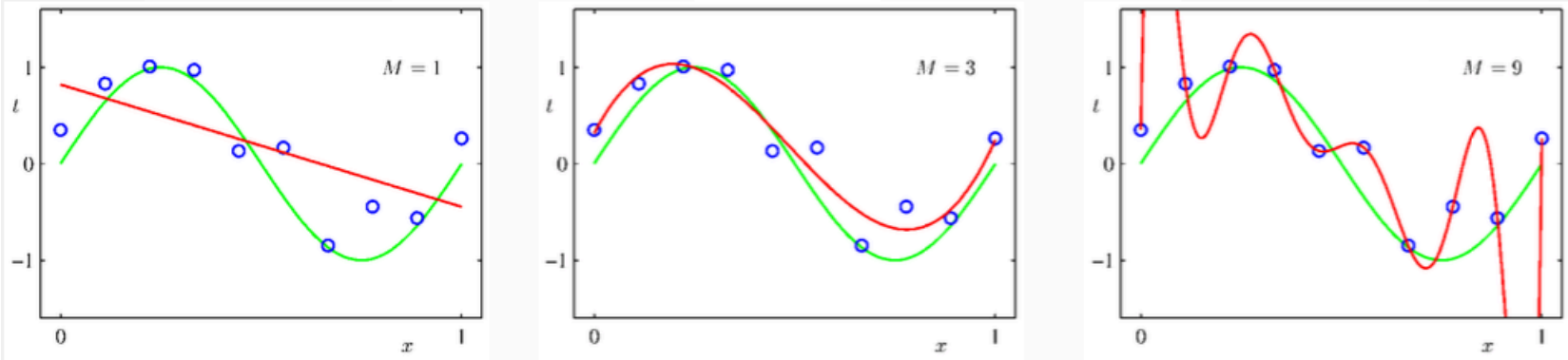
- **Supervised Learning**
  - Classification
  - Regression
- **Unsupervised Learning**
  - Clustering
  - Dimensionality Reduction
  - Anomaly Detection
- **Reinforcement Learning**

Classification and Regression Tasks

- **Classification** - How to find the best decision boundary ?



- **Regression** - How to determine the correct model ?



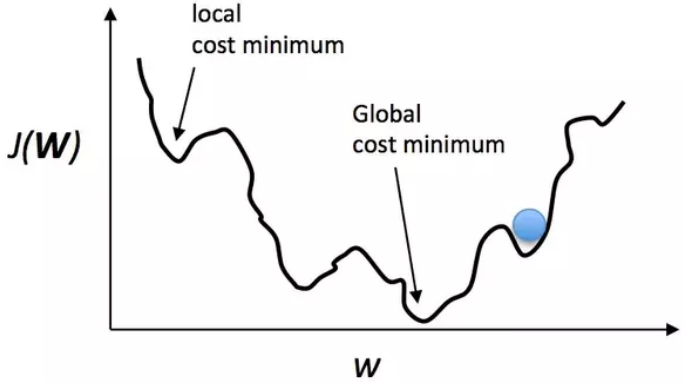
[E. v. Toerne]

Supervised Learning

How does it works ?

- Given the data { $x_i \in \boldsymbol{\mathcal{X}}$ } and targets { $y_i \in \boldsymbol{\mathcal{Y}}$ }
- choose a model $\boldsymbol{\mathcal{F}} = \{ f(\mathbf{x}; \mathbf{w}) \}$ and with optional constraint $\boldsymbol{\Omega}(\mathbf{w})$ mapping $y = f(\mathbf{x}; \mathbf{w})$
- Design a Loss function measuring the cost of choosing badly

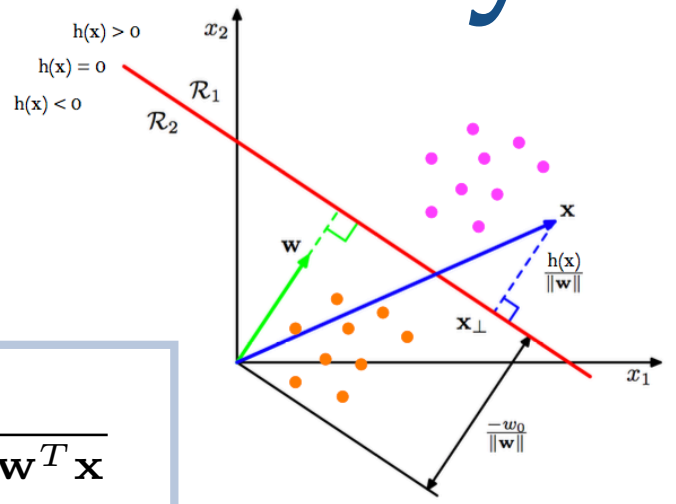
$$L(\mathbf{w}, \mathbf{x}) = \frac{1}{N} \sum_{i=1}^{N} L(y_i, f(\mathbf{x}_i, \mathbf{w}))$$



- Find best values of the parameters **w** that minimize the loss function L(**w, x**)
- Estimate the final performance on an independent data set

Linear Decision Boundary

$$h(\mathbf{x};\mathbf{w}) = \mathbf{w}^T\mathbf{x}$$



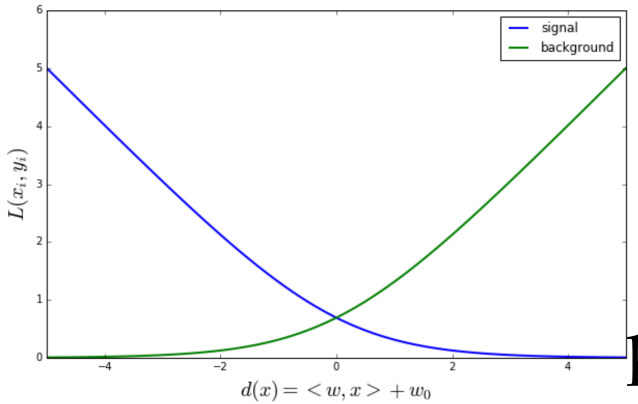
[Bishop]

$$p(y=1|\mathbf{x}) \equiv p_i = \frac{1}{1+e^{-h(\mathbf{x};\mathbf{w})}} = \frac{1}{1+e^{-\mathbf{w}^T\mathbf{x}}}$$

Binary cross entropy Loss function

$$L(\mathbf{w}) = -\sum_i (y_i \ln p_i + (1-y_i)\ln(1-p_i))$$

Loss



h(x;w)

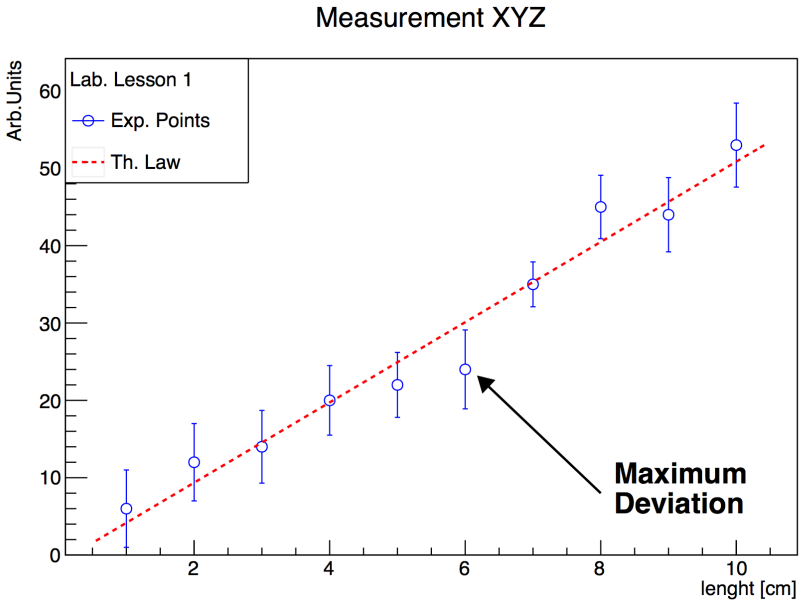
Least Square Regression

- Least Square Loss function

$$L(\mathbf{w}) = \frac{1}{2}\sum_{i=1}^{N}(y_i - f(\mathbf{x_i}; \mathbf{w}))^2$$

- Find minimum of L(w)

- Used also for parameter estimation i.e. Least square fit (χ^2 fit)



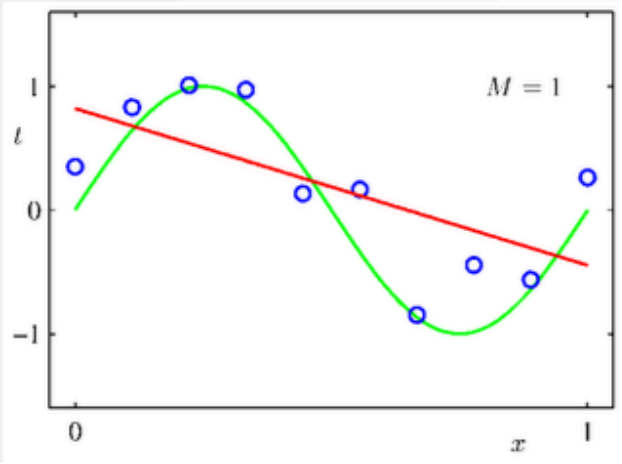
Regularization

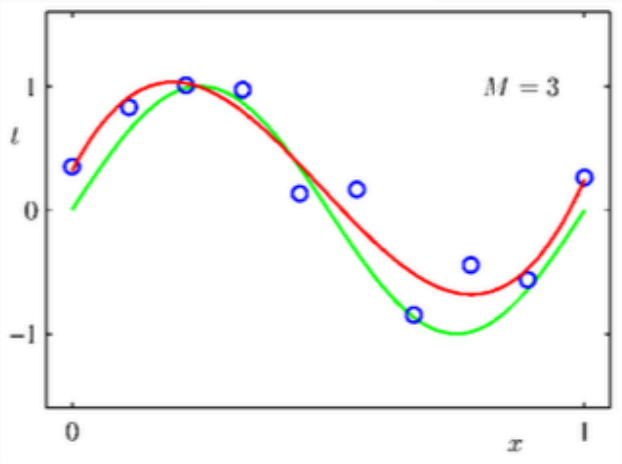
- Method to find optimal model is to add a parameter constraint in the loss function
  - aim to trade some bias to reduce variance
- Modify loss function (e.g. for linear regression):

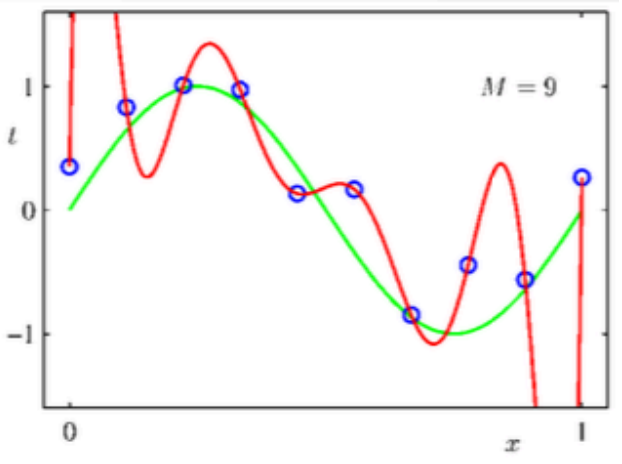
$$L(\mathbf{w}) = \frac{1}{2}\sum_{i=1}^{N}(y_i - f(\mathbf{x_i}; \mathbf{w}))^2 + \lambda\Omega(\mathbf{w})$$

  - **L2 norm** $\Omega(\mathbf{w}) = \|\mathbf{w}\|^2 = \sum_i w_i^2$
    - equivalent to Gaussian prior on the weights
  - **L1 norm** $\Omega(\mathbf{w}) = \|\mathbf{w}\| = \sum_i |w_i|$
    - equivalent to Laplace prior on the weights

What is the correct model ?







$f(x|\mathbf{w}) = w_0 + w_1 x$

$f(x|\mathbf{w}) = w_0 + w_1 x + w_2 x^2 + w_3 x^3$

$f(x|\mathbf{w}) = w_0 + w_1 x + ... + w_9 x^9$

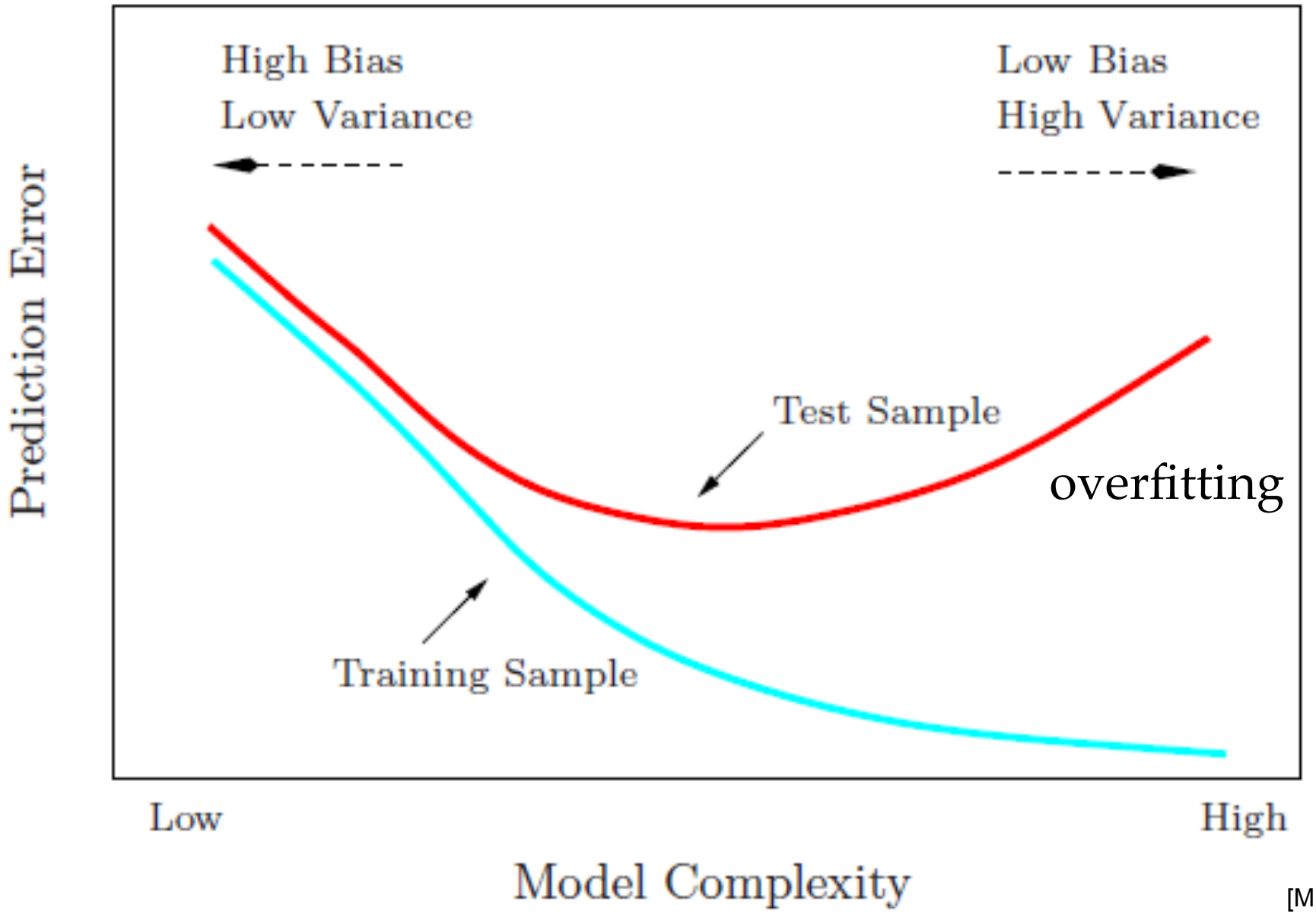
Under fitting
Large Bias

model does not
reproduce well the data

Over fitting
Large Variance

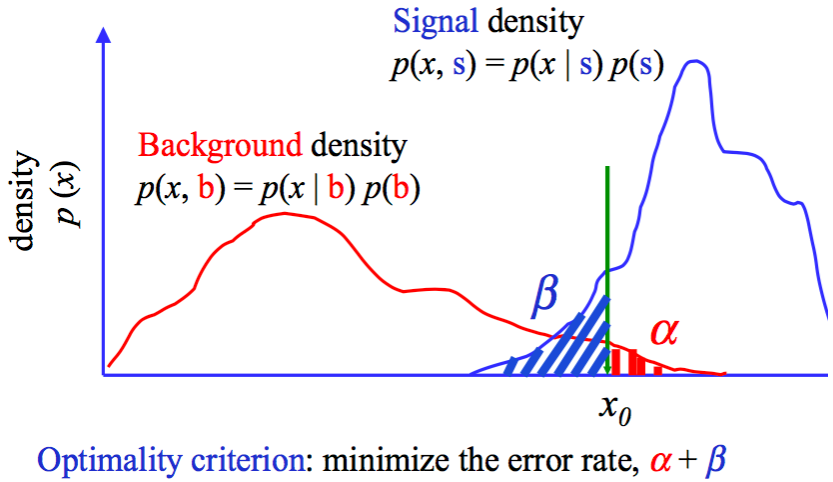
model reproduce the
training data too well

Hyper-Parameter Optimization



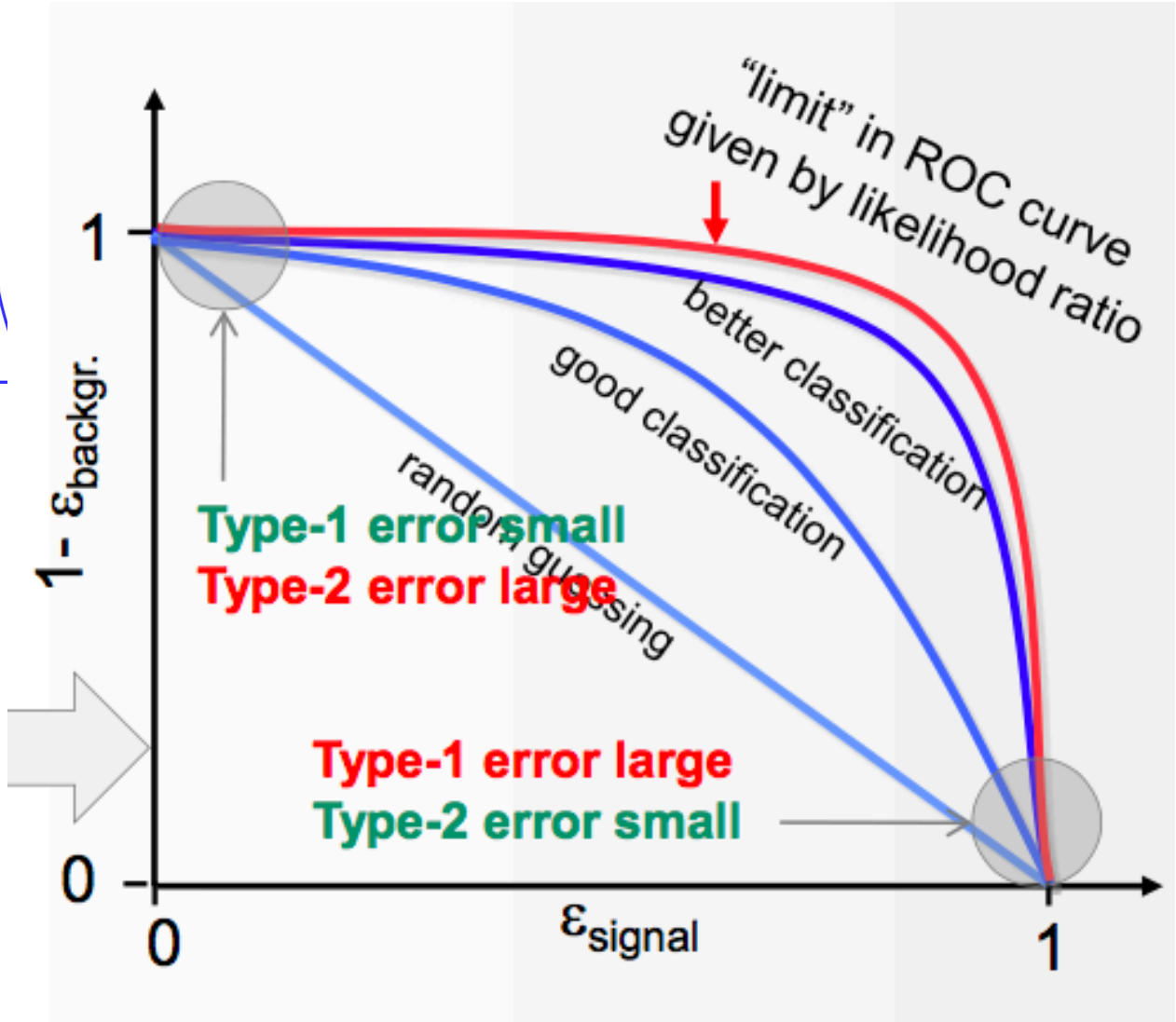
[M. Kagan]

ROC Curve



Optimality criterion: minimize the error rate, $\alpha + \beta$

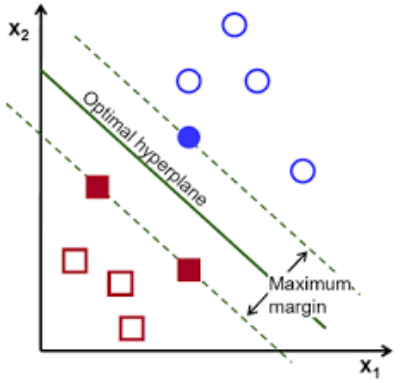
Optimal classifier maximises the area under the ROC curve **(AUC)**

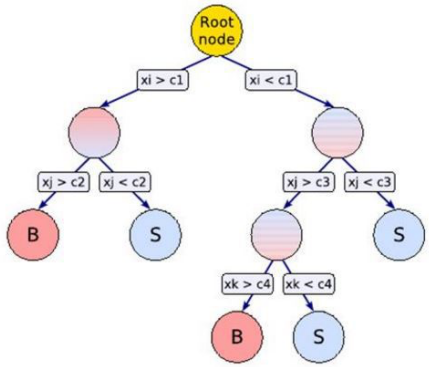


E. Toerne]

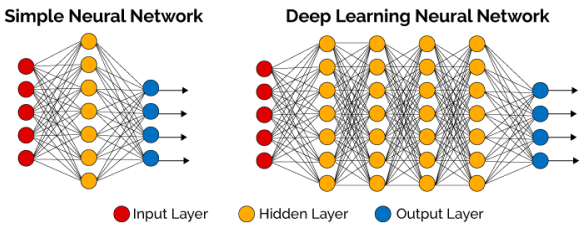
Outline for today

- Lecture 2 (today)
  - Fischer discriminant
  - **Support Vector Machine** (SVM)
  - **Decision Trees**



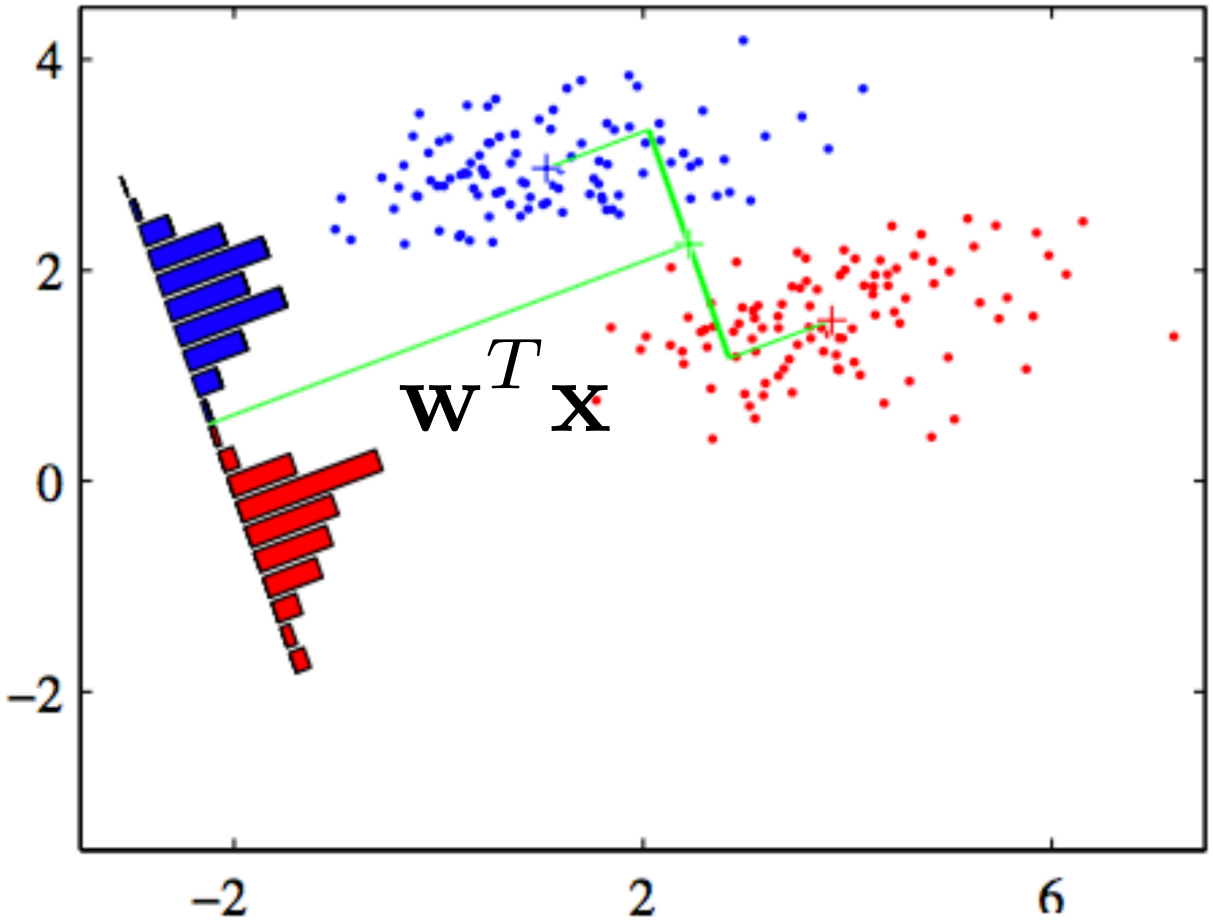


- Lecture 3 (tomorrow)
  - Introduction to Neural Networks
  - Deep Learning



Fischer Discriminant

- Find projection that maximises the separation between two classes



Fisher Linear Discriminant

- How to find the projection maximising the separation ?
  - Want means ($\mathbf{m}_i$) of two classes (C_i) to be as far apart as possible → large *between-class* variation

$$\mathbf{S}_B = (\mathbf{m}_2 - \mathbf{m}_1)^T(\mathbf{m}_2 - \mathbf{m}_1)$$

  - Want each class tightly clustered, as little overlap as possible → small *within-class* variation

$$\mathbf{S}_W = \sum_{i \in C_1} (\mathbf{x}_i - \mathbf{m}_1)^T(\mathbf{x}_i - \mathbf{m}_1) + \sum_{i \in C_2} (\mathbf{x}_i - \mathbf{m}_2)^T(\mathbf{x}_i - \mathbf{m}_2)$$

- Maximize Fisher criteria

$$\max \quad J(\mathbf{w}) = \frac{\mathbf{w}^T \mathbf{S}_B \mathbf{w}}{\mathbf{w}^T \mathbf{S}_W \mathbf{w}} \quad \rightarrow \quad \boxed{\mathbf{w} \propto \mathbf{S}_W^{-1}(\mathbf{m}_2 - \mathbf{m}_1)}$$

[M. Kagan]

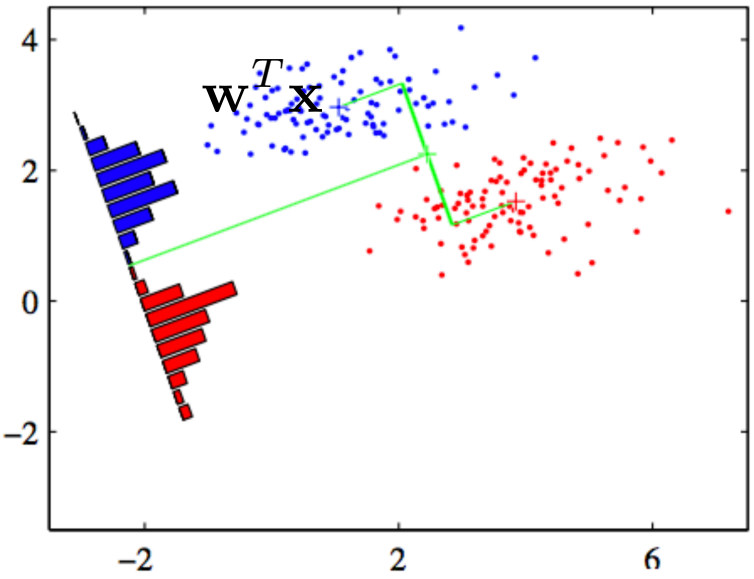
Fischer Discriminant

- **Easy to compute**
  - requires computing data covariances (within class matrix) for training data
- **Easy to understand** (visual representation)
- Optimal classifier for separating Gaussian data with same covariance but different mean
- Performance might be poor for complex data (when a non linear decision boundary is needed)

$$\mathbf{w} \propto \mathbf{S}_W^{-1}(\mathbf{m}_2 - \mathbf{m}_1)$$

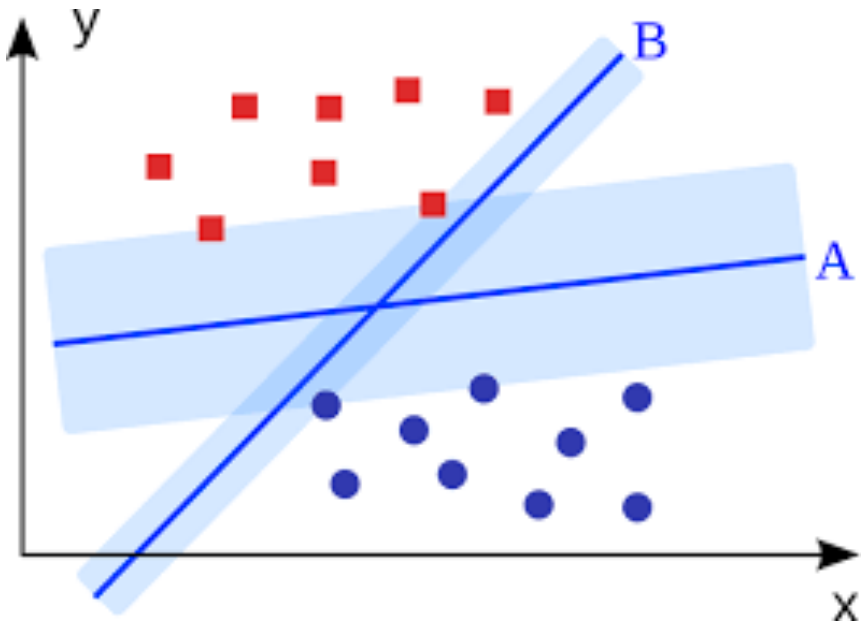
$$S_W = C_1 + C_2$$

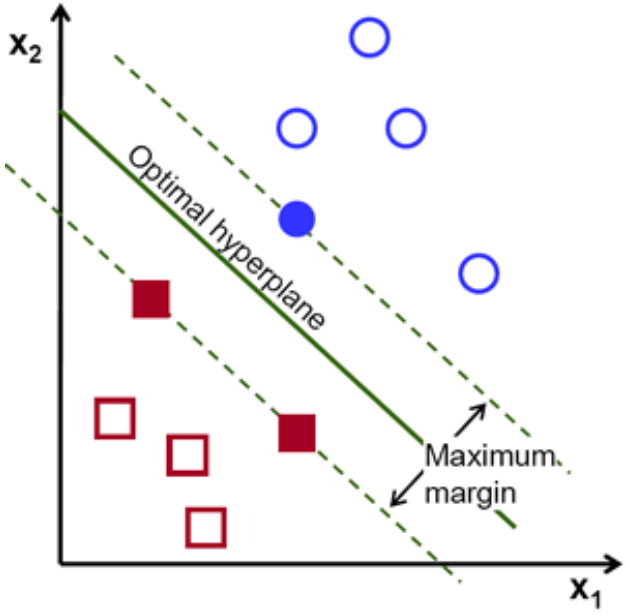
covariance matrices for classes 1 and 2



Support Vector Machine

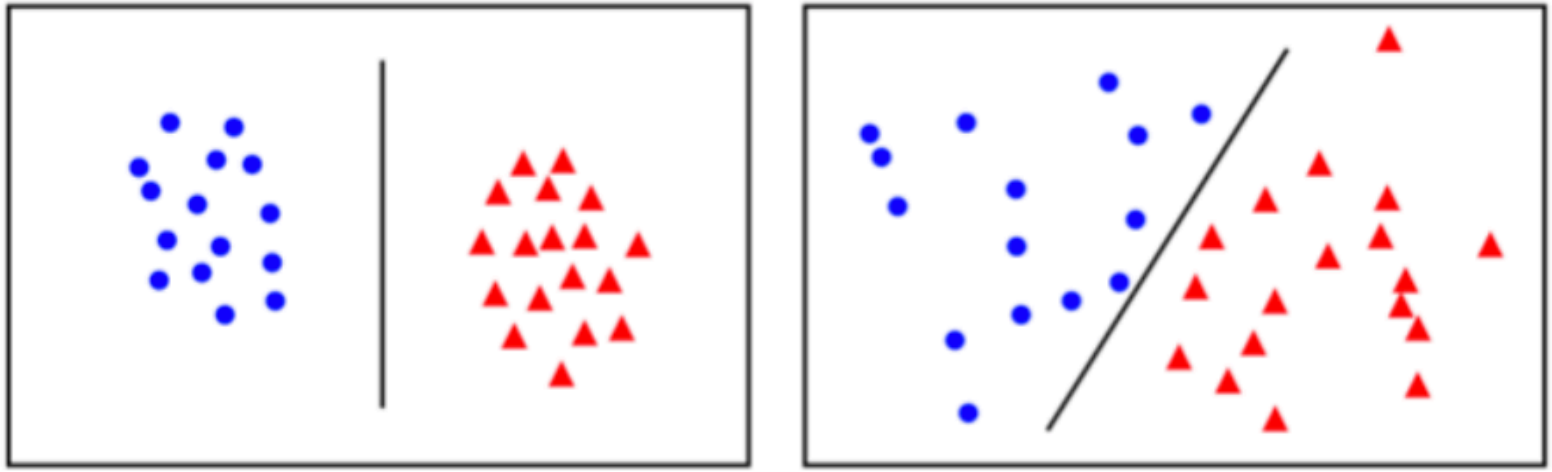
- find a decision boundary which maximise the possible margin (separation between the points)



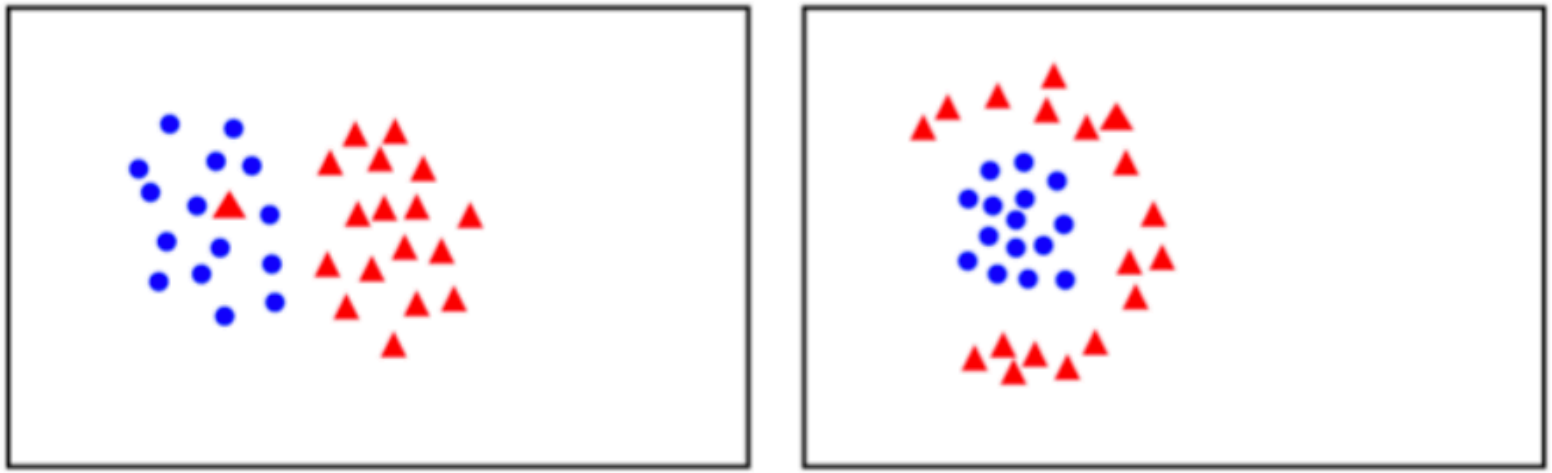


Linear Separability

linearly separable

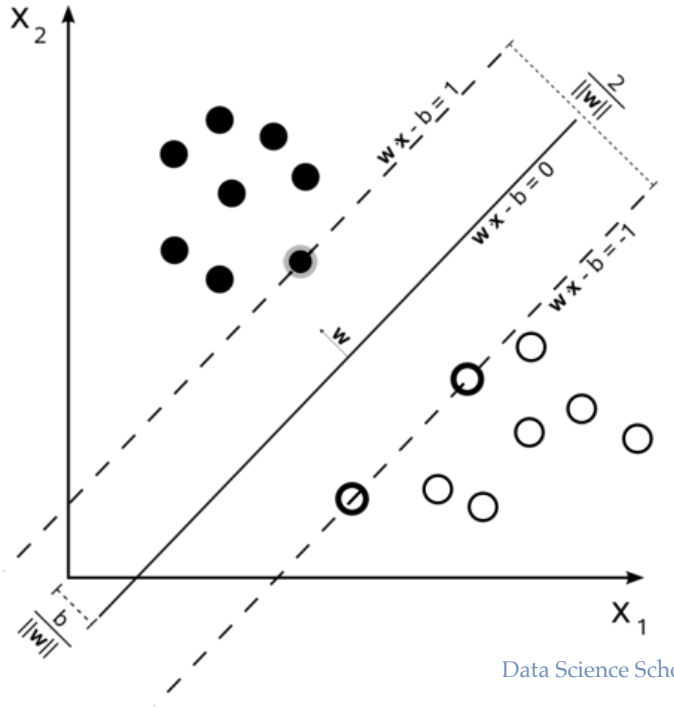


not linearly separable

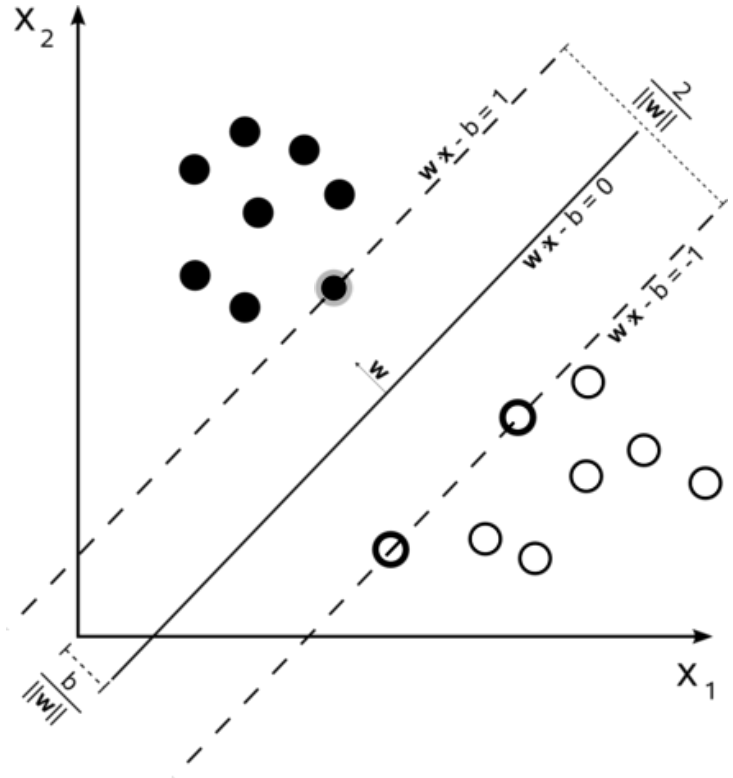


Best Decision Boundary

- How to find the optimal decision boundary ?
- For classes which are linearly separable a possible solution is to find:
  - maximum distance between decision boundary and the points (Maximum Margin Classifiers)



Best Decision Boundary



- Linear classifier :

$$h(\mathbf{x};\mathbf{w}) = \mathbf{w}^T\mathbf{x} + w_0$$

- Distance point x_i to decision boundary : ($y_i = \{-1,1\}$)

$$\frac{y_i(\mathbf{w}^T\mathbf{x}_i + w_0)}{\sqrt{\mathbf{w}^T\mathbf{w}}}$$

Optimization problem

$$\arg\max_{\mathbf{w},\, w_0} \left\{ \frac{1}{\sqrt{\mathbf{w}^T\mathbf{w}}} \min_i y_i(\mathbf{w}^T\mathbf{x}_i + w_0) \right\} \longrightarrow \arg\min_{\mathbf{w},w_0} \frac{1}{2}\mathbf{w}^T\mathbf{w}$$

$$s.\,t.\;\; y_i(\mathbf{w}^T\mathbf{x}_i + w_0) \geq 1 \text{ for all } i$$

Support Vector Machine

- Loss function for SVM

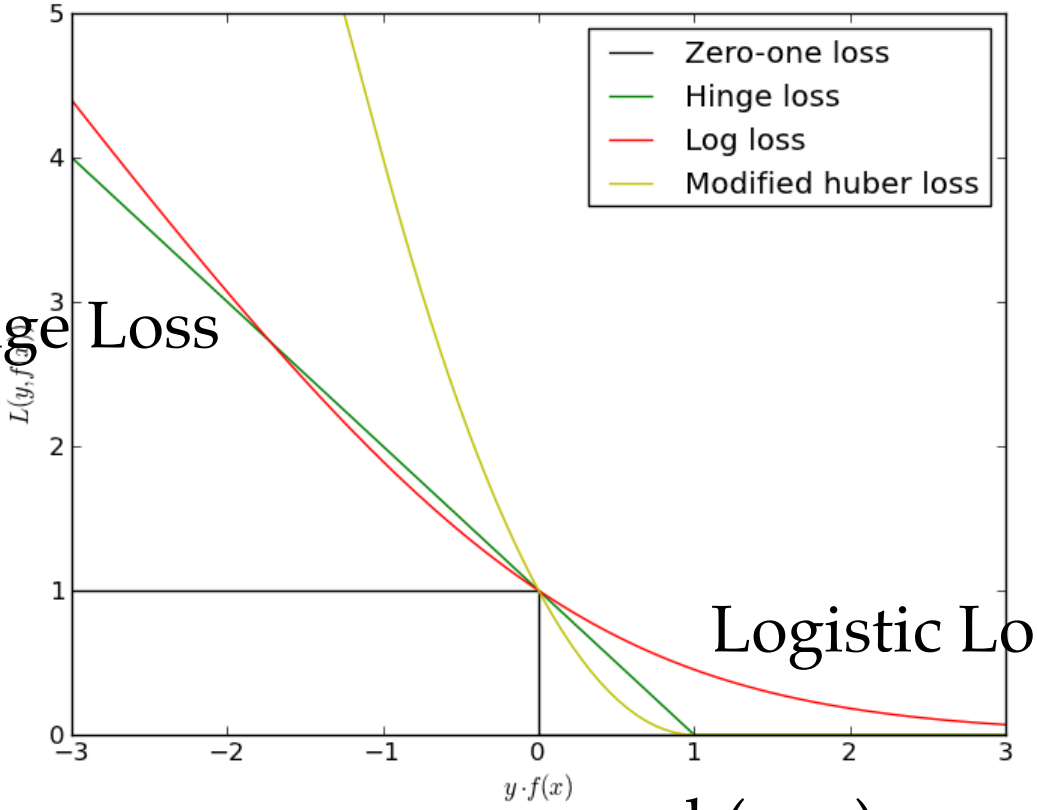
$$L(\mathbf{w}) = C \sum_i \max(0, 1 - y_i(\mathbf{w}^T \mathbf{x}_i + w_0)) + \frac{1}{2} \sum_i w_i^2$$

  - C is the regularisation hyper-parameter
    - control how much softening/hardening of the boundary is allowed.
    - increasing C makes the boundary harder

Hinge Loss Function

- The Hinge Loss function

$$L(y_i, x_i; \mathbf{w}) = \max(0, 1 - y_i h(x_i; \mathbf{w})) \quad y_i = \{-1, 1\}$$



Hinge Loss

Logistic Loss

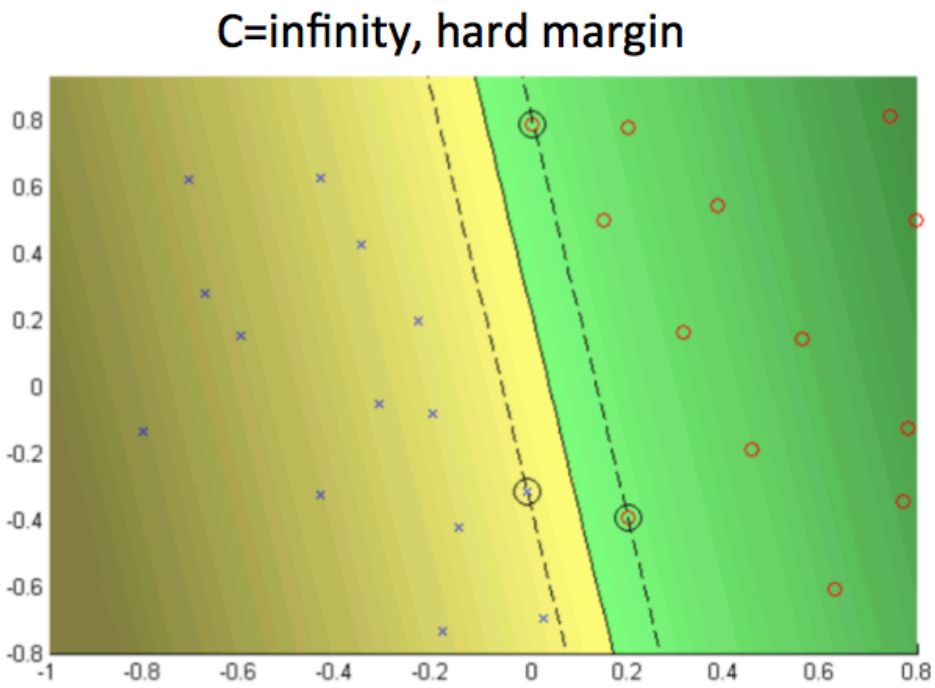
$y_i h(x;w)$

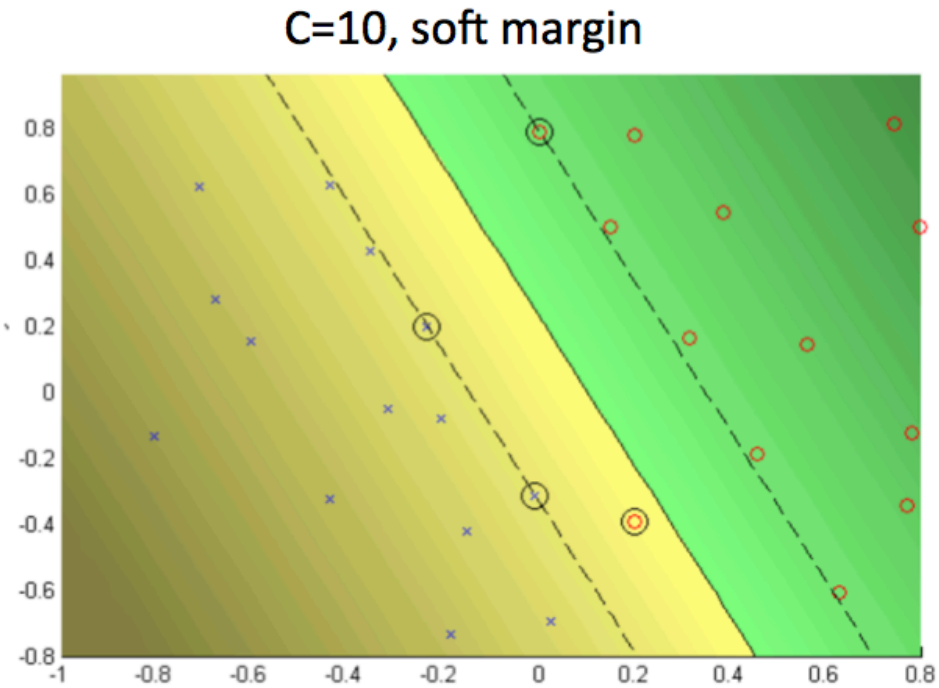
for linear classifiers

$$h(\mathbf{x}; \mathbf{w}) = \mathbf{w}^T \mathbf{x} + w_0$$

This loss function penalises only data points that give $|h(x)| < 1$

SVM : Soft/Hard Margin





SVM: Kernel Trick

- For data which are not linearly separable can use a mapping $\phi(x)$ from $\mathbf{R}^m \rightarrow \mathbf{R}^k$
- Decision boundary can be written as

$$h(\mathbf{x}; \mathbf{a}, w_0) = \sum_i a_i y_i \phi(\mathbf{x}_i)^T \phi(\mathbf{x}) + w_0$$

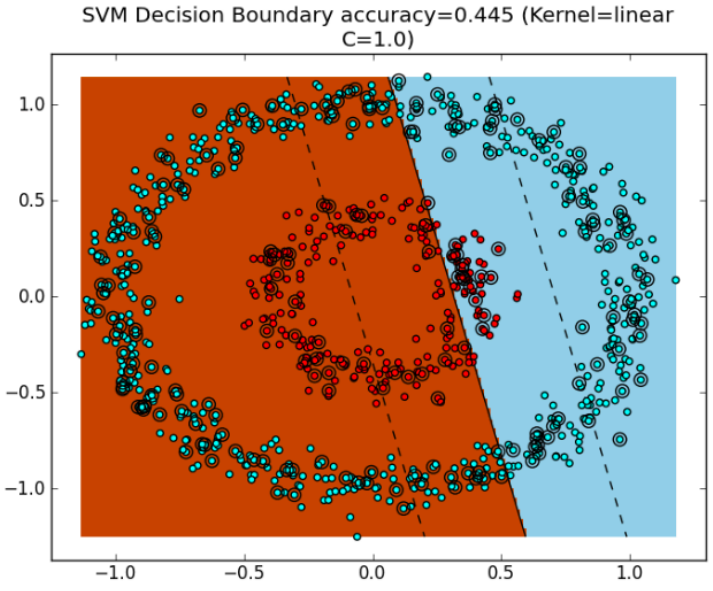
- Kernel function $K(x,x') = \phi(x)\,\phi(x')$
- **Kernel trick**:
  - compute the Kernel $K(x,x')$ without computing $\phi(x)$

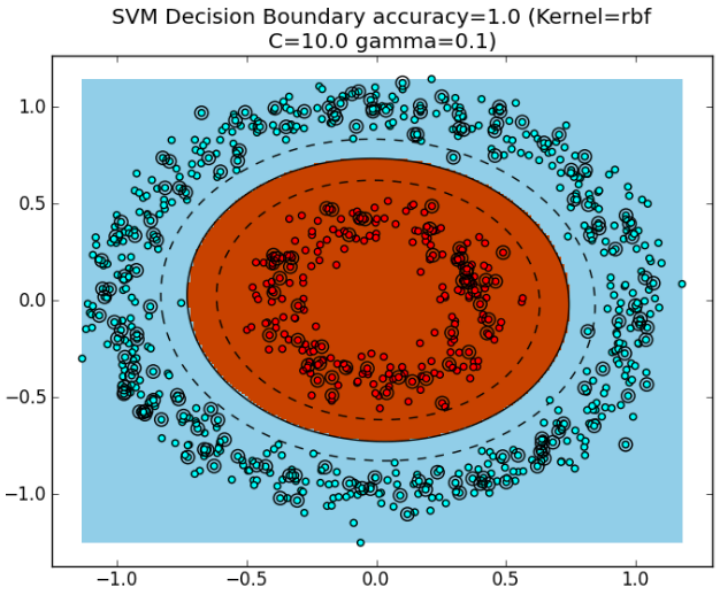
SVM Kernels

- Linear Kernel: $K(\mathbf{x}, \mathbf{x}') = \mathbf{x}^T \mathbf{x}'$
- Polynomial Kernel: $K(\mathbf{x}, \mathbf{x}') = (1 + \mathbf{x}^T \mathbf{x}')^q$
- Gaussian Kernel: $K(\mathbf{x}, \mathbf{x}') = \exp\left(-\frac{(\mathbf{x} - \mathbf{x}')^2}{2\sigma^2}\right)$

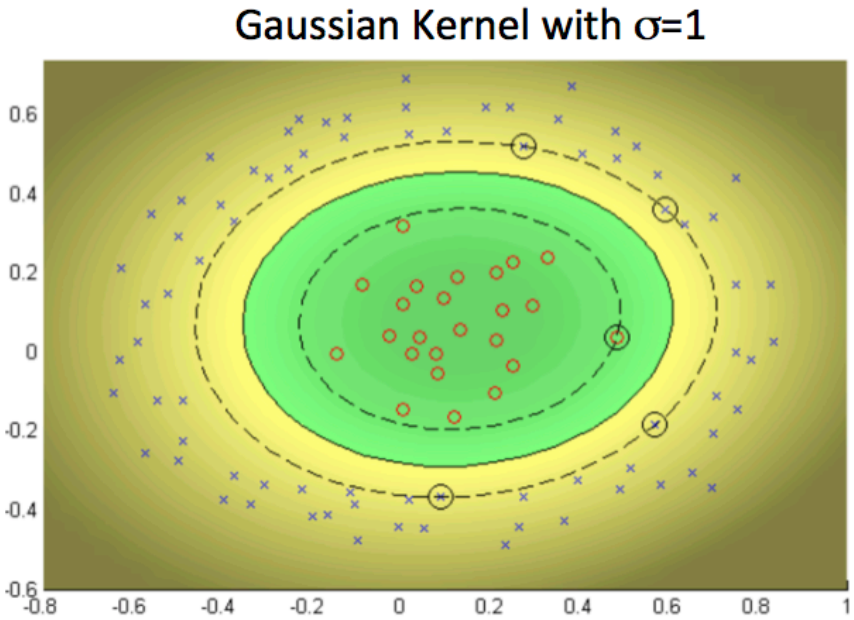
SVM: Kernel Examples

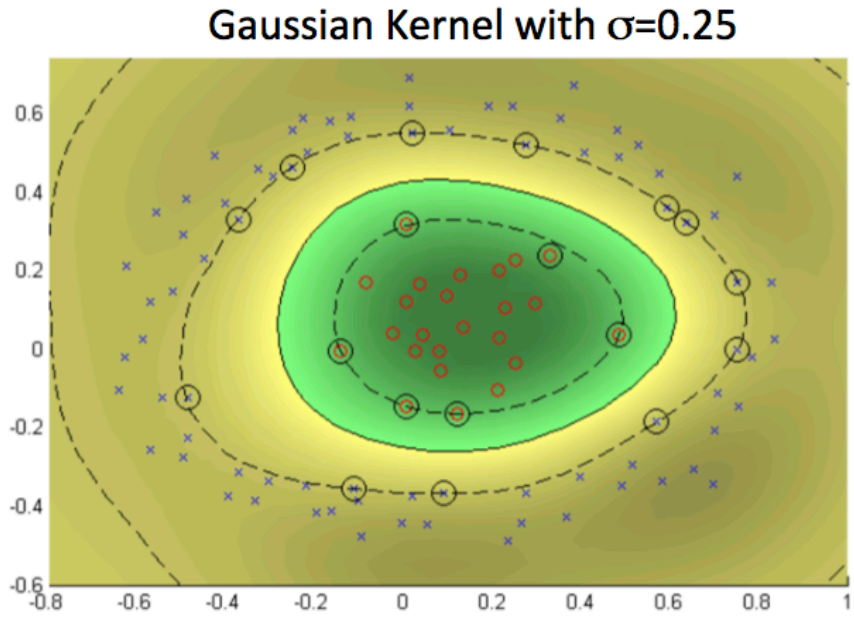
Linear kernel





Gaussian kernel





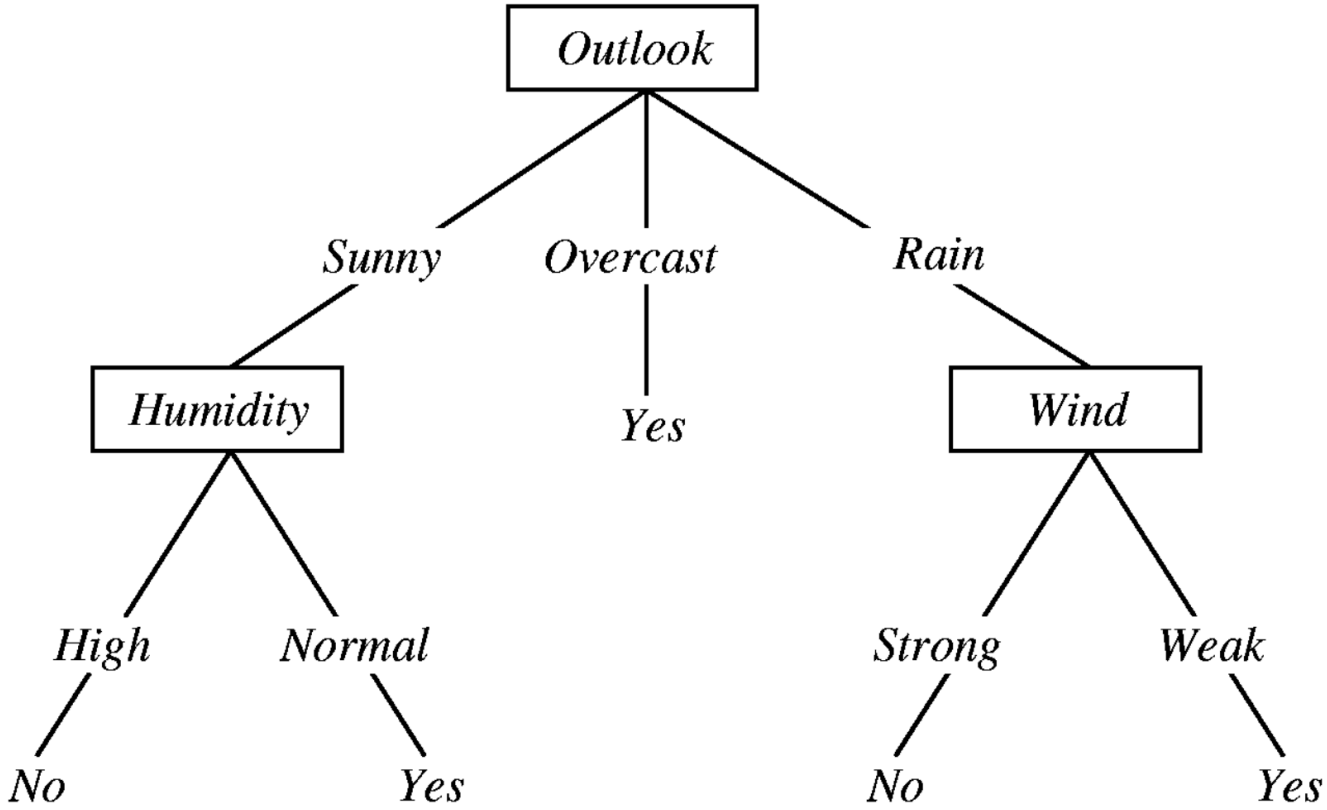
Conclusions on SVM

- Powerful tool effective with high dimensional data
- Flexible in dealing with non linearity by choosing different possible kernels
- Disadvantages:
  - complex algorithm for training
  - does not scale well with large data sets
    - complexity between $O(N_{\text{features}} \times N^2)$ and $O(N_{\text{features}} \times N^3)$ depending on implementation
    - *libSVM* best implementation (available in TMVA with R interface or in scikit-learn)
    - TMVA native SVM implementation is not very computational efficient

Decision Tree

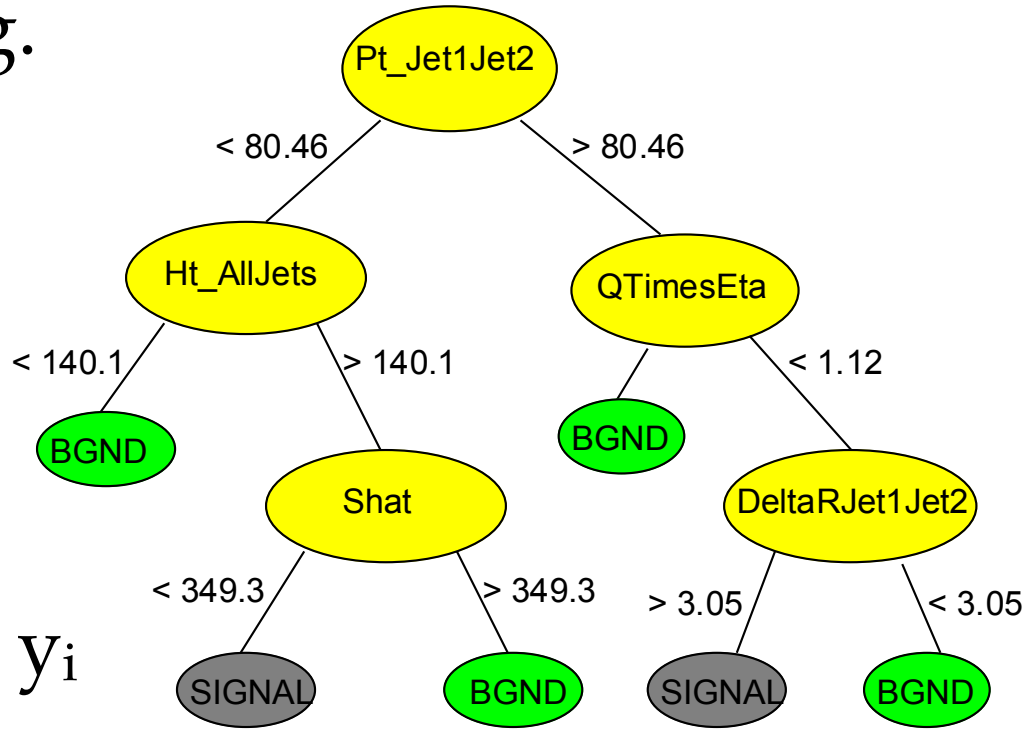
Example:

- Predict to play or not to play golf depending on whether conditions



Decision Trees

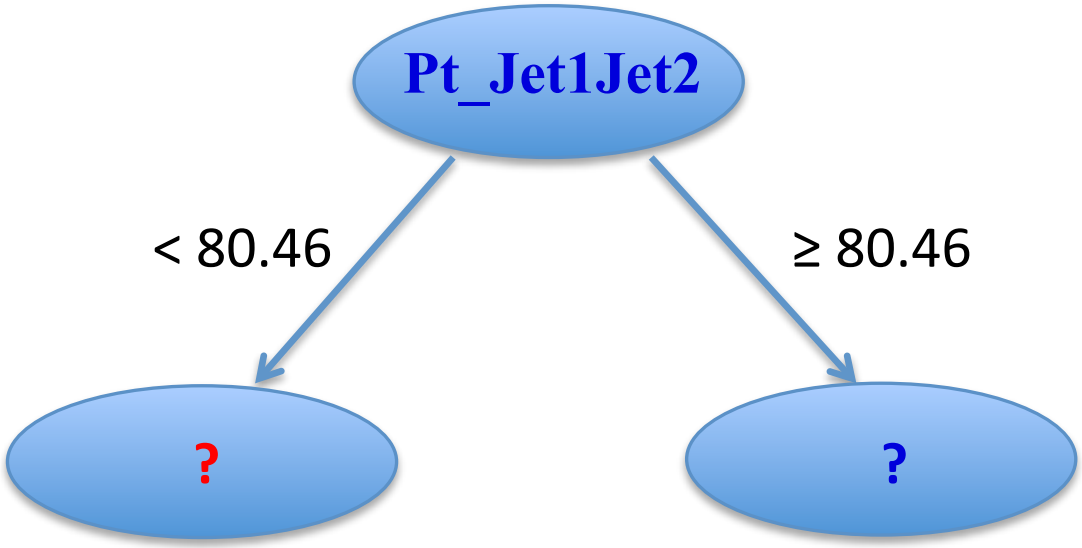
- **Node** :
  - test one attribute x_i (e.g. Wind)
- **Branch**:
  - select one value for x_i
- **Leaf**:
  - predict probability for y_i $P(y \mid x)$



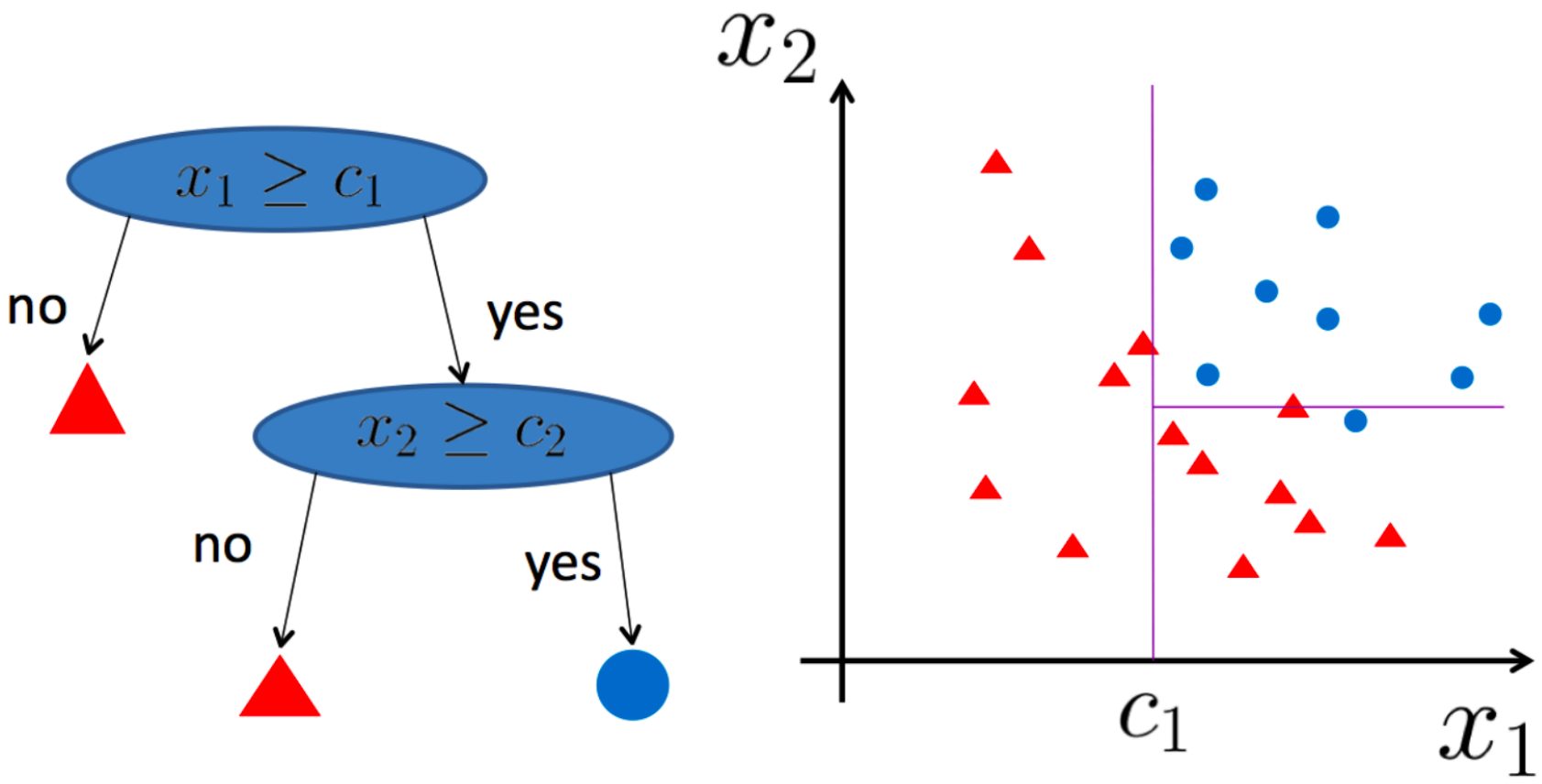
- Decision trees are like multi-dimensional histograms (bins are constructed recursively)

Building a Tree

- Scan along each input variable x_i and propose a **DECISION**
  - A cut on value that maximised class separation



Decision Tree: Splitting

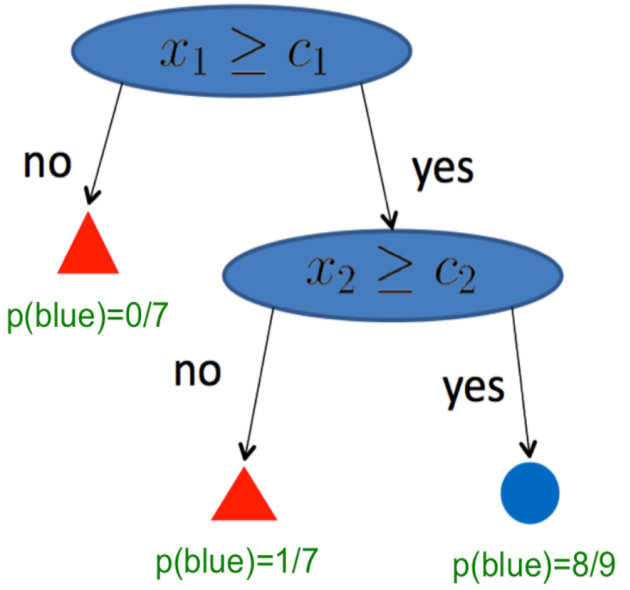


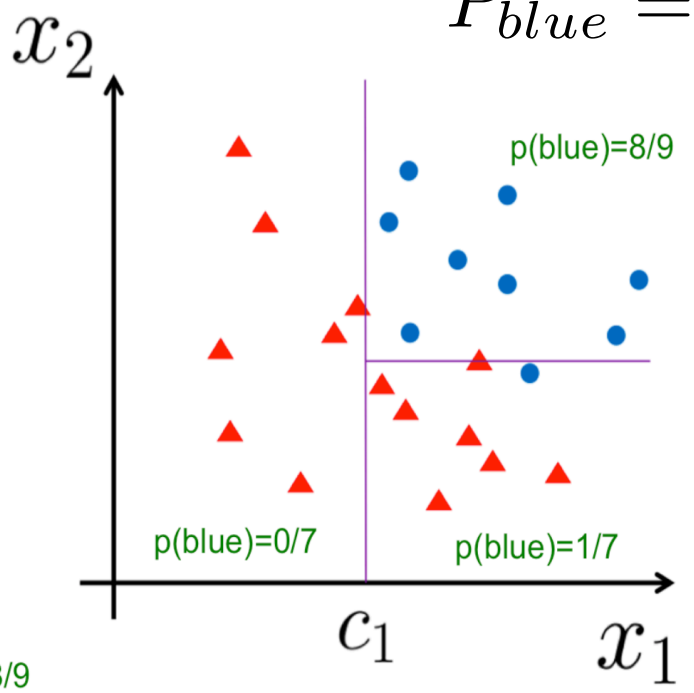
Partition the data using a sequence of cuts

Building Decision Trees

- Choose decisions that maximise separation between classes (e.g. signal and background)
- After each decision one can estimate the class probabilities p(yi | x)

$$P_{blue} = \frac{N_{blue}}{N_{blue} + N_{red}}$$





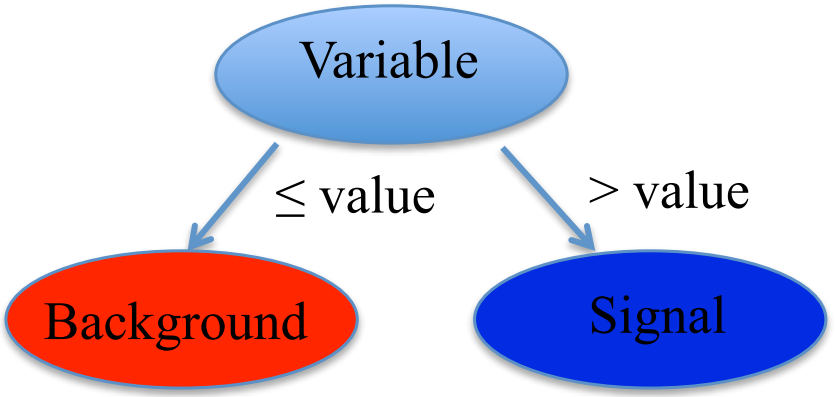
Training Decision Trees

- Measure separation gains with impurity functions H to have optimal split

$$G(Q,\theta) = \frac{n_{\text{left}}}{N_m} H(Q_{\text{left}}(\theta)) + \frac{n_{\text{right}}}{N_m} H(Q_{\text{right}}(\theta))$$

  - optimize splitting by minimise the impurity

$$\theta^* = \arg\min_{\theta} G(Q,\theta)$$



  - Greedy training: optimize one splitting at a time and not all together at the same time

Decision Trees: Impurity Functions

- Measure separation gains with impurity functions H to have optimal split

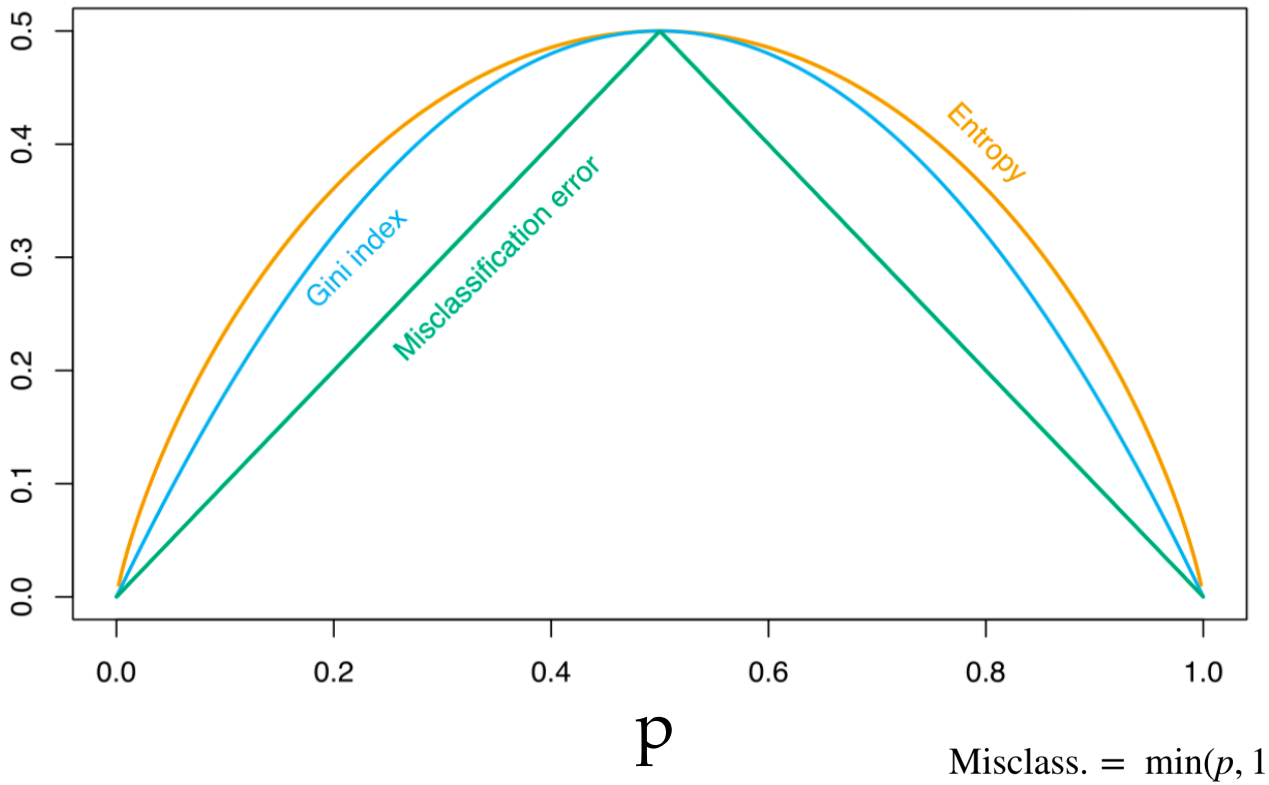
$$G(Q,\theta) = \frac{n_{\text{left}}}{N_m} H(Q_{\text{left}}(\theta)) + \frac{n_{\text{right}}}{N_m} H(Q_{\text{right}}(\theta))$$

- Classification
  - Proportion of class k in node m: $p_{mk} = \frac{N_k}{N_m}$
  - Gini: $H(X_m) = \sum_k p_{mk}(1 - p_{mk})$
  - Cross entropy: $H(X_m) = -\sum_k p_{mk} \log(p_{mk})$
  - Miss-classification: $H(X_m) = 1 - \max_k(p_{mk})$

Splitting

- Impurity as function of proportion p

H(p)



p

Gini index and Cross-entropy rescaled to pass through (0.5,0.5)

$$\text{Misclass.} = \min(p, 1-p)$$

$$Gini = p(1-p)$$

$$Entropy = -p\log p - (1-p)\log(1-p)$$

Decision Tree: Regression

- In regression we want to estimate a continuous target y
  - compute average of y in region after splitting

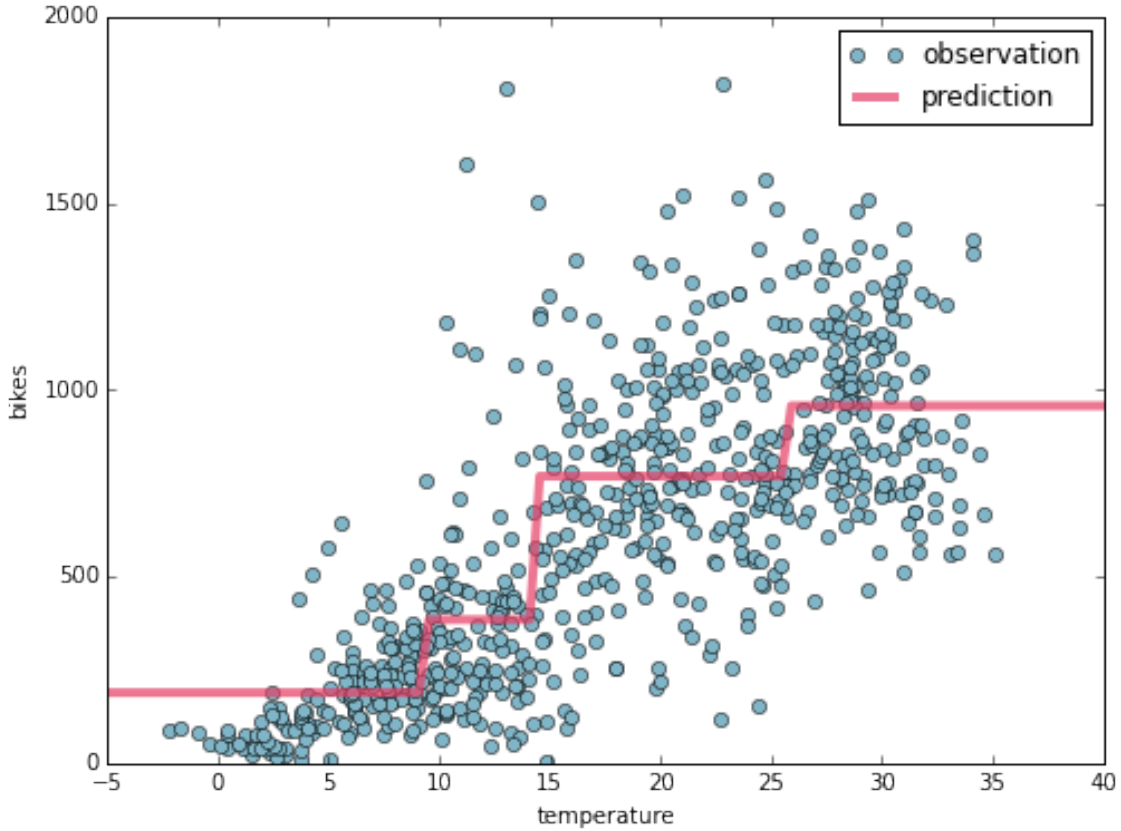
$$c_m = \frac{1}{N_m} \sum_{i \in N_m} y_i$$

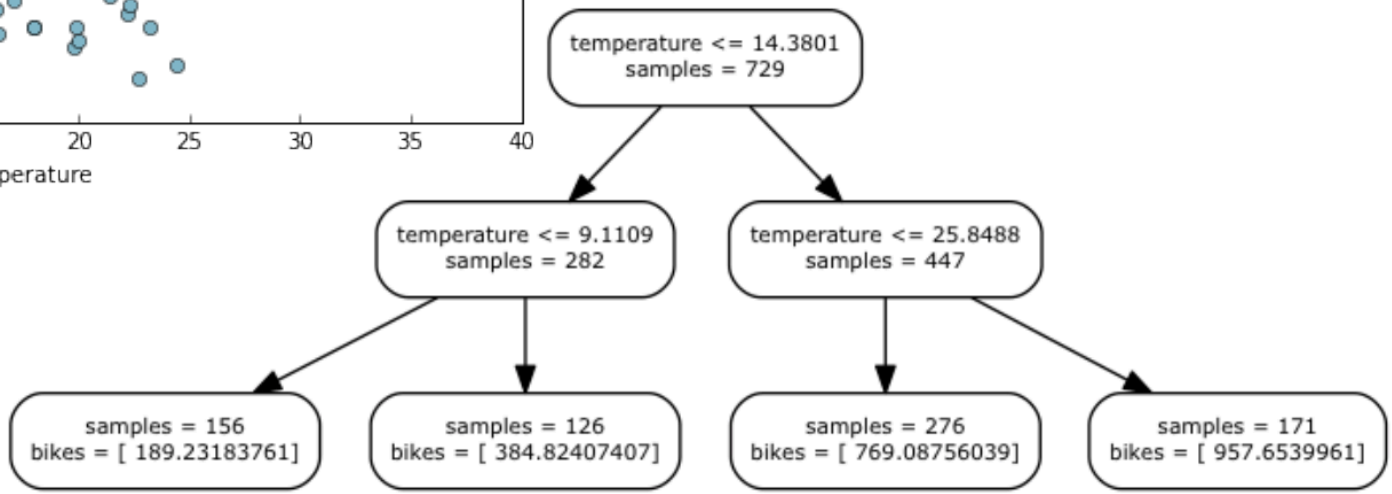
  - use it for computing square error

$$H(X_m) = \frac{1}{N_m} \sum_{i \in N_m} (y_i - c_m)^2$$

  - minimize total square error after splitting
- It is like finding an optimal binning of the data

Example of Decision Tree Regression

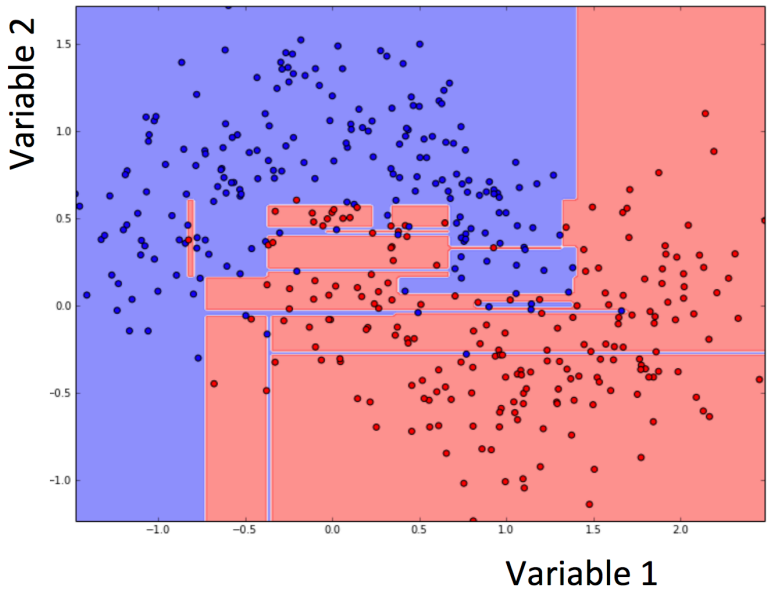




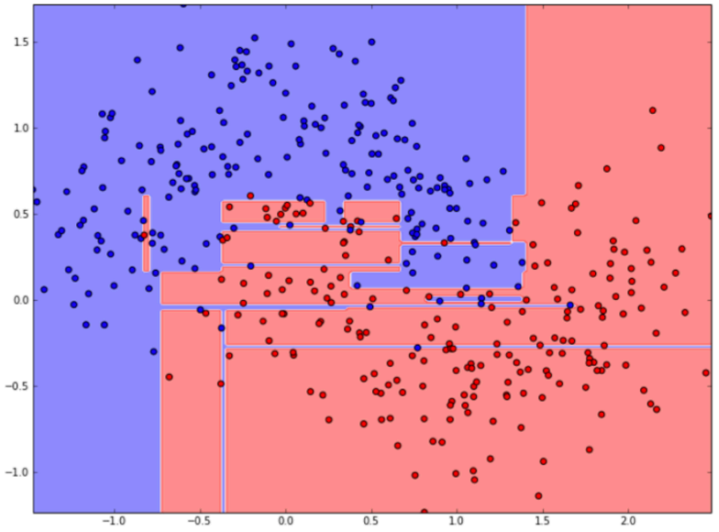
[Cambrige Coding Academy]

Building Decision Trees

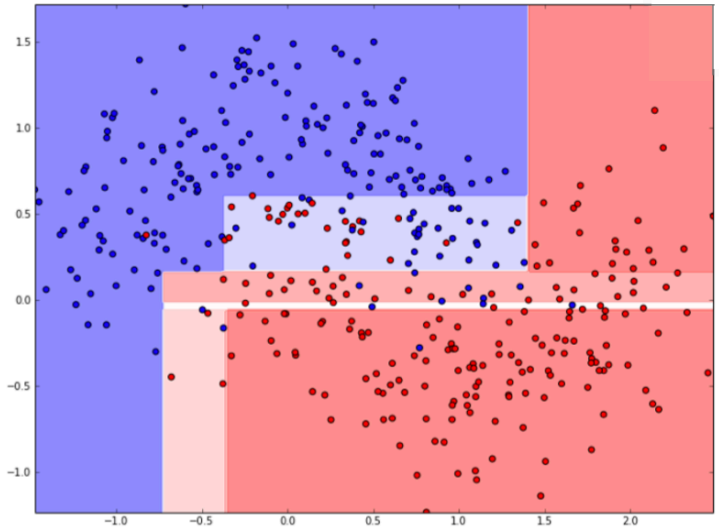
- By applying different decision that maximised the separation, we build regions of increasing purity.
- When to stop building a tree ?
  - in principle we could stop when all events are classified
    - overfitting
  - Need to stop then earlier. Different rules:
    - fixed depth
    - fixed minimum sample
    - minimum information gain, etc..



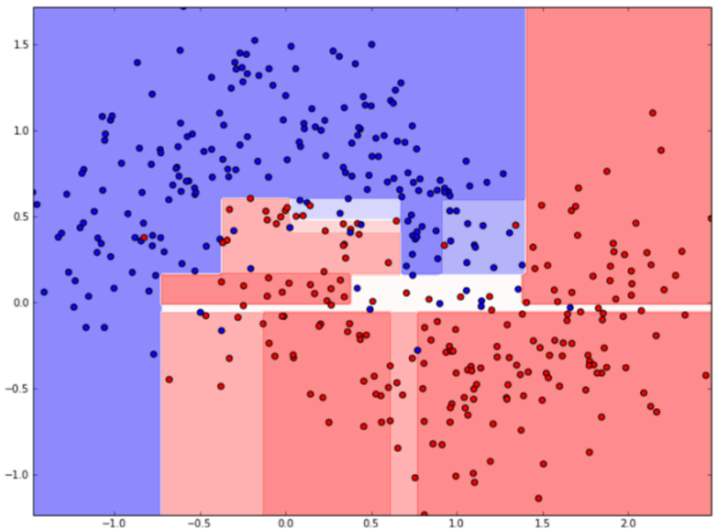
Mitigating Overfitting



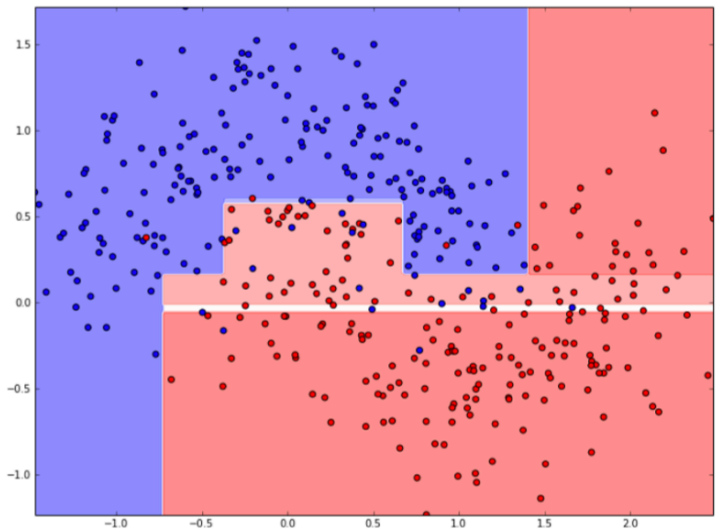
no pre-stopping



max_depth



min # of samples in leaf



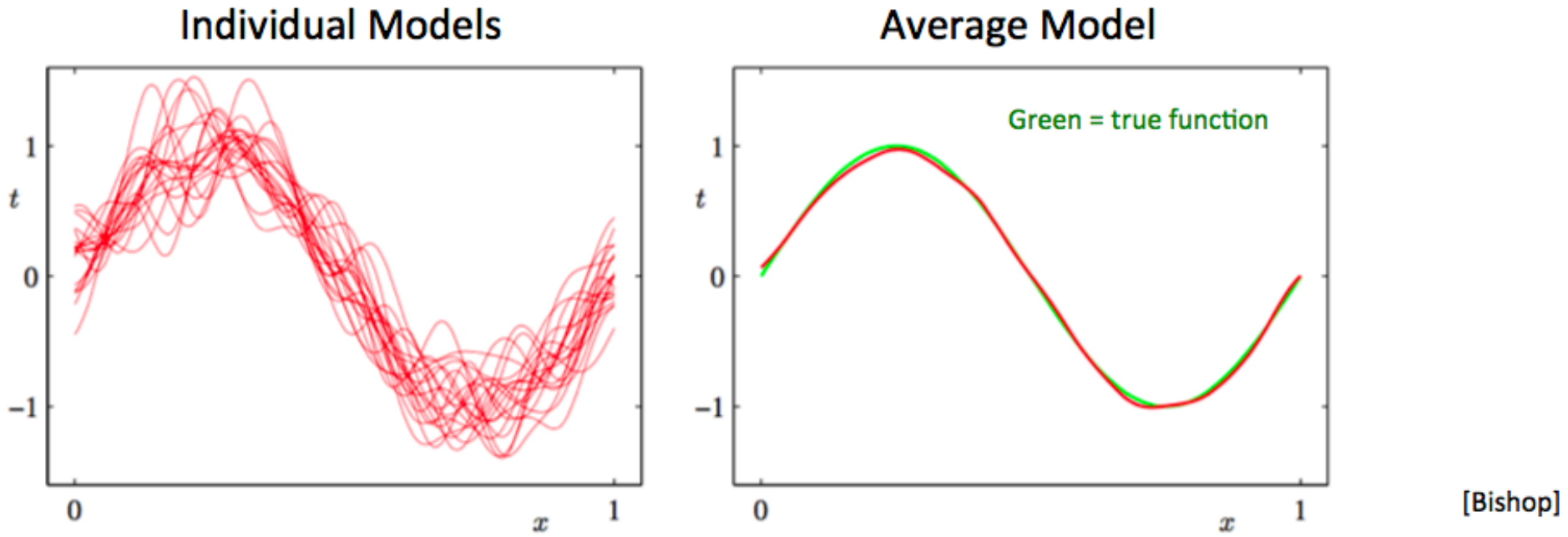
maximal number of leaves

[Rogozhnikov]

Ensemble Methods

- How can we reduce variance of the model and not increasing the variance ?
- Train slightly different models
  - take majority vote (classification)
  - average prediction (regression)
- Bias does not increase because average of ensembles
- Variance decreases because spurious pattern picked by a model will not be picked by others

Example Ensemble Methods



[Bishop]

- Ensemble methods are very useful to overcome problem of overfitting with decision trees
- Combining several methods has been found to be very powerful
- Two techniques exist for ensemble methods:
  - **Bagging** and **Boosting**

Ensemble Methods

- **Bagging** (Bootstrap aggregation)
  - sample dataset with replacement and train a different model on each trained set
  - take average or classify using a majority vote
  - **Random Forest**
    - bagging using randomised trees
    - random subset of features used at each split

$$h(x) = \frac{1}{N_{trees}} \sum_{i=1}^{N_{trees}} h_i(x)$$

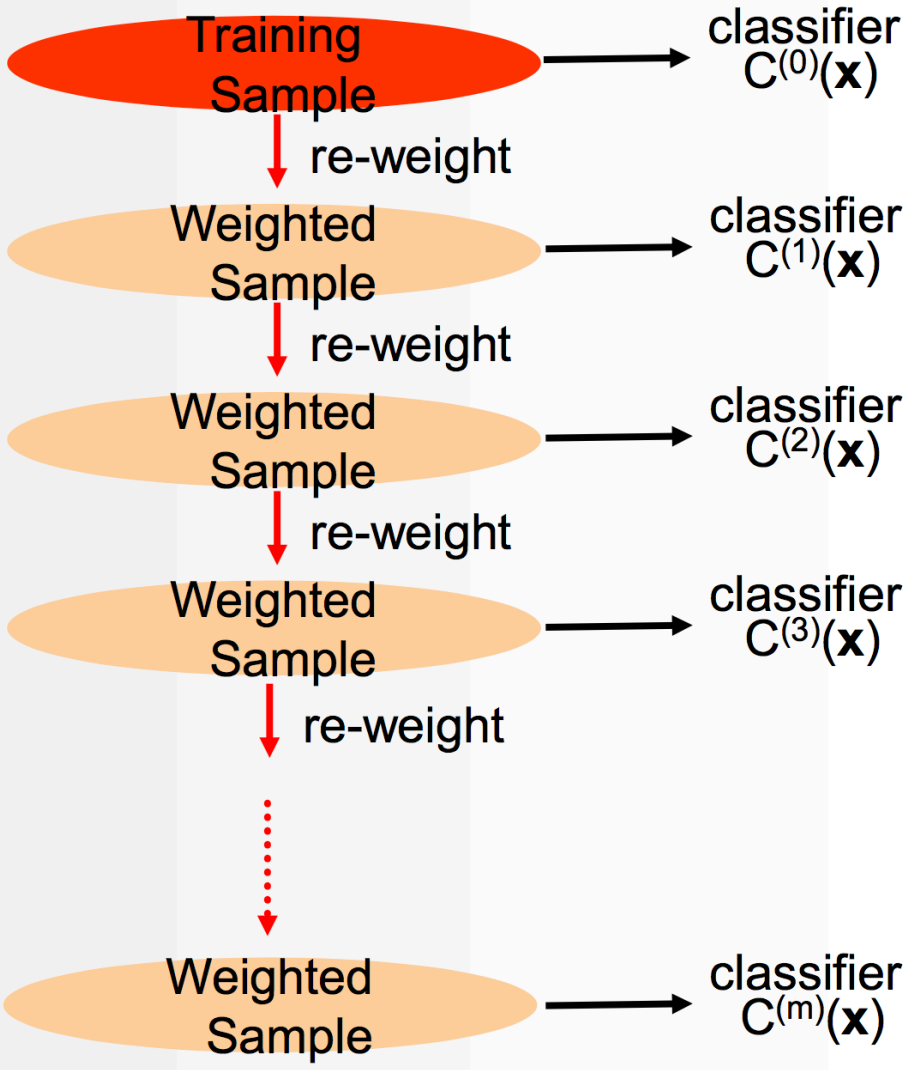
- **Boosting**
  - Each tree trained on a different weighting of full training set.
  - give more weight to events previously not correct classified
  - Popular algorithms: **AdaBoost**, **GradientBoost**

Adaptive Boosting

- Train in stages
- Adaptive weights
- ADABoost: Freund & Schapire 1997
- Misclassified events get a larger weight going into the next training stage
- Classify with a majority vote from all trees
- Works very well to improve classification power of "greedy" decision trees

$$h(x) = \sum_{i=1}^{N_{trees}} \alpha_i h_i(x) / \sum_{i=1}^{N_{trees}} \alpha_i$$

ADABoost



AdaBoost re-weights events misclassified by previous classifier by:

$$\frac{1-f_{err}}{f_{err}} \quad \text{with:}$$

$$f_{err} = \frac{\text{misclassified events}}{\text{all events}}$$

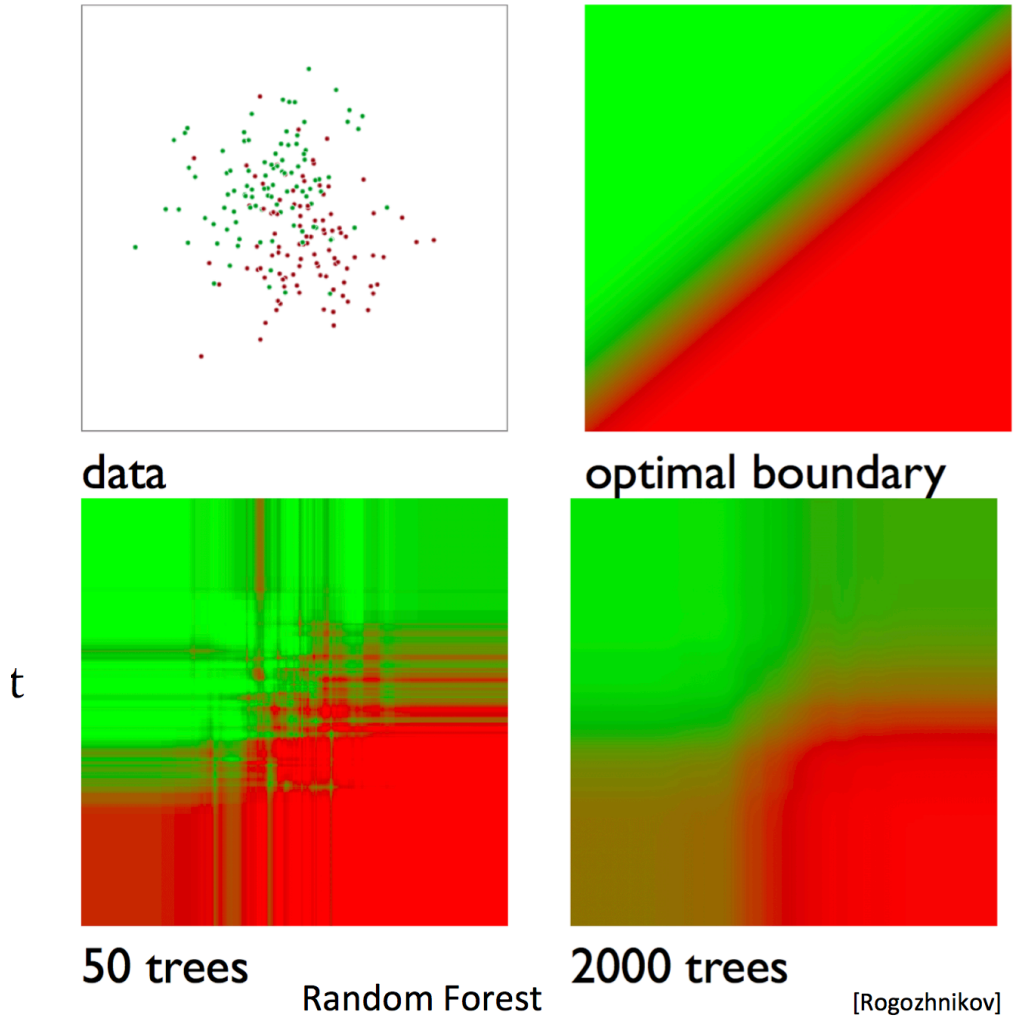
AdaBoost weights the classifiers also using the error rate of the individual classifier according to:

$$y(x) = \sum_{i}^{N_{Classifier}} \log\left(\frac{1-f_{err}^{(i)}}{f_{err}^{(i)}}\right) C^{(i)}(x)$$

[E. v. Toerne]

Example Decision Trees

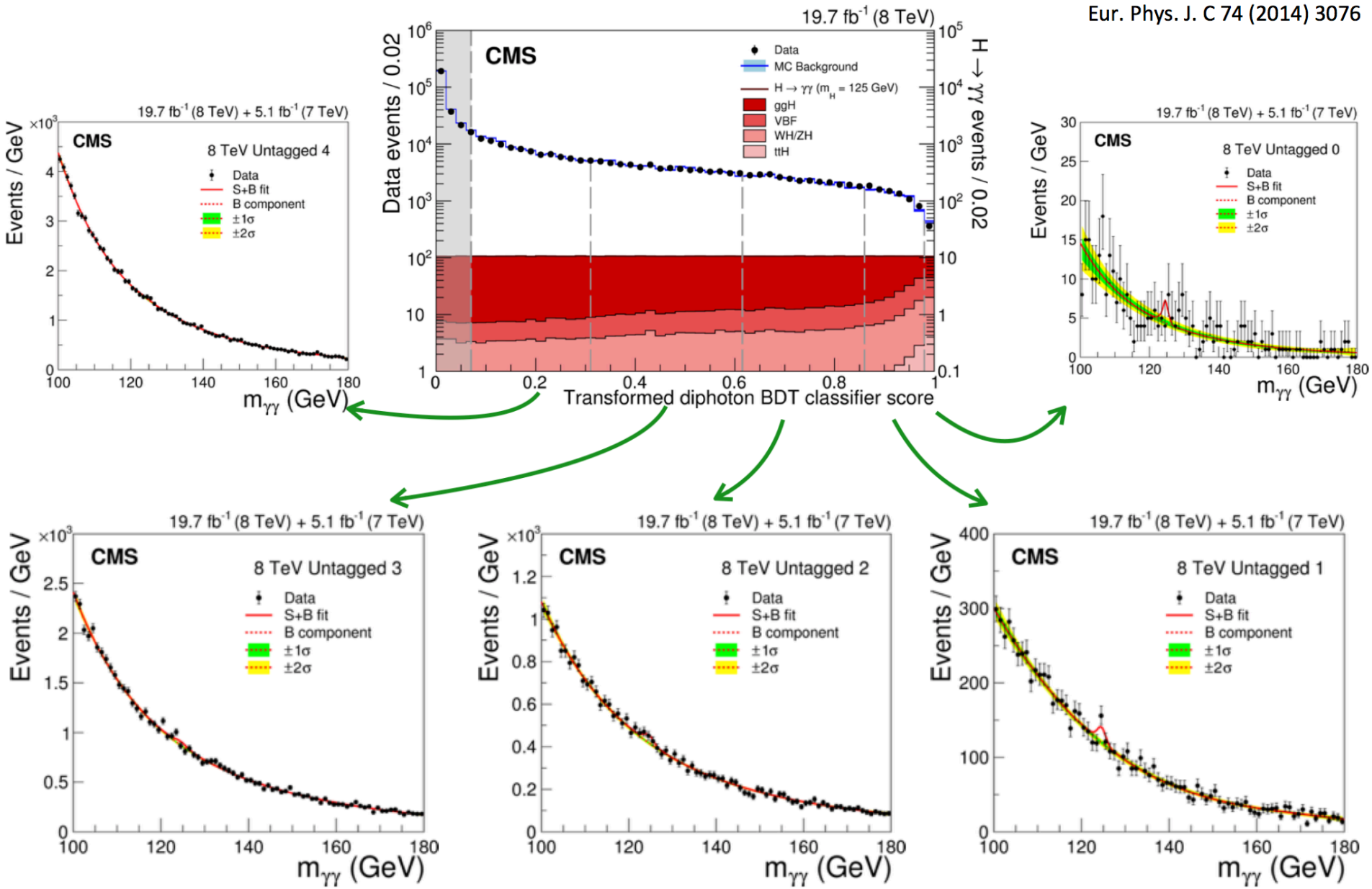
- Example of Random Forest



Conclusions on Decision Trees

- Methods based on ensembles of Decision Trees work very well
  - several variants exists (Random Forest, ADABoost, Gradient Boost)
  - Rarely overfitting
  - Robust to noisy data
  - Can use heterogeneous or missing inputs
  - Easy and fast to train them
- Example: using 179 classifier on 121 public data sets:
  - Random Forest found best classifier

HEP Example: BDT for H→γγ



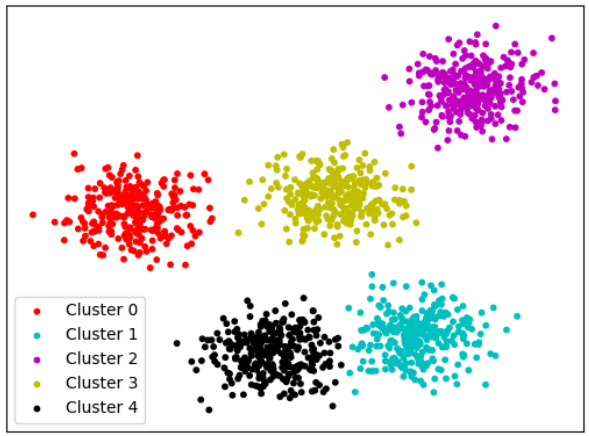
Decision Trees in TMVA

- TMVA provides a very good implementation of Decision Trees (BDT)
  - ADABoost with 3 different variants
  - Gradient Boost
  - Bagging
  - Random Forest (bagging and randomised trees)
- Several configuration options available (See TMVA Users Guide)

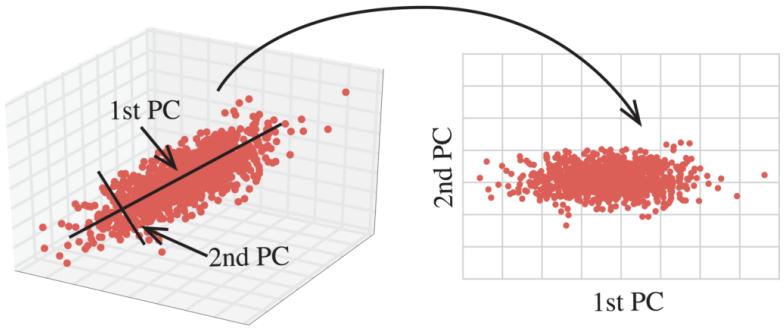
Unsupervised Learning

- Given some data $D = \{x_i\}$, but no labels
- Find possible structure in the data

  - Clustering:
    - partition the data into groups
      $D = \{ D_1 \cup D_2 \cup D_3 ..... \cup D_n \}$



  - Dimensionality reduction:
    - find a low dimensional representation of the data with a mapping $Z = h(X)$



Principal Component Analysis

- Unsupervised Learning Method
- Given data D = $\{\mathbf{x_i}\}$
  - find directions that explains the most variation of the data
  - Equivalent to find eigenvectors of the data covariance matrix

$$\mathbf{S} = \frac{1}{N}\sum_{i=1}^{N}(\mathbf{x}_i - \bar{\mathbf{x}})^2$$

Variance of projected data

Unit length vector constraint

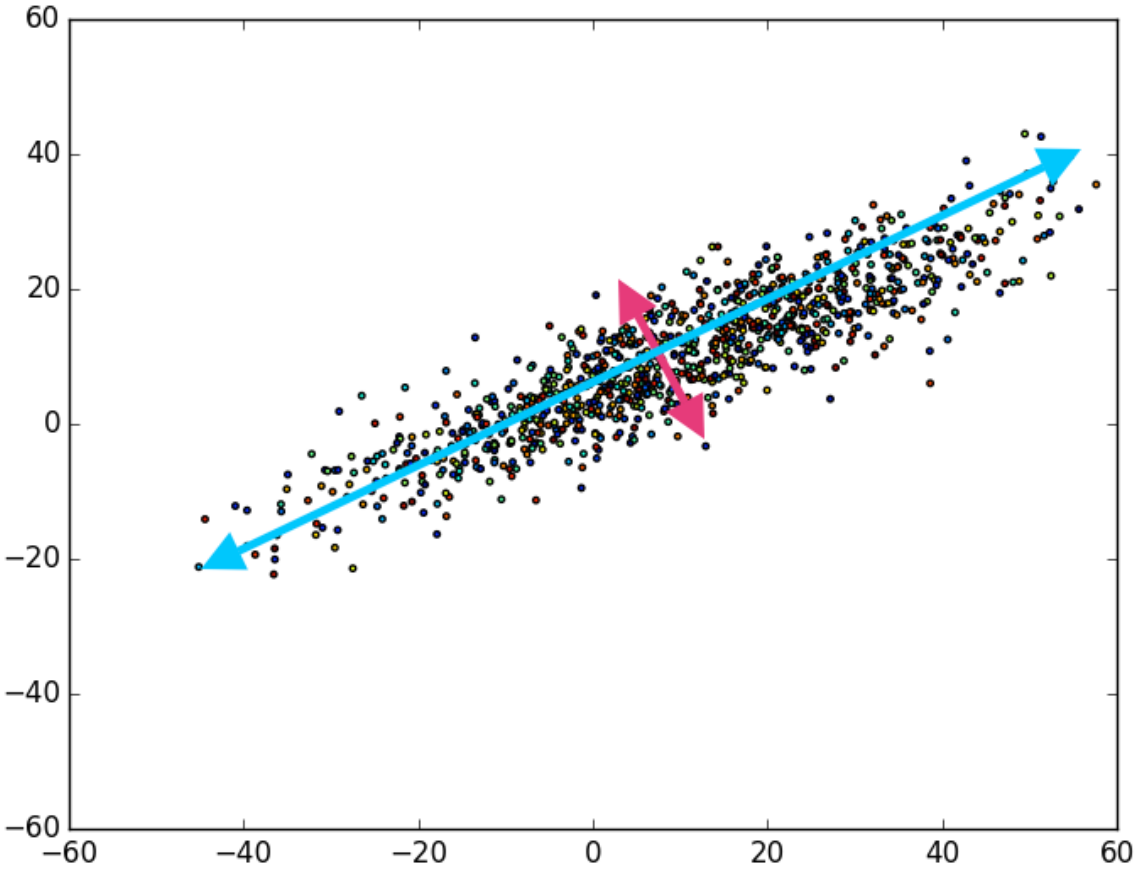
projected direction

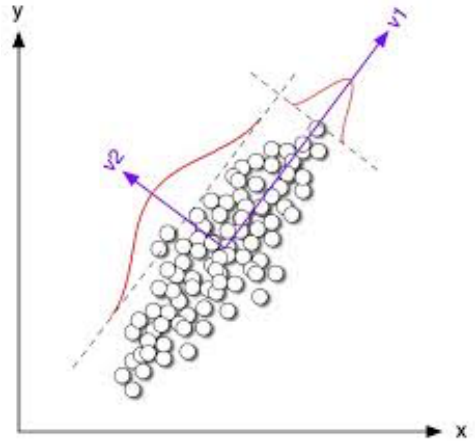
$$\mathbf{u}_1^* = \arg\max_{\mathbf{u}_1} \; \mathbf{u}_1^T \mathbf{S} \mathbf{u}_1 + \lambda(1 - \mathbf{u}_1^T \mathbf{u}_1)$$

$$\rightarrow \mathbf{S}\mathbf{u}_1 = \lambda \mathbf{u}_1$$

$\mathbf{u}_1$ is eigenvector of S

Principal Component Analysis





K-Means clustering

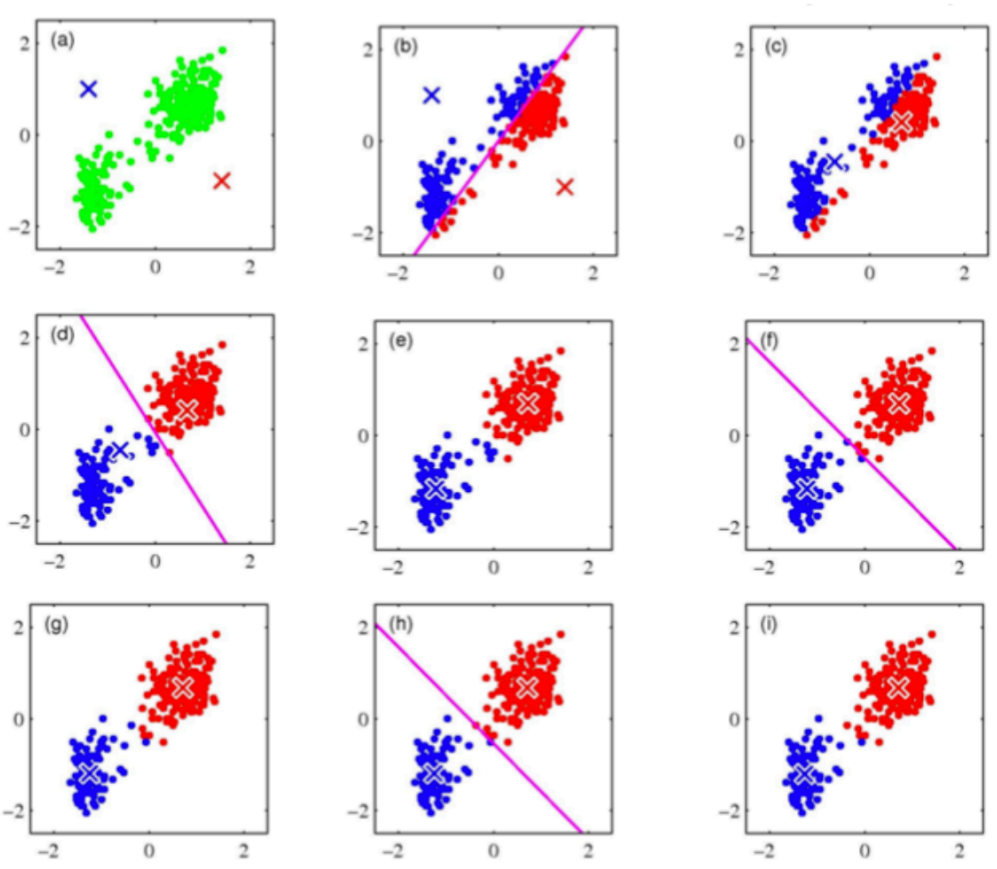
- Initialize means μ_k randomly
  - e.g using k-means++ initialization
- Assign data to closest prototype cluster looking at the minimal distance

$$\min_{k \in \{1 \ldots K\}} \sqrt{(\mathbf{x}_i - \mu_k)^2}$$

- Update the μ_k values using prototype clusters

$$\mu_k = \frac{1}{n_k} \sum_{i \in S_k} \mathbf{x}_i$$

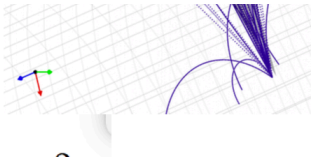
- Re-assign data using new μ_k values
- Iterate until convergence



Clustering in HEP

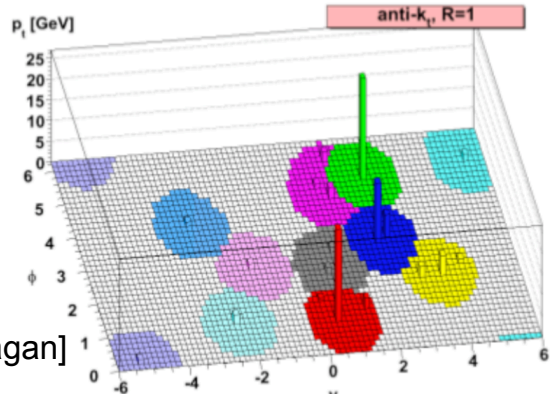
Jet clustering

- Sequential clustering algorithm
- Combine particles together to form jets
  - Compute distance between pseudojets i and j



$$d_{ij} = \min\left(k_{\mathrm{T}i}^{2p}, k_{\mathrm{T}j}^{2p}\right)\frac{\Delta_{ij}}{D} \qquad \Delta_{ij}^2 = (y_i - y_j)^2 + (\phi_i - \phi_j)^2$$

  - Distance between pseudojet and beam $d_{iB} = k_{\mathrm{T}i}^{2p}$
  - Find smallest distance between pseudojets d_{ij} or d_{iB}
    - Combine (sum 4-momentum) of two pseudojets if d_{ij} smallest
    - If d_{iB} is smallest, remove pseudojet i, call it a jet
    - Repeat until all pseudojets are jets



[M. Kagan]