Machine learning methods applied to the analysis of central exclusive production events in ALICE

Sebastian Ratzenböck[1]

[1]Stefan Meyer Institut
Österreichische Akademie der Wissenschaften

26. April 2018

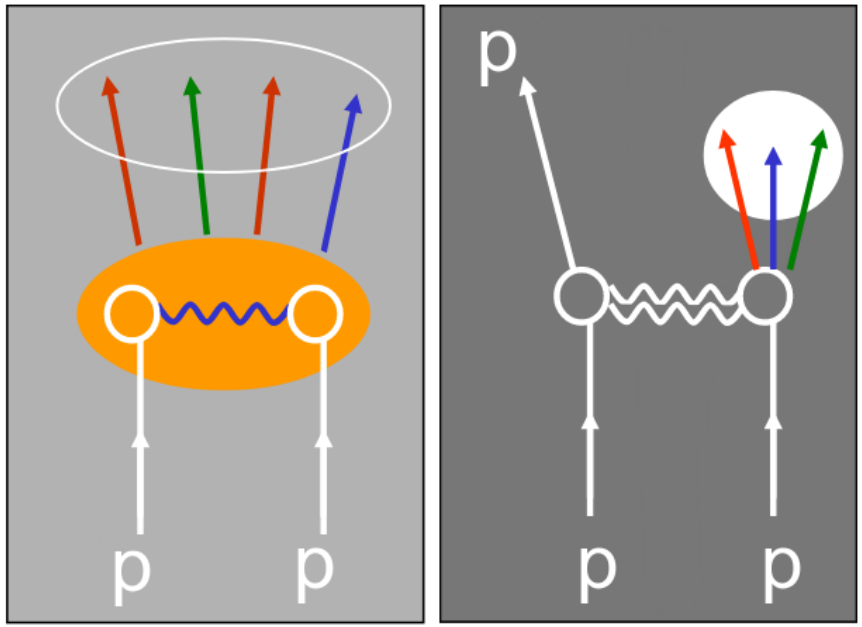
ÖAW ÖSTERREICHISCHE AKADEMIE DER WISSENSCHAFTEN

Outline

Diffractive events

Diffractive events are reactions where no quantum numbers are exchanged

Incident hadrons aquire color
$\rightarrow$ break apart



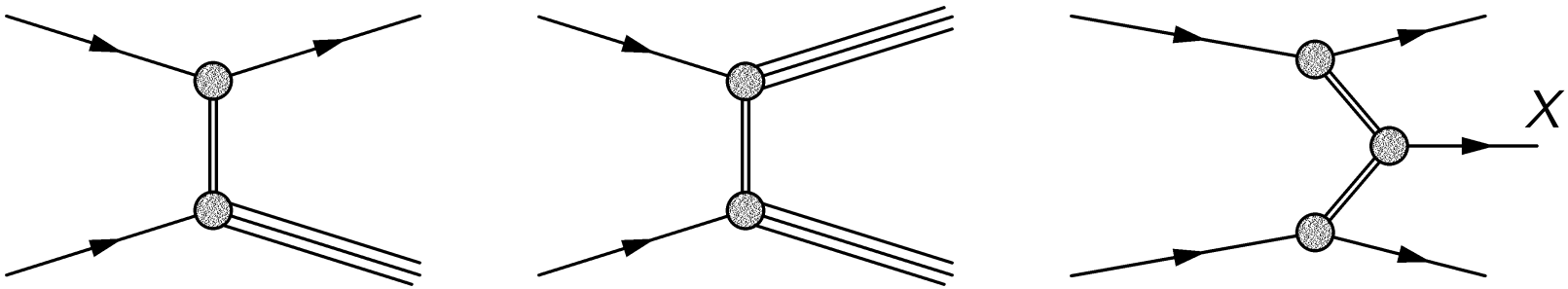
Colorless exchange with *vacuum* quantum numbers

η gap exponentially supressed

η gap not supressed

Diffractive processes

Different types of diffractive events are distinguished



Single diffractive | Double diffractive | CEP

Described by *Regge* theory

Regge theory - short overview

- Reactions characterized by a color neutral t -channel exchange carrying vacuum quantum numbers
- t-channel process governed by exchange of resonances sitting on *Regge trajectory* $\alpha(t) = \alpha + \alpha' t$
- Resonance of mass m sitting on the trajectory at $t = m^2$ has angular momentum of value $\alpha(m^2)$
- Trajectory makes contribution to total inelastic cross section $\sigma_{tot}(s) \propto s^{\alpha-1}$

Regge theory - short overview

- Reactions characterized by a color neutral t -channel exchange carrying vacuum quantum numbers
- t-channel process governed by exchange of resonances sitting on *Regge trajectory* $\alpha(t) = \alpha + \alpha' t$
- Resonance of mass m sitting on the trajectory at $t = m^2$ has angular momentum of value $\alpha(m^2)$
- Trajectory makes contribution to total inelastic cross section $\sigma_{tot}(s) \propto s^{\alpha-1}$

Regge theory - short overview

- Reactions characterized by a color neutral t -channel exchange carrying vacuum quantum numbers
- t-channel process governed by exchange of resonances sitting on *Regge trajectory* $\alpha(t) = \alpha + \alpha' t$
- Resonance of mass m sitting on the trajectory at $t = m^2$ has angular momentum of value $\alpha(m^2)$
- Trajectory makes contribution to total inelastic cross section $\sigma_{tot}(s) \propto s^{\alpha-1}$

Regge theory - short overview

- Reactions characterized by a color neutral t -channel exchange carrying vacuum quantum numbers
- t-channel process governed by exchange of resonances sitting on *Regge trajectory* $\alpha(t) = \alpha + \alpha' t$
- Resonance of mass m sitting on the trajectory at $t = m^2$ has angular momentum of value $\alpha(m^2)$
- Trajectory makes contribution to total inelastic cross section $\sigma_{tot}(s) \propto s^{\alpha-1}$

Regge theory - short overview

- Pomeron $\mathbb{P}$ represents special Regge trajectory & is a color singlet state of gluons & has vacuum quantum numbers
- Contribution to σ_{tot} grows with increasing energy
- Double Pomeron exchange = production mechanism to study glueballs in the mass region below 2 MeV
- Pomeron exchange leads to restriction of quantum numbers of X
  $\rightarrow J^{PC} = (\text{even})^{++}$

Regge theory - short overview

- Pomeron $\mathbb{P}$ represents special Regge trajectory & is a color singlet state of gluons & has vacuum quantum numbers
- Contribution to σ_{tot} grows with increasing energy
- Double Pomeron exchange = production mechanism to study glueballs in the mass region below 2 MeV
- Pomeron exchange leads to restriction of quantum numbers of X
  $\rightarrow J^{PC} = (\text{even})^{++}$

Regge theory - short overview

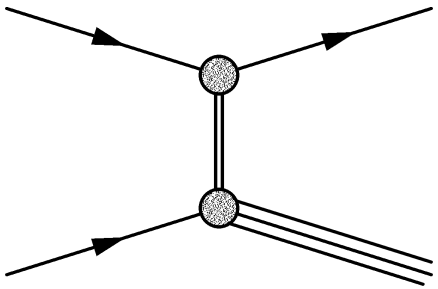
- Pomeron $\mathbb{P}$ represents special Regge trajectory & is a color singlet state of gluons & has vacuum quantum numbers
- Contribution to σ_{tot} grows with increasing energy
- Double Pomeron exchange = production mechanism to study glueballs in the mass region below 2 MeV
- Pomeron exchange leads to restriction of quantum numbers of X
  $\rightarrow J^{PC} = (\text{even})^{++}$

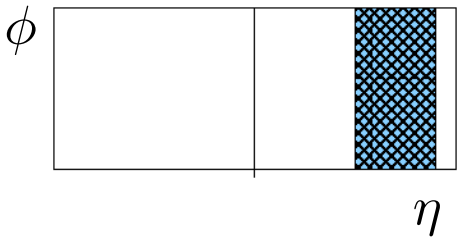
Regge theory - short overview

- Pomeron $\mathbb{P}$ represents special Regge trajectory & is a color singlet state of gluons & has vacuum quantum numbers
- Contribution to σ_{tot} grows with increasing energy
- Double Pomeron exchange = production mechanism to study glueballs in the mass region below 2 MeV
- Pomeron exchange leads to restriction of quantum numbers of X
  $\rightarrow J^{PC} = (\text{even})^{++}$

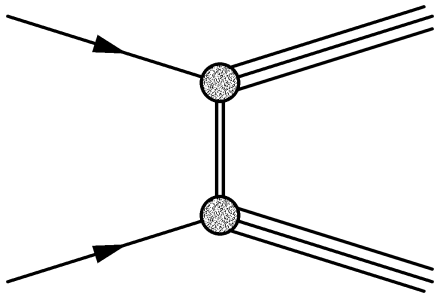
Event topology

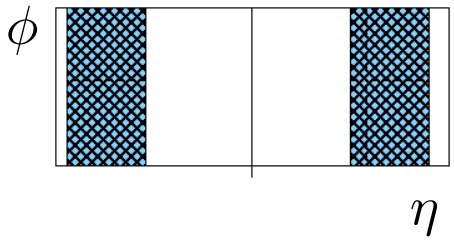
Single diffractive



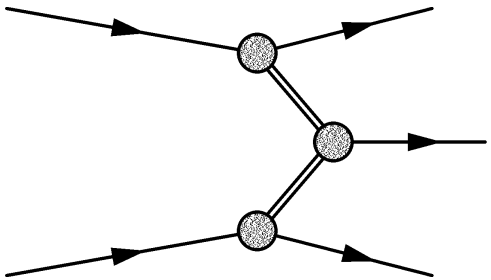


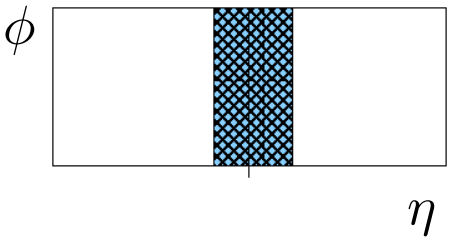
Double diffractive



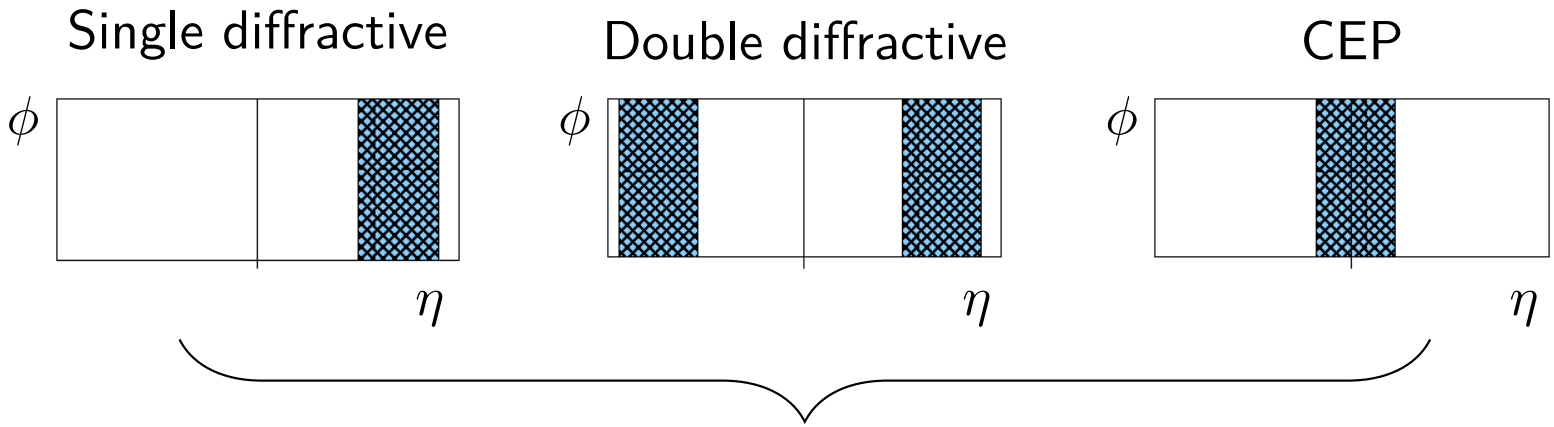


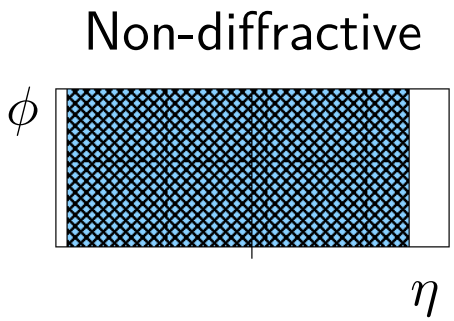
CEP





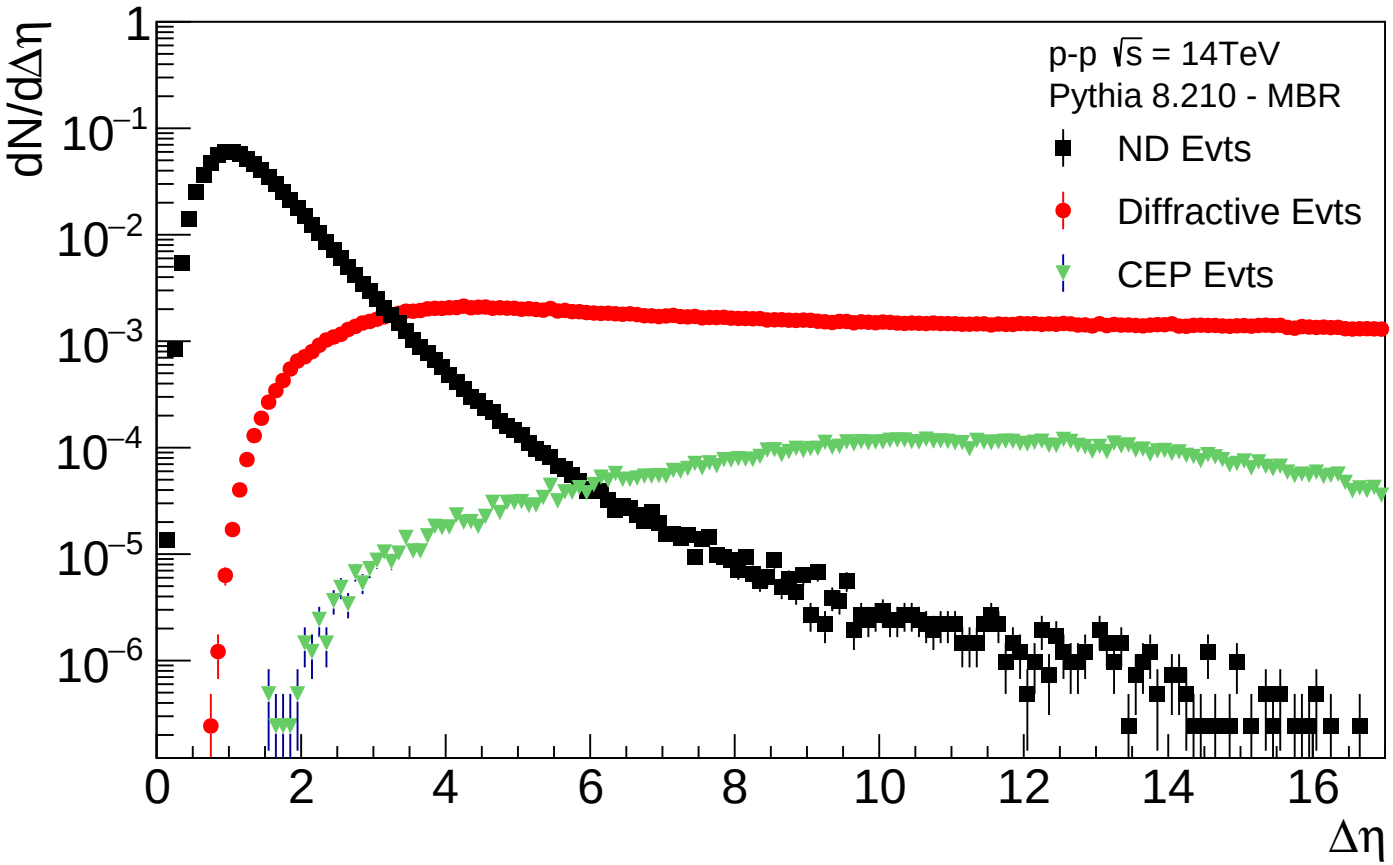
Event topology





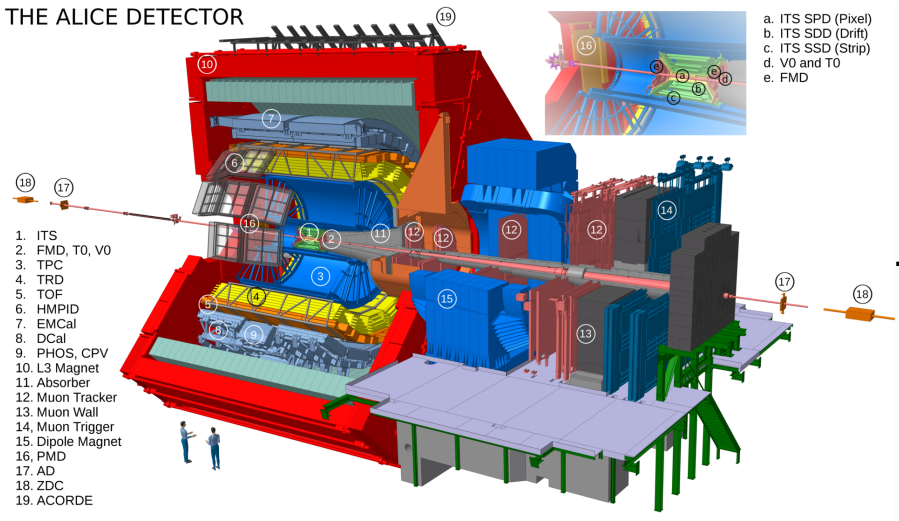
Event selection

Large rapidity gaps in *non-diffractive* events are exponentially supressed

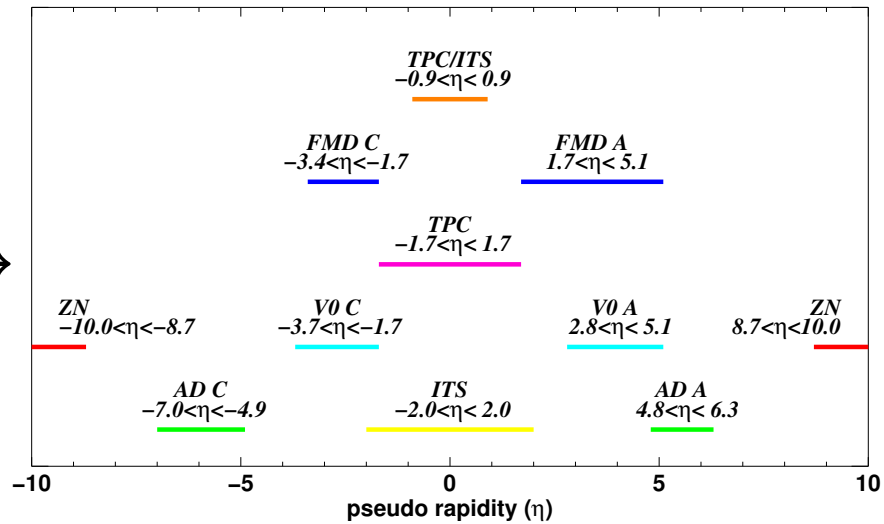


The ALICE detector system

ALICE detector

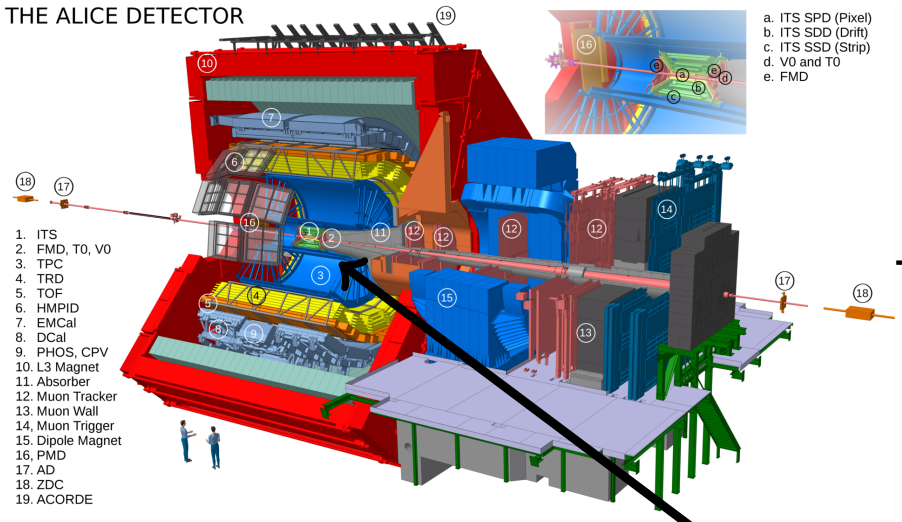


η coverage

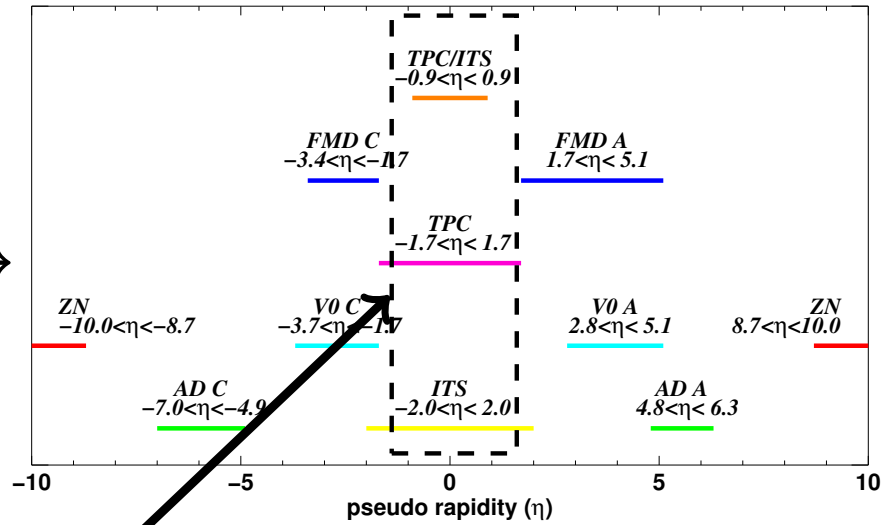


The ALICE detector system

ALICE detector



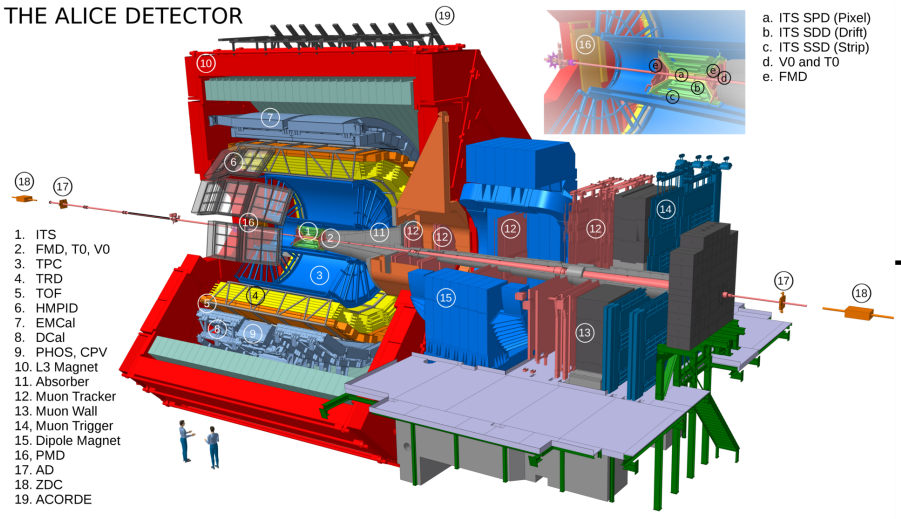
η coverage



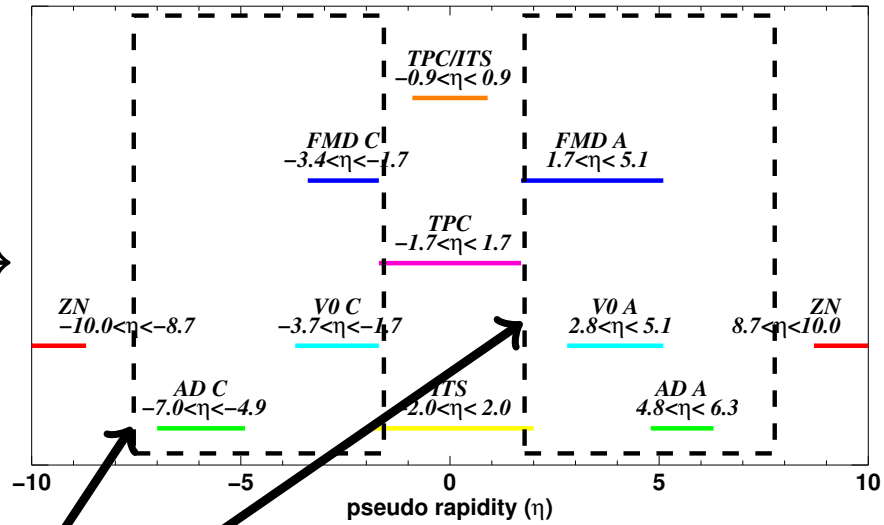
Central barrel $\rightarrow$ determine p^{μ}

The ALICE detector system

ALICE detector



η coverage



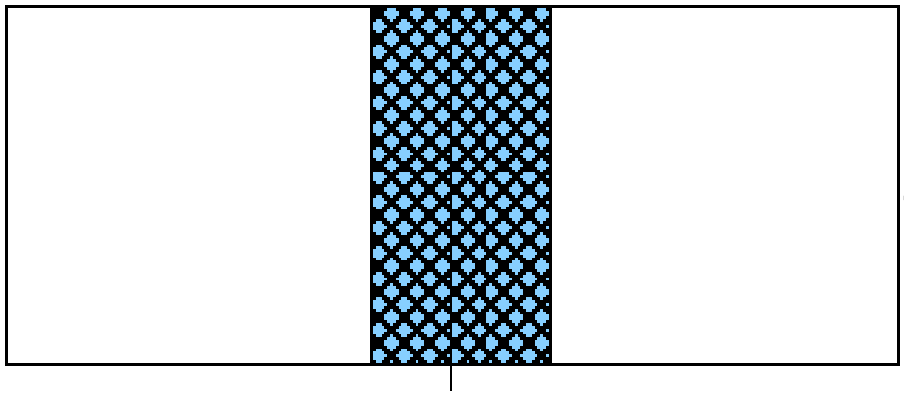
Small detectors for global event characterisation

Central exclusive production at ALICE

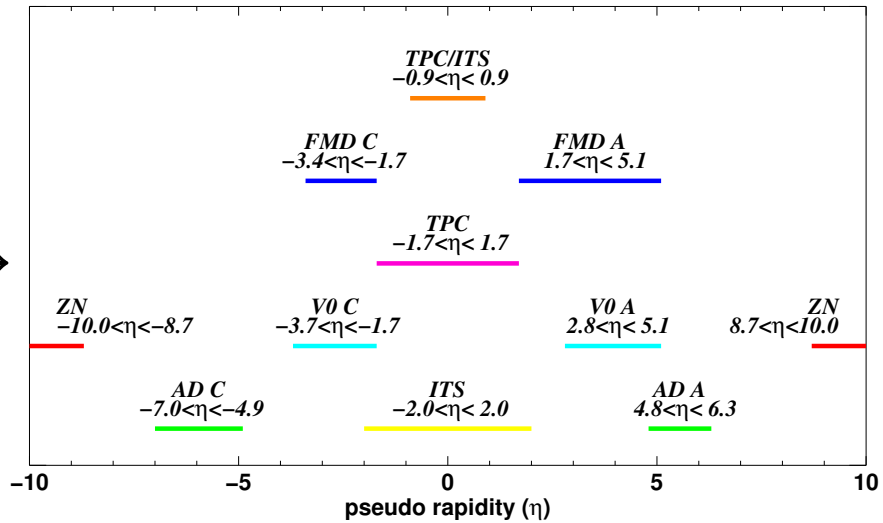
To study the CEP events a η gap condition is used

1. Signal in the central barrel
2. No activity in veto detectors

CEP event



η coverage

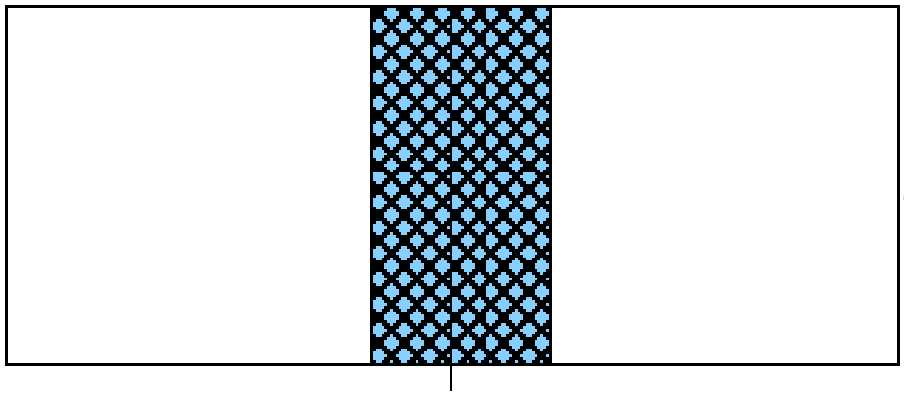


Central exclusive production at ALICE

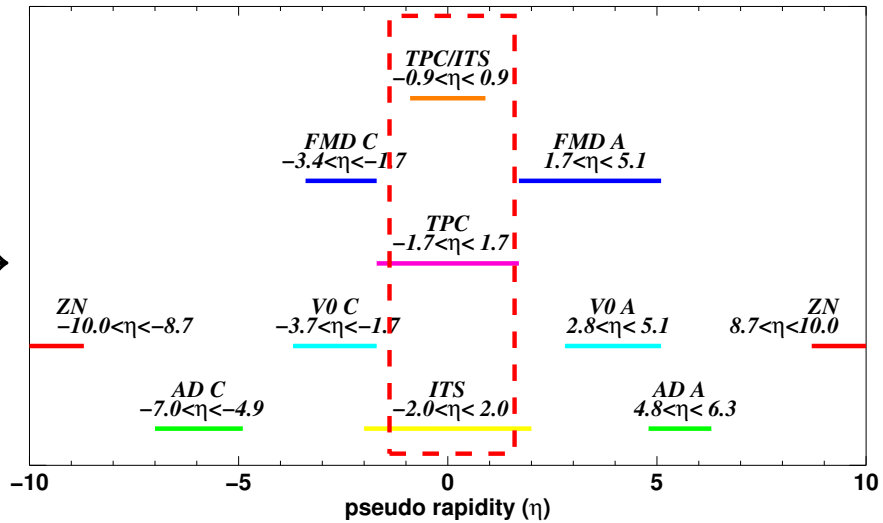
To study the CEP events a η gap condition is used

1. Signal in the central barrel
2. No activity in veto detectors

CEP event



η coverage

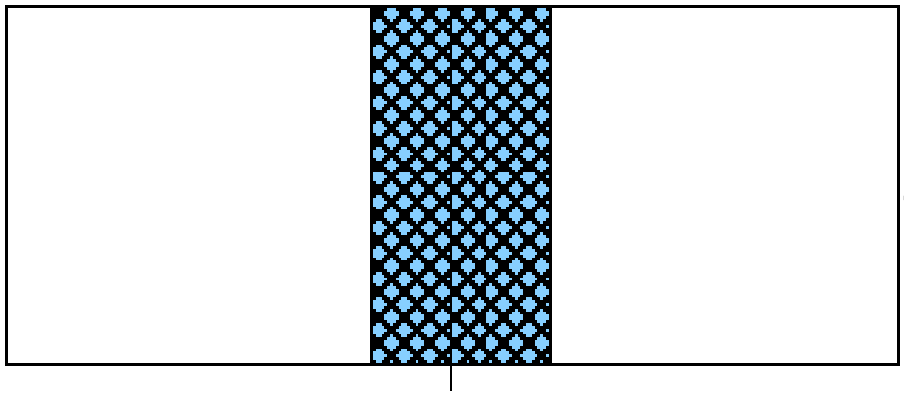


Central exclusive production at ALICE

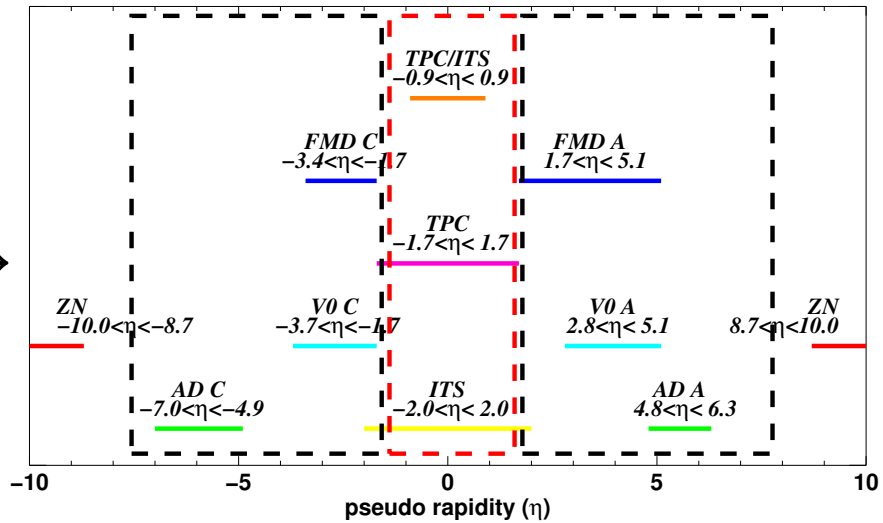
To study the CEP events a η gap condition is used

1. Signal in the central barrel
2. No activity in veto detectors

CEP event



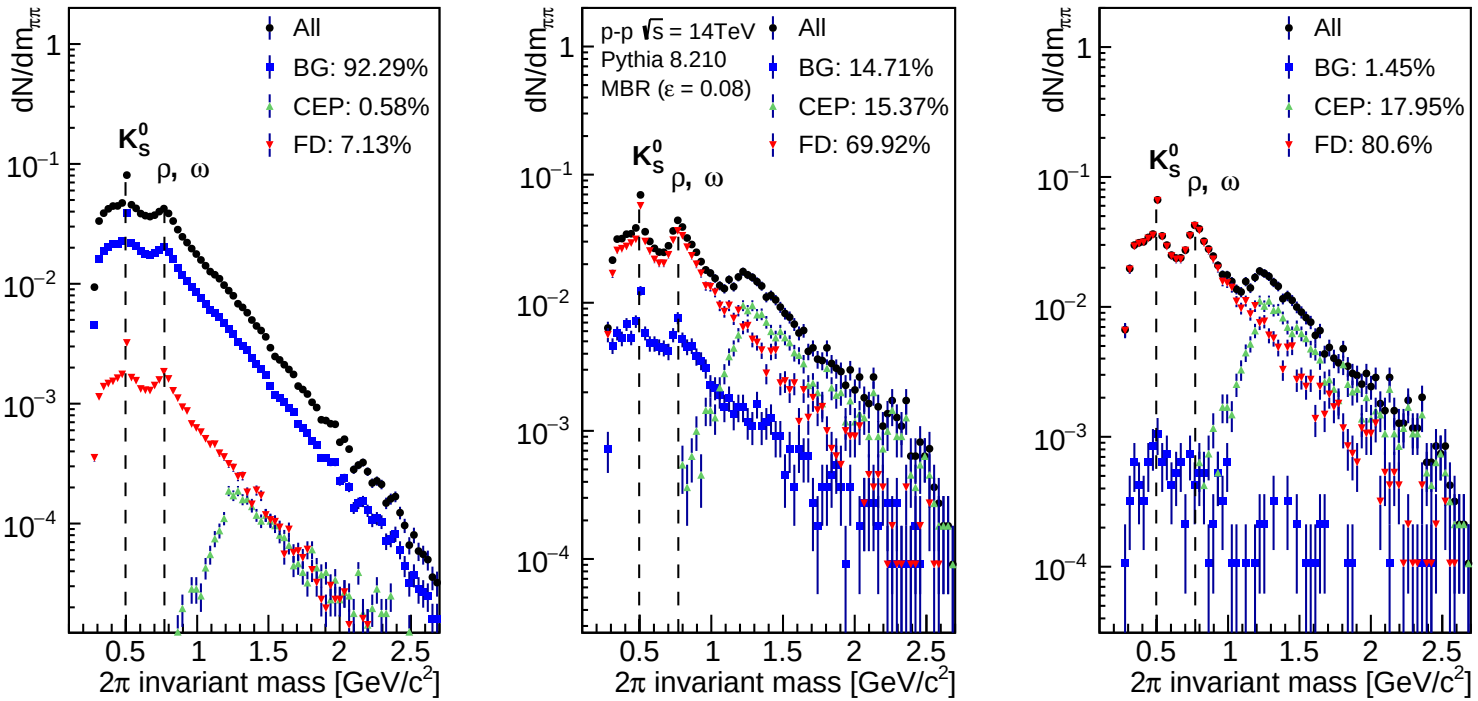
η coverage



Invariant mass spectrum

Studying *Pythia-8* simulations yield

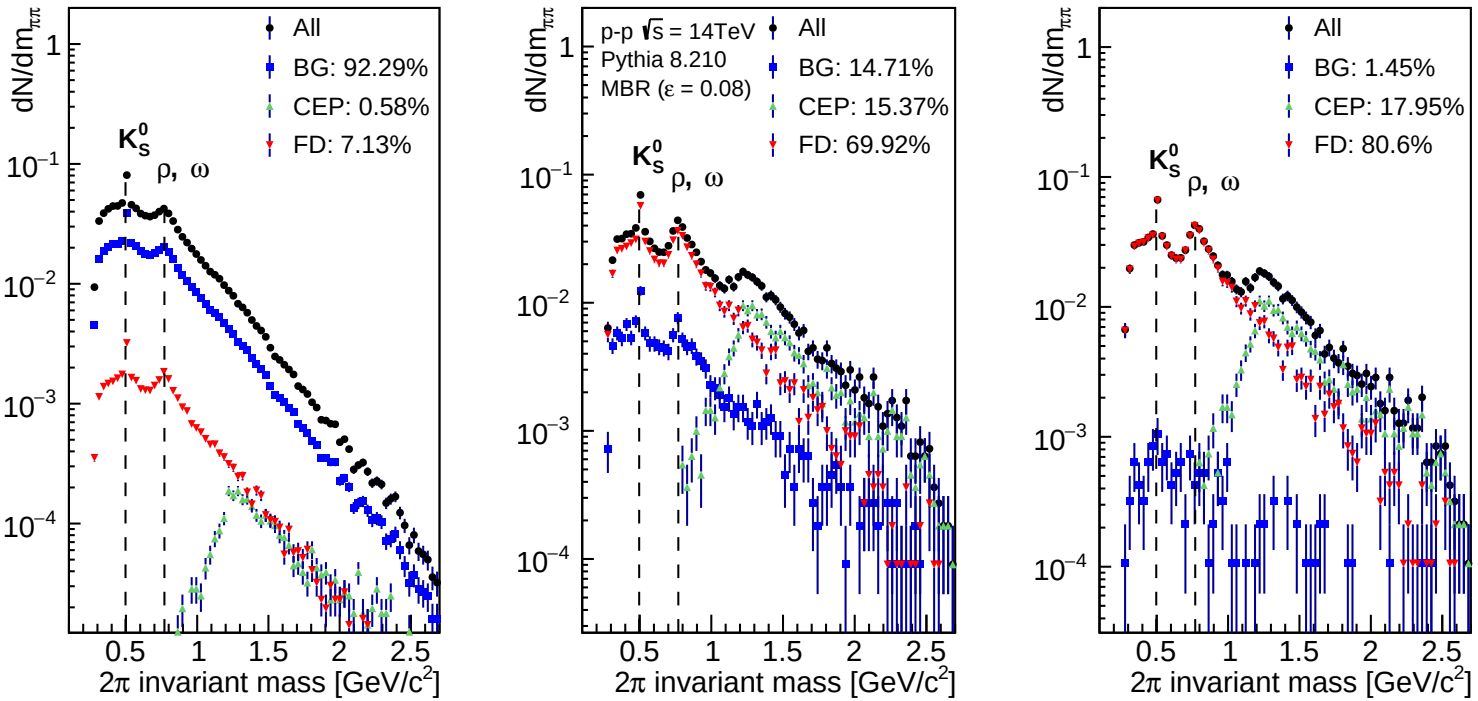
- Enforcing η gap cut reduces non-diffractive almost entirely
- Remaining background are partially recontructed CEP events - **feed down**



Invariant mass spectrum

Studying *Pythia-8* simulations yield

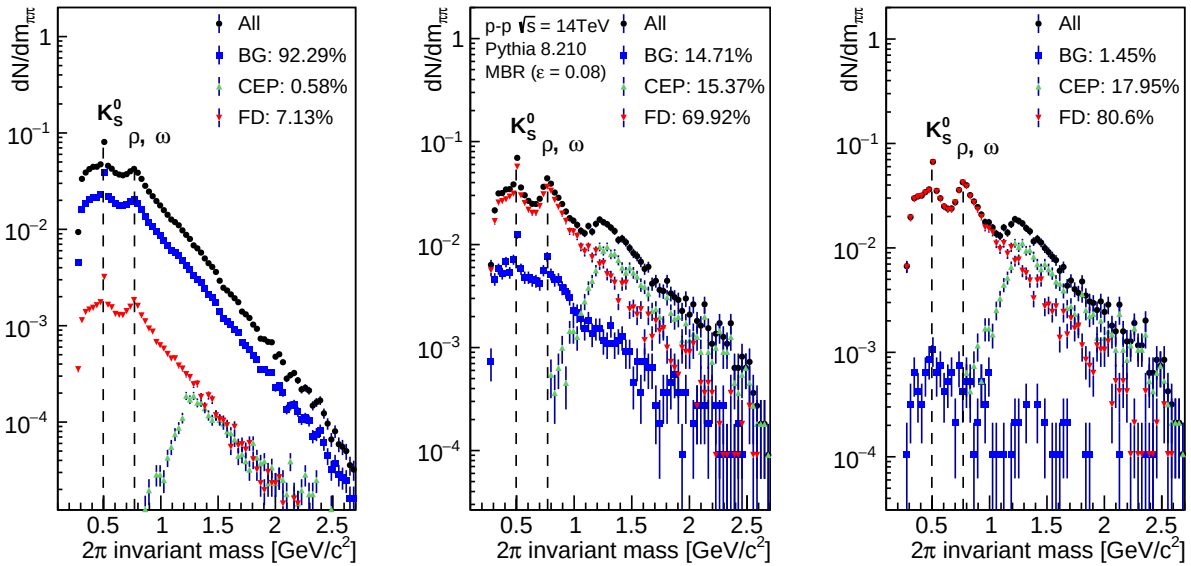
- Enforcing η gap cut reduces non-diffractive almost entirely
- Remaining background are partially recontructed CEP events - **feed down**



Invariant mass spectrum

Studying *Pythia-8* simulations yield

- Enforcing η gap cut reduces non-diffractive almost entirely
- Remaining background are partially recontructed CEP events - **feed down**



ML: an overview

In general ML represents a contrast to a *rule based systems*

Rule-based system

System that uses rules to make deductions or choices

- Domain-specific expert system
- Knowledge base: facts & rules (if $\rightarrow$ then statement)
- Rules manually specified (by expert) $\rightarrow$ expensive, incomplete

ML: an overview

In general ML represents a contrast to a *rule based systems*

Rule-based system

System that uses rules to make deductions or choices

- Domain-specific expert system
- Knowledge base: facts & rules (if $\rightarrow$ then statement)
- Rules manually specified (by expert) $\rightarrow$ expensive, incomplete

ML: an overview

In general ML represents a contrast to a *rule based systems*

Rule-based system

System that uses rules to make deductions or choices

- Domain-specific expert system
- Knowledge base: facts & rules (if $\rightarrow$ then statement)
- Rules manually specified (by expert) $\rightarrow$ expensive, incomplete

ML: an overview

In general ML represents a contrast to a *rule based systems*

Machine learning

- Alorithms that learn from *data* & make predictions on *data*
- Automatic methods $\rightarrow$ no human needed
- Human work required for defining problem & assessing the data

ML: an overview

In general ML represents a contrast to a *rule based systems*

Machine learning

- Alorithms that learn from *data* & make predictions on *data*
- Automatic methods $\rightarrow$ no human needed
- Human work required for defining problem & assessing the data

ML: an overview

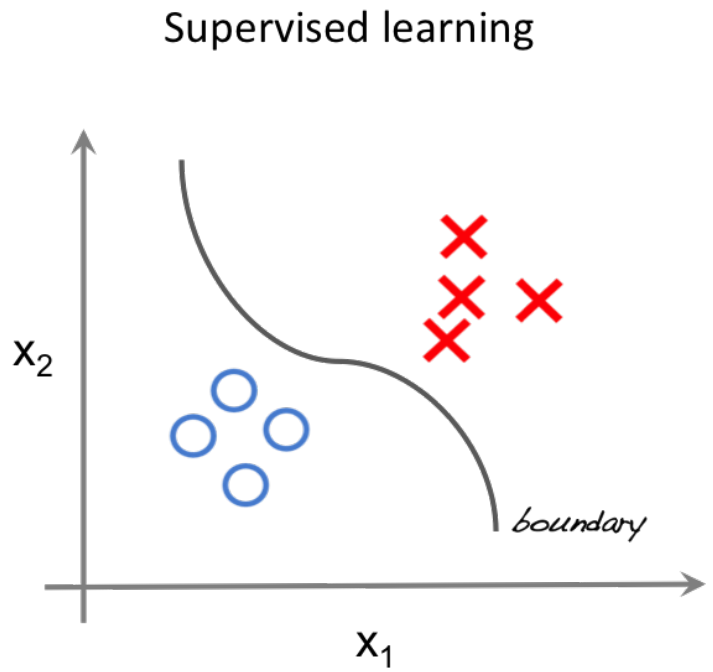
In general ML represents a contrast to a *rule based systems*

Machine learning

- Alorithms that learn from *data* & make predictions on *data*
- Automatic methods $\rightarrow$ no human needed
- Human work required for defining problem & assessing the data

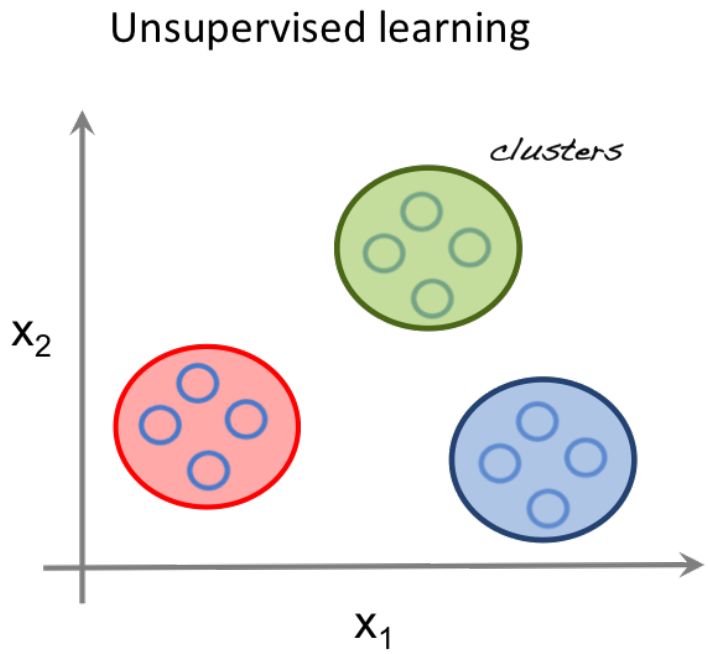
Types of ML

- Supervised
  - Classification
  - Regression
- Unsupervised



Types of ML

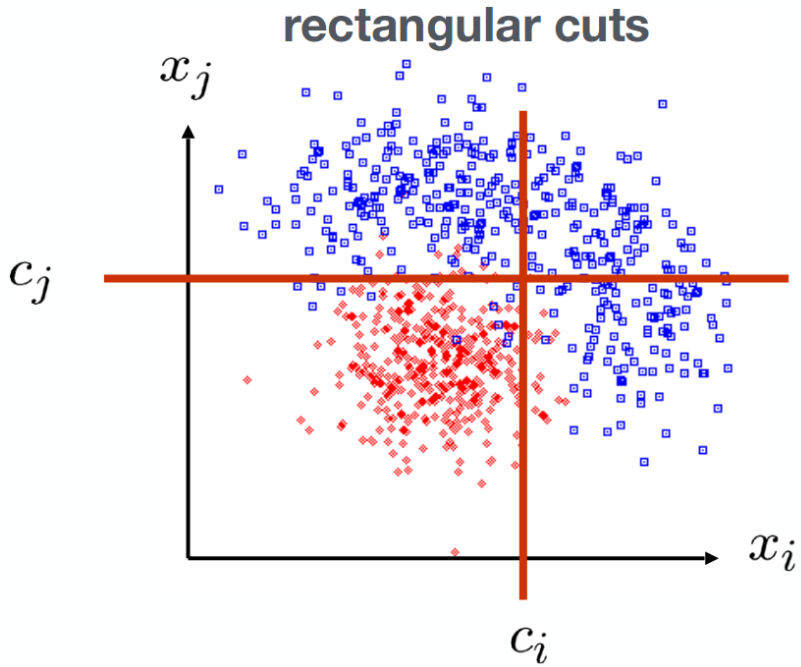
- Supervised
  - Classification
  - Regression
- Unsupervised



Rectangular cuts

Standard cut in one variable

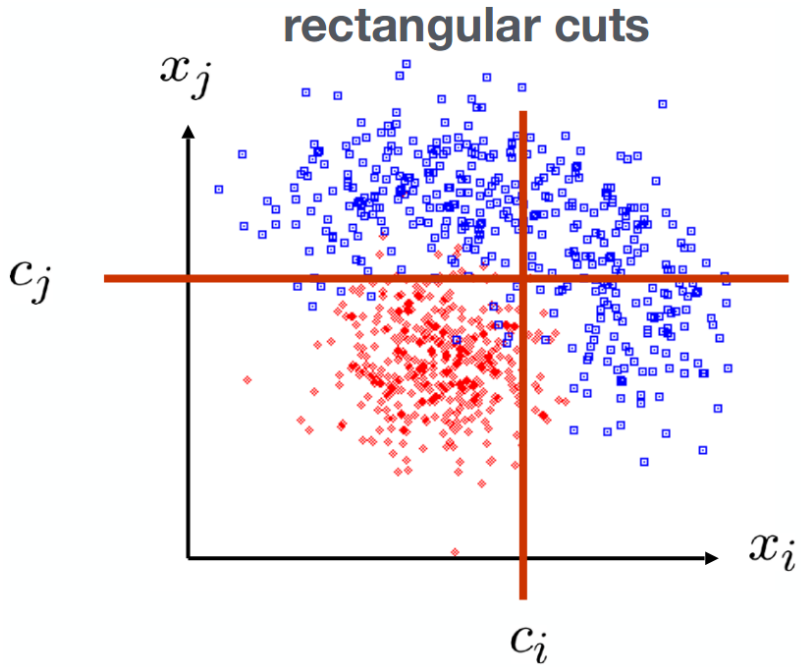
- Cuts only in lower-dimensional subspaces
- Ignores possible dependencies between the input variables
- Signal might behave like BG in several observables
  $\rightarrow$ misclassification



Rectangular cuts

Standard cut in one variable

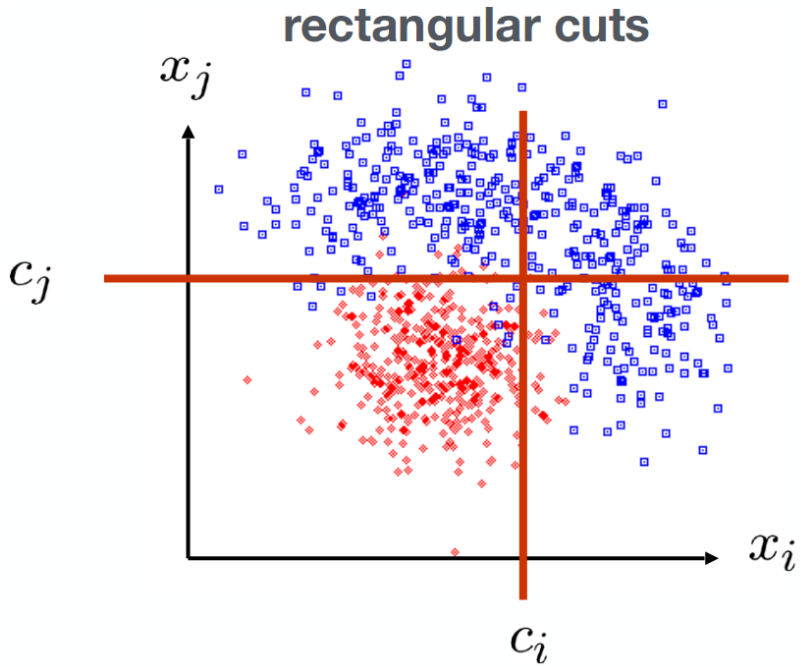
- Cuts only in lower-dimensional subspaces
- Ignores possible dependencies between the input variables
- Signal might behave like BG in several observables
  $\rightarrow$ misclassification



Rectangular cuts

Standard cut in one variable

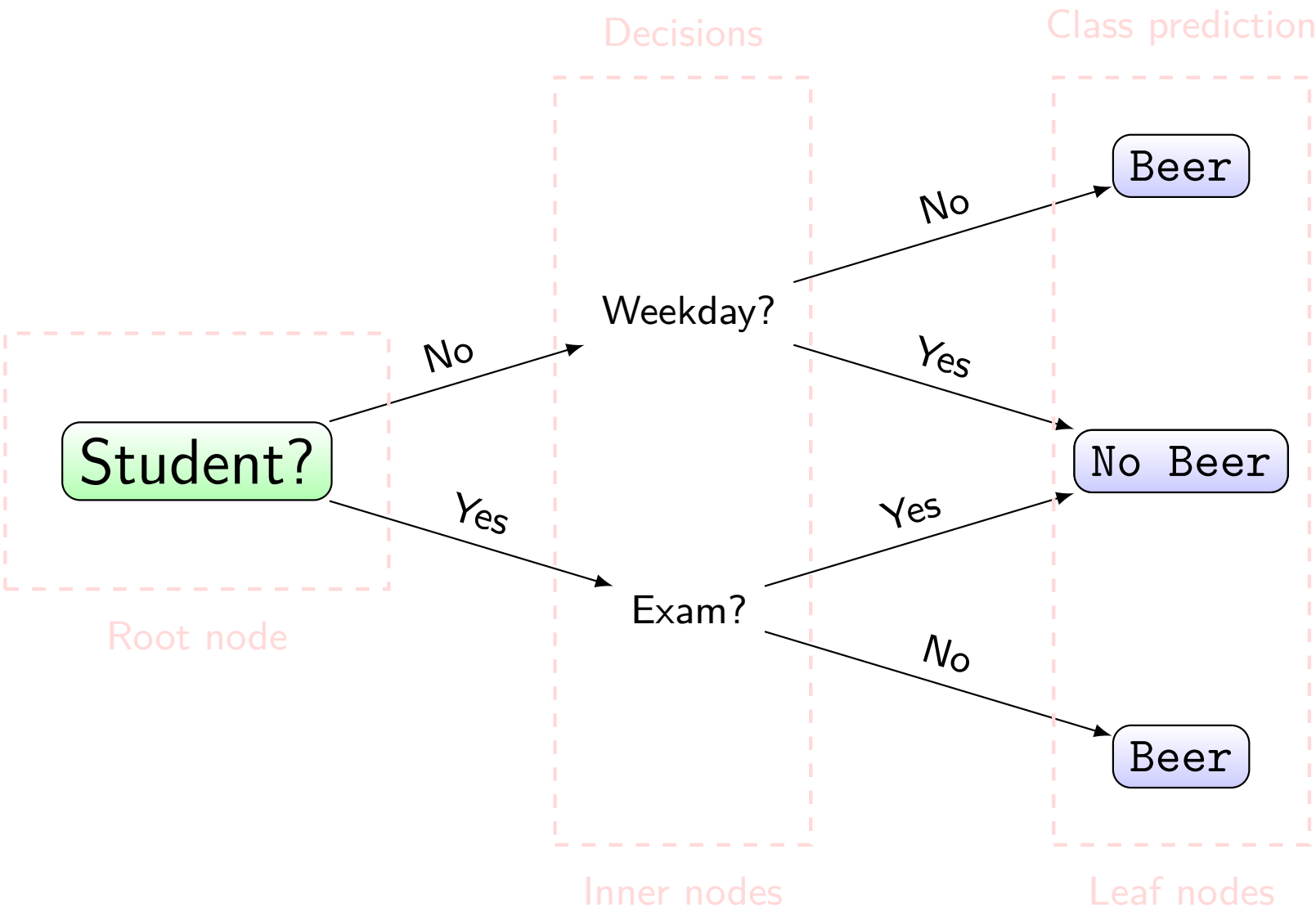
- Cuts only in lower-dimensional subspaces
- Ignores possible dependencies between the input variables
- Signal might behave like BG in several observables
  $\rightarrow$ misclassification



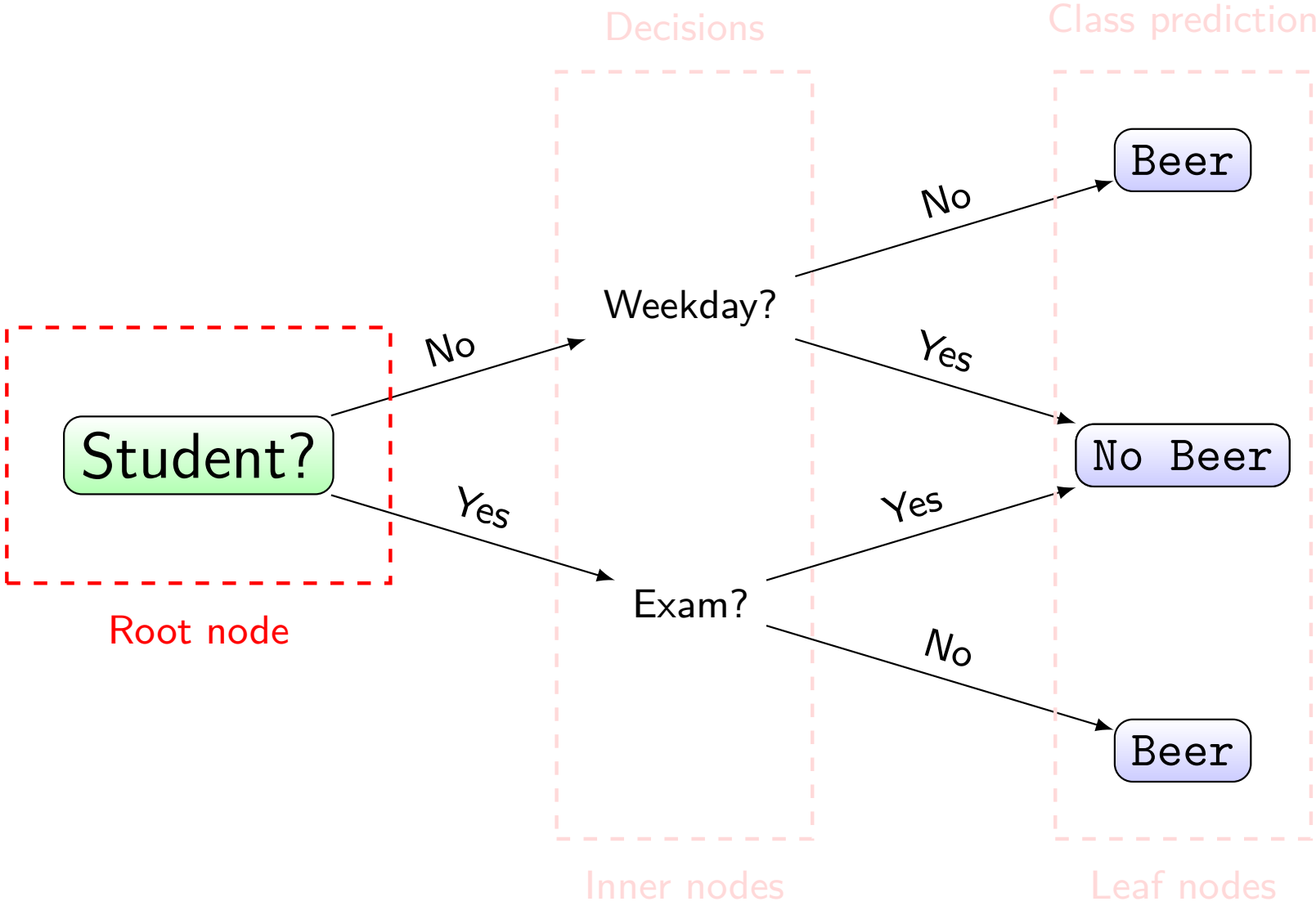
Rectangular cuts with *decision trees*

- Tree-like graph $\rightarrow$ flowchart
- Easy to understand
- Either be manually modelled by experts or learned from training data

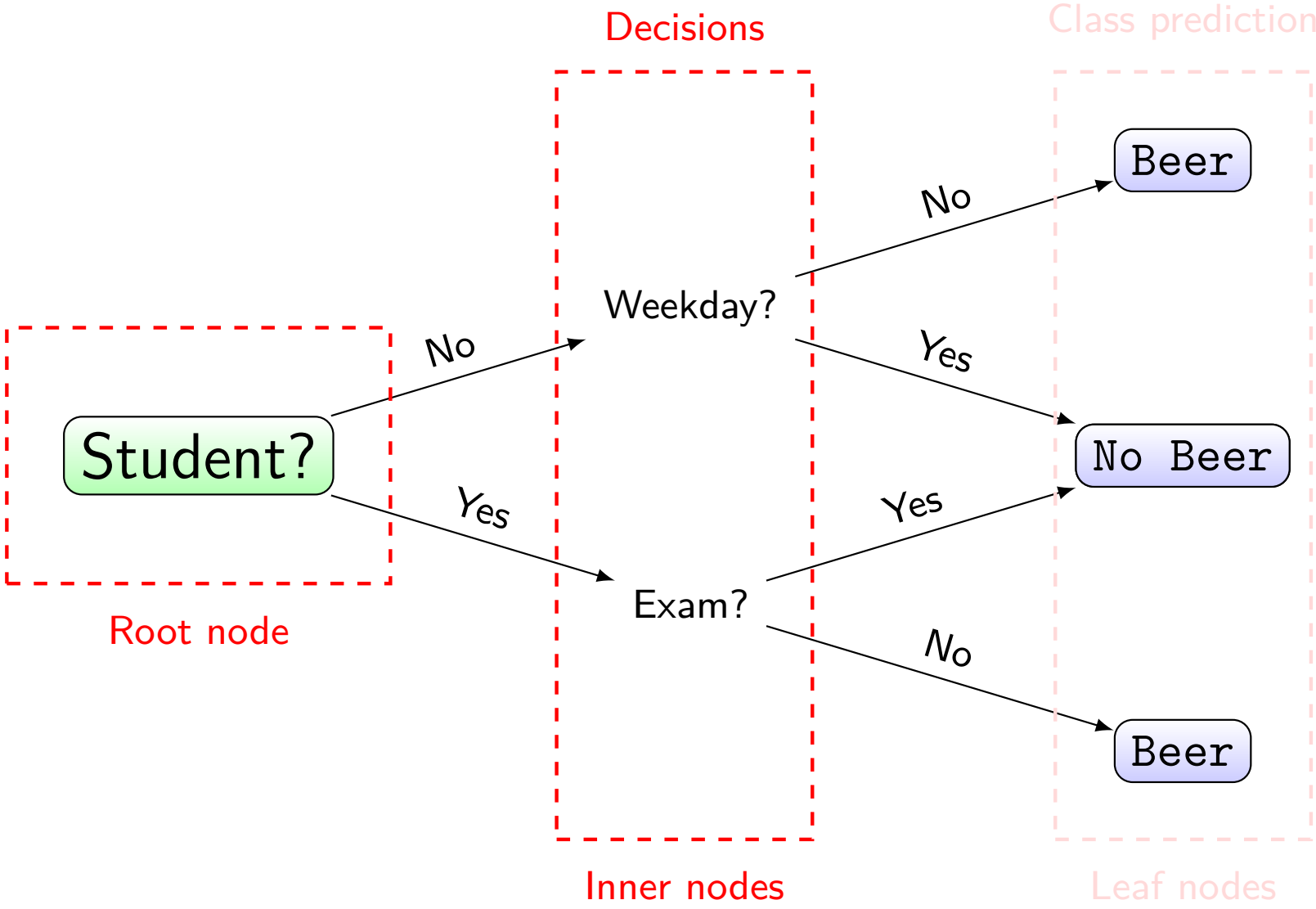
Rectangular cuts with *decision trees*



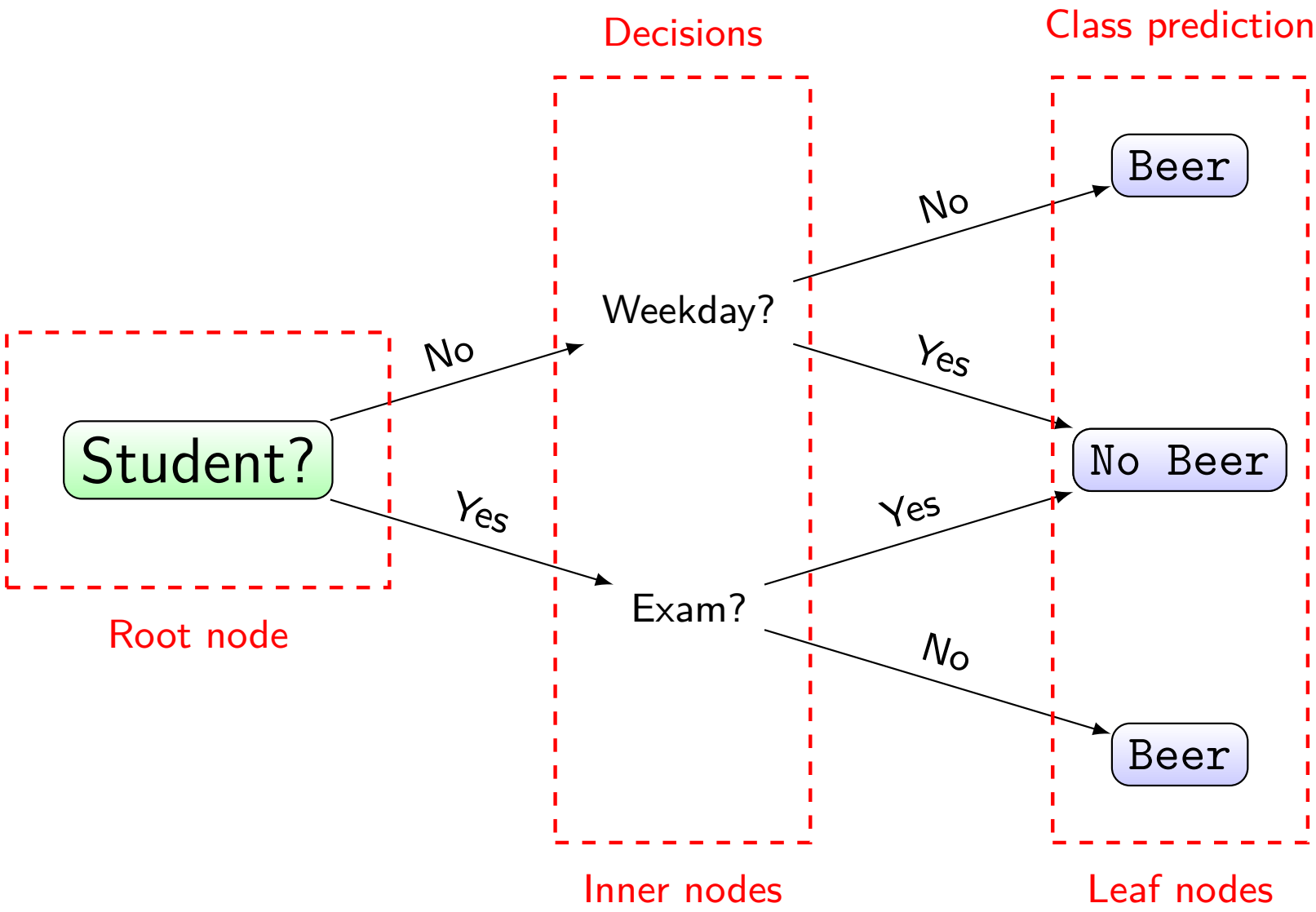
Rectangular cuts with *decision trees*



Rectangular cuts with *decision trees*



Rectangular cuts with *decision trees*



Decision tree learning

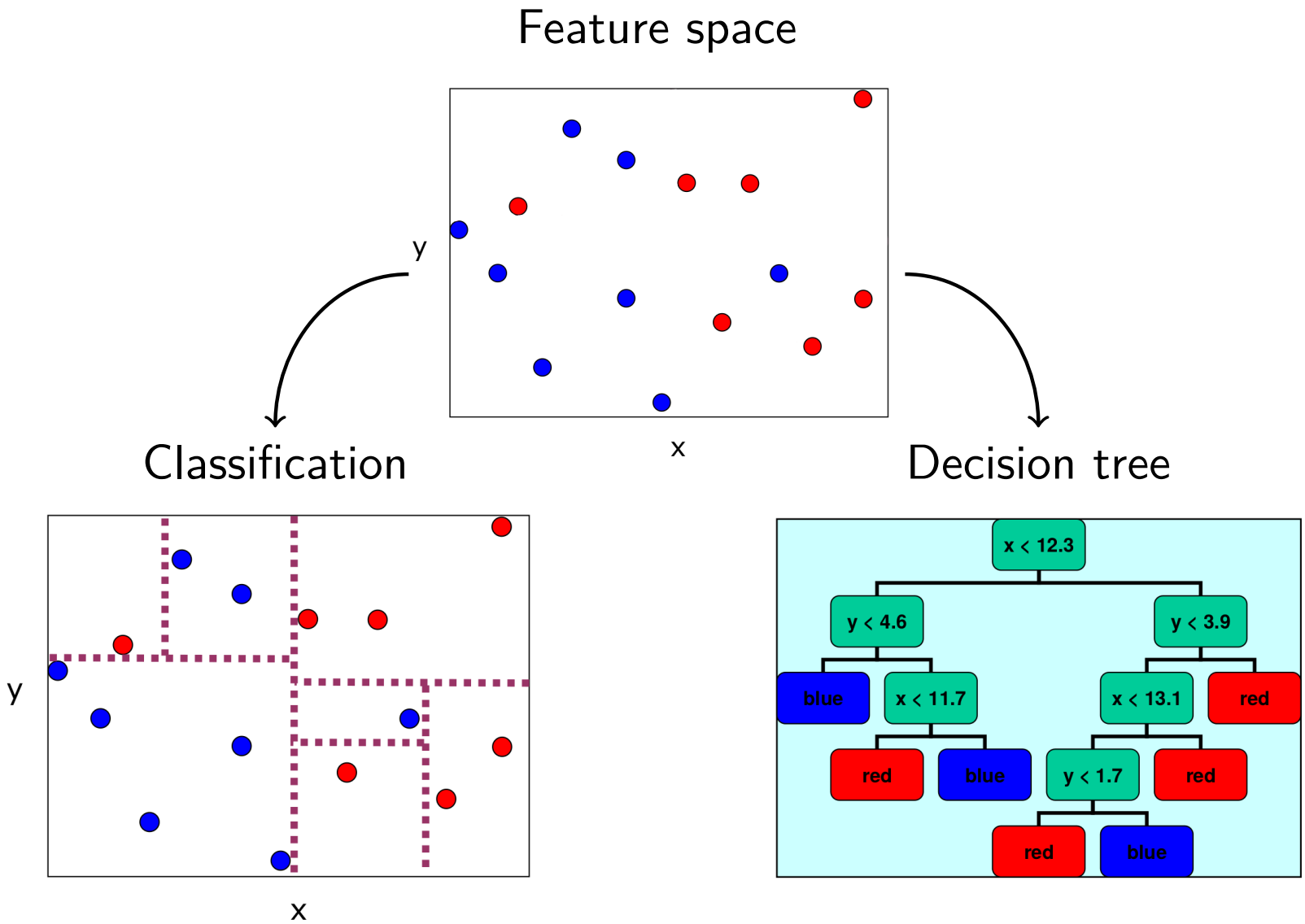
Training

Recursively split feature space into sub-spaces at each step

$\rightarrow$ Measures to evaluate split

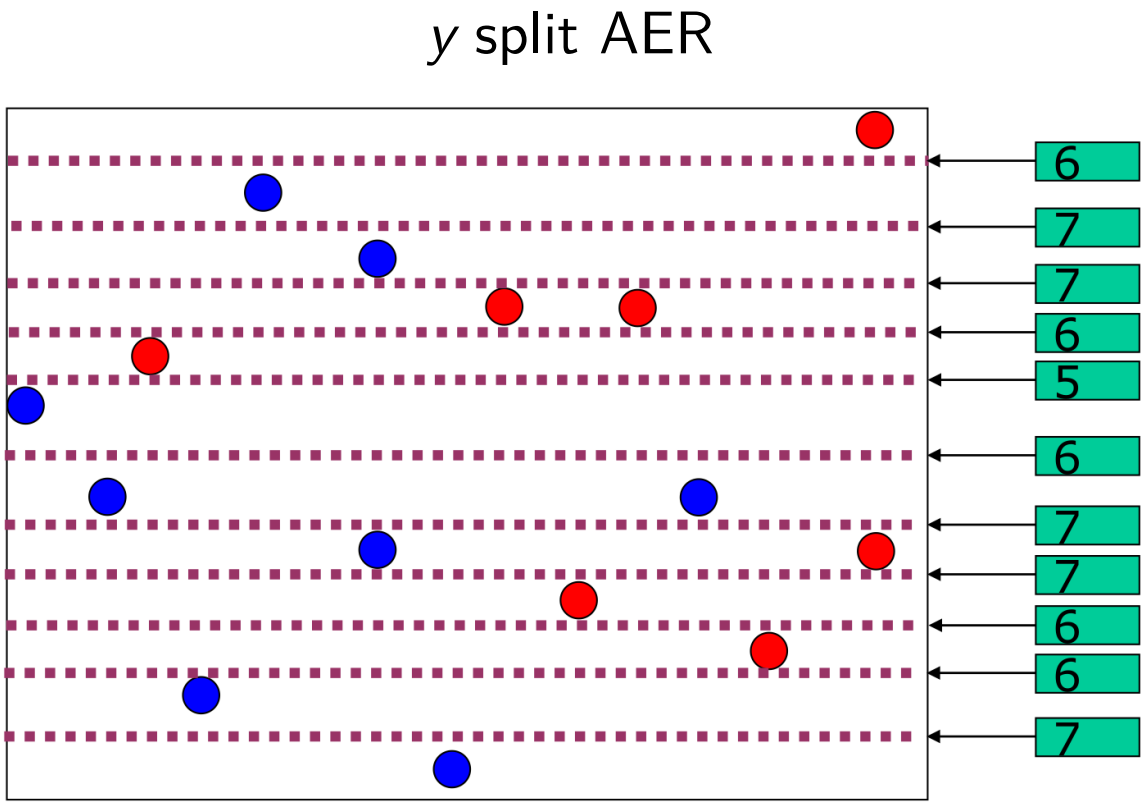
- Error rate
- Information gain
- Gini index

Decision tree learning



Decision tree learning

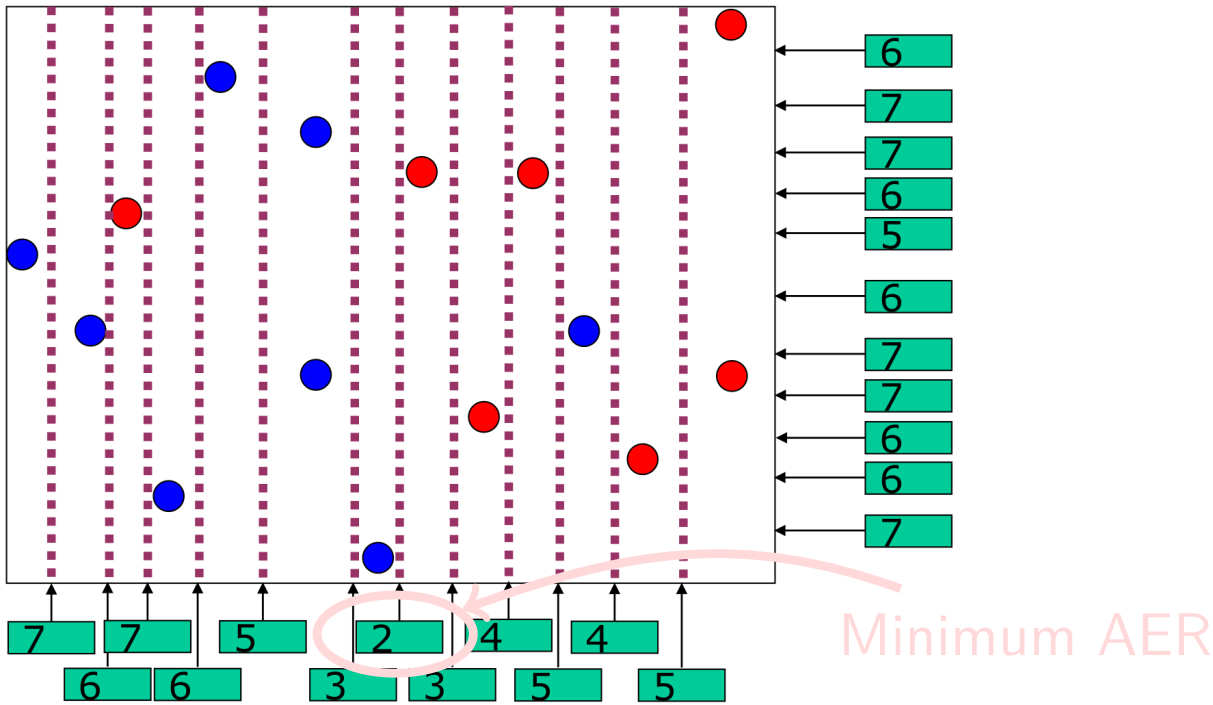
1) We compute a measure for *each possible split* in each feature
   $\rightarrow$ here **absolute error rate** (AER)



Decision tree learning

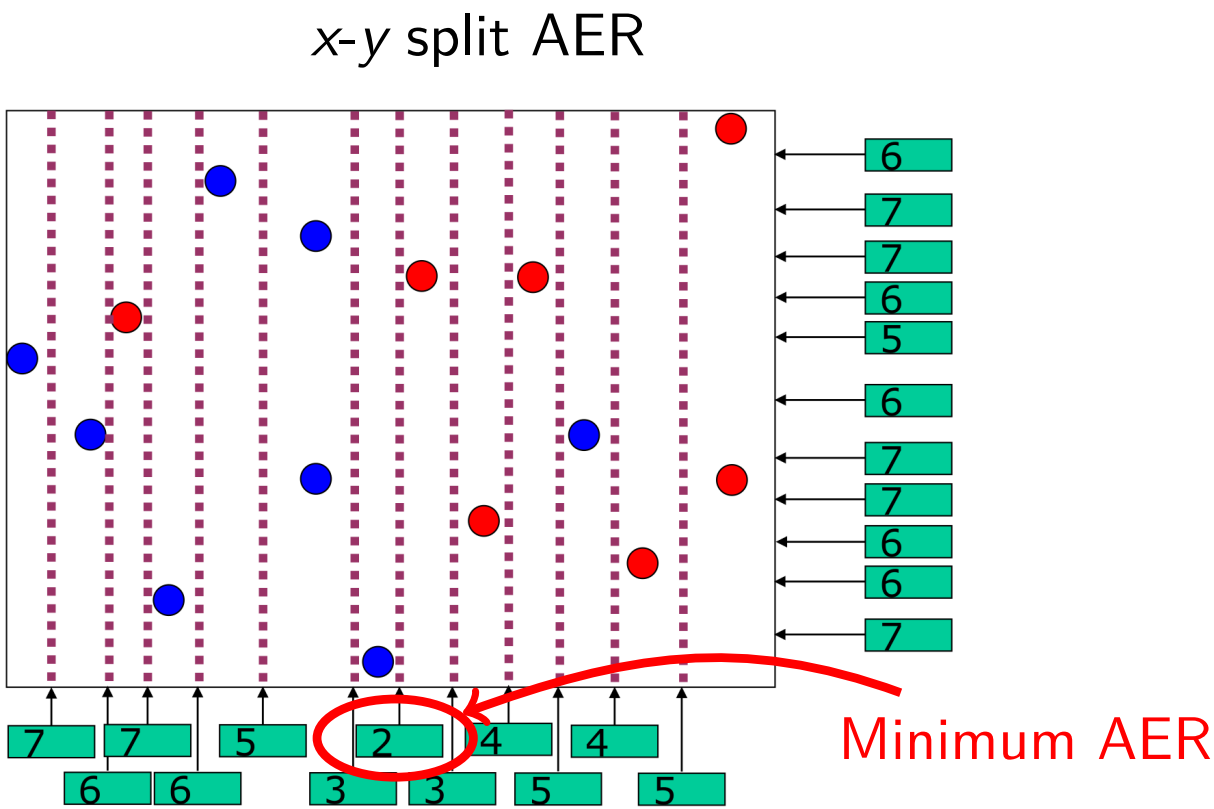
1) We compute a measure for *each possible split* in each feature
   $\rightarrow$ here **absolute error rate** (AER)

x-*y* split AER



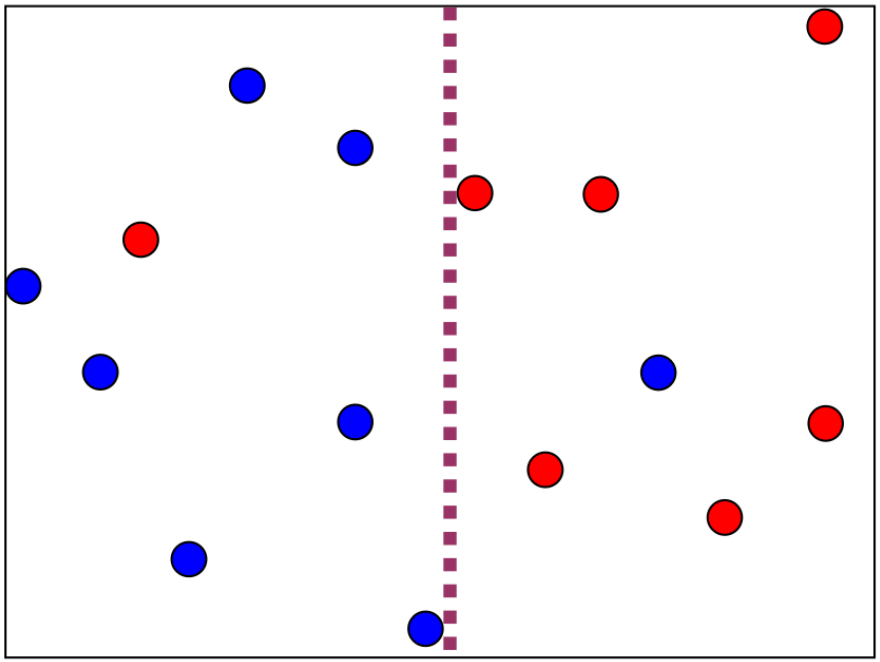
Decision tree learning

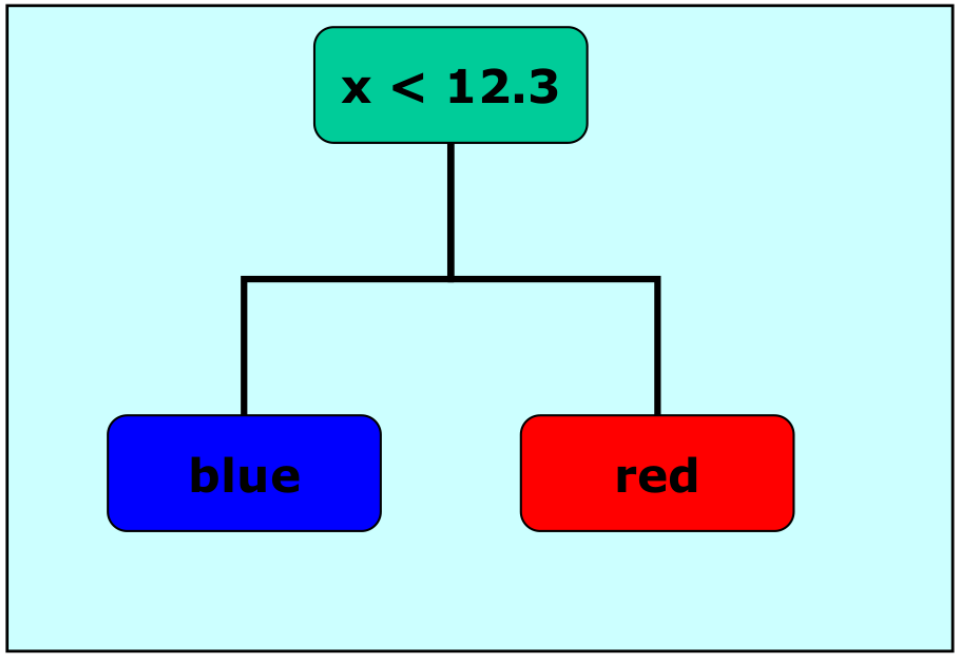
1) We compute a measure for *each possible split* in each feature
   $\rightarrow$ here **absolute error rate** (AER)



Decision tree learning

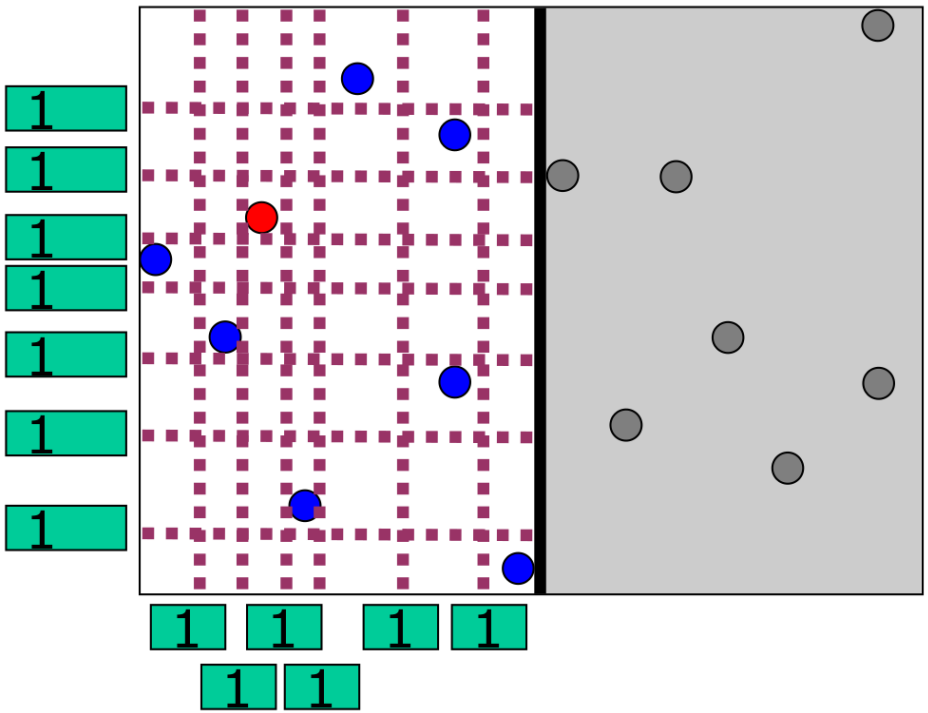
1) We compute a measure for *each possible split* in each feature
   $\rightarrow$ here **absolute error rate** (AER)
2) Recursively repead step (1) for each subspace until AER $\rightarrow 0$

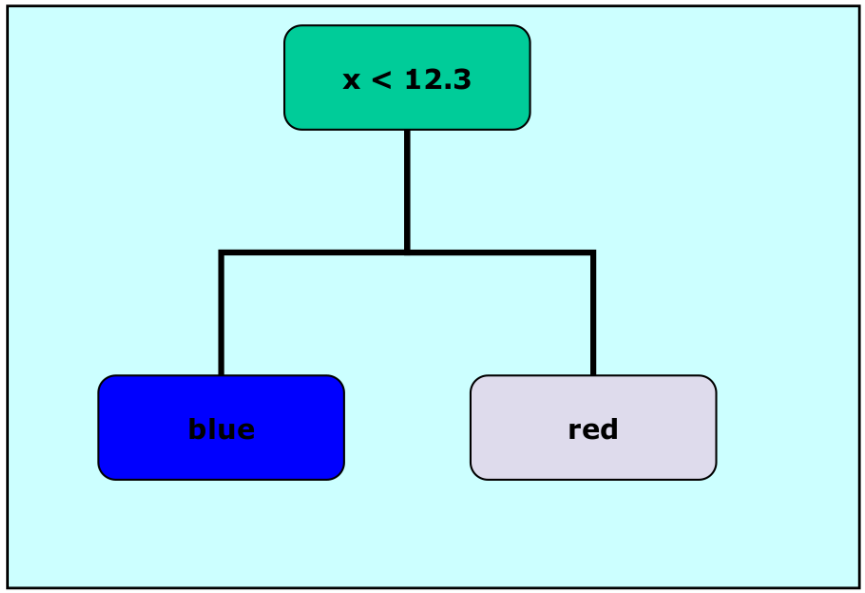




Decision tree learning

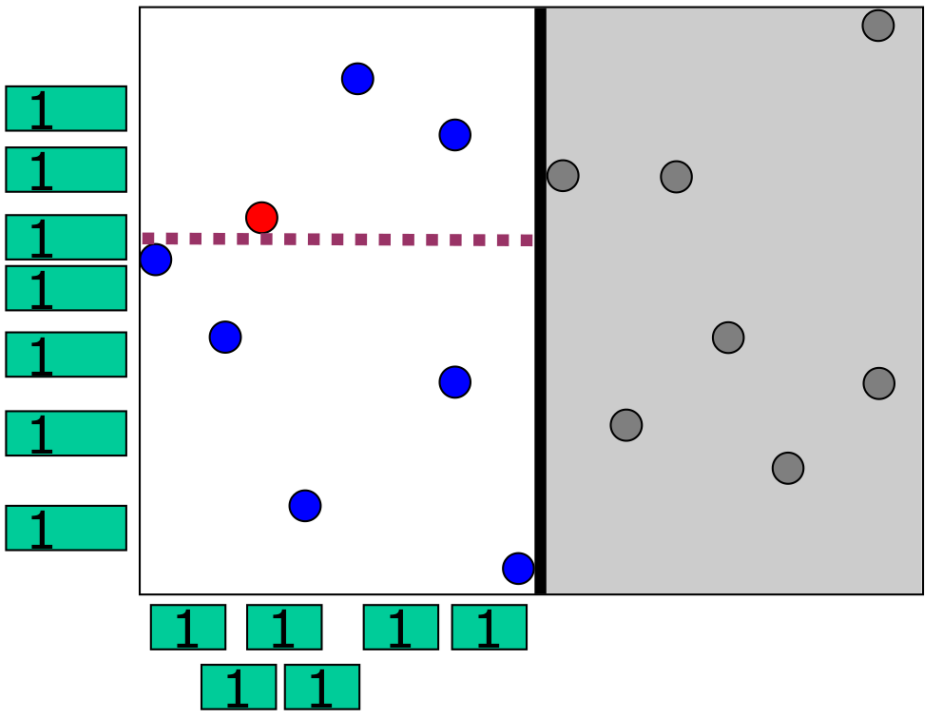
1) We compute a measure for *each possible split* in each feature
   $\rightarrow$ here **absolute error rate** (AER)
2) Recursively repead step (1) for each subspace until AER $\rightarrow 0$

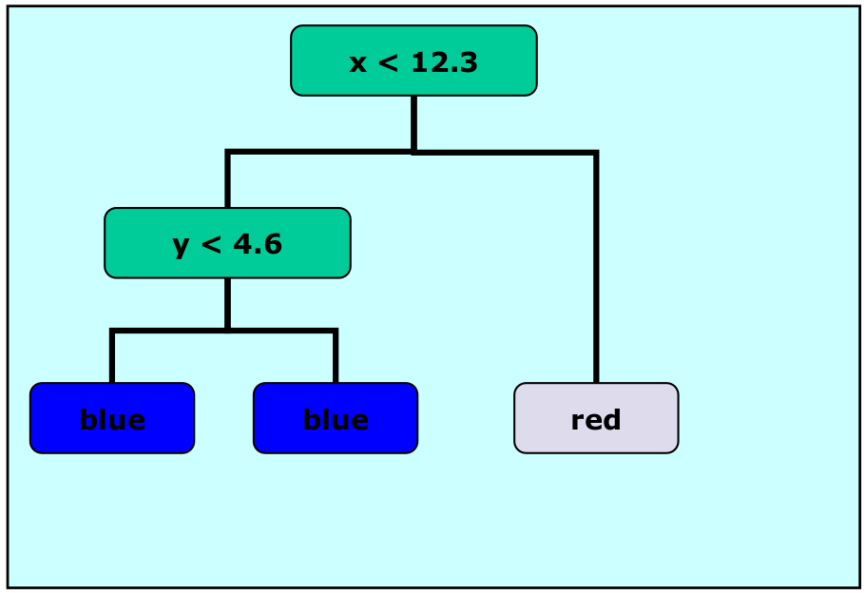




Decision tree learning

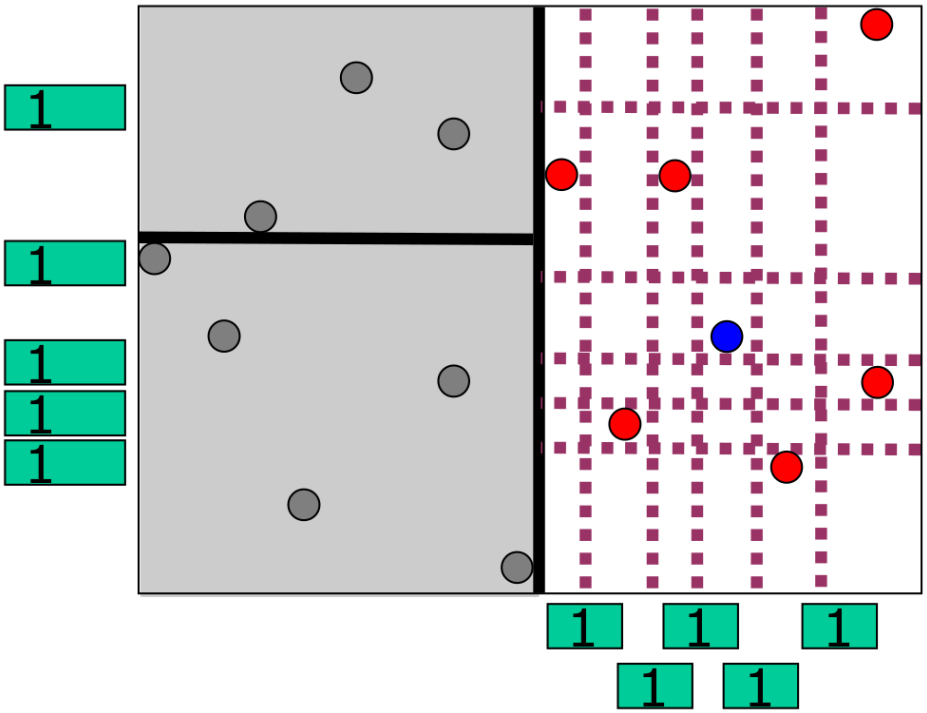
1) We compute a measure for *each possible split* in each feature
   $\rightarrow$ here **absolute error rate** (AER)
2) Recursively repead step (1) for each subspace until AER $\rightarrow 0$

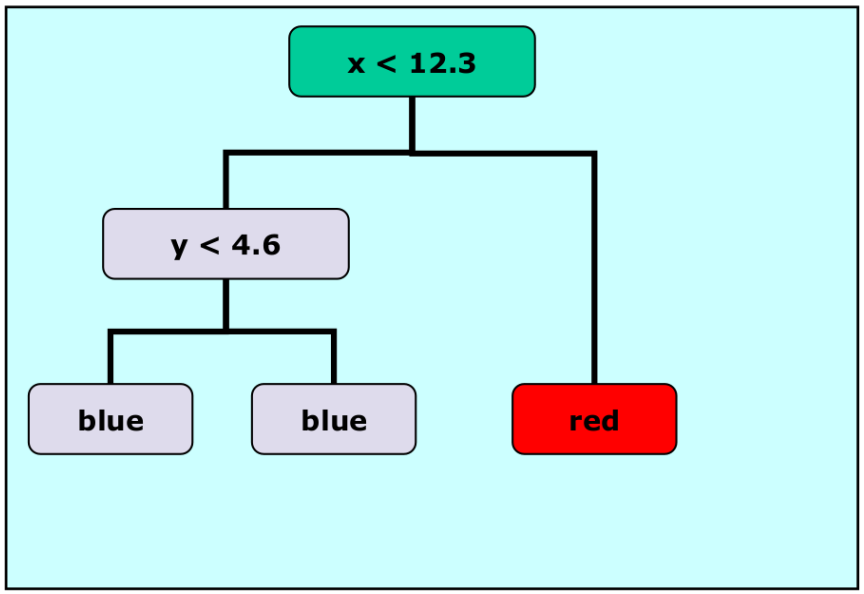




Decision tree learning

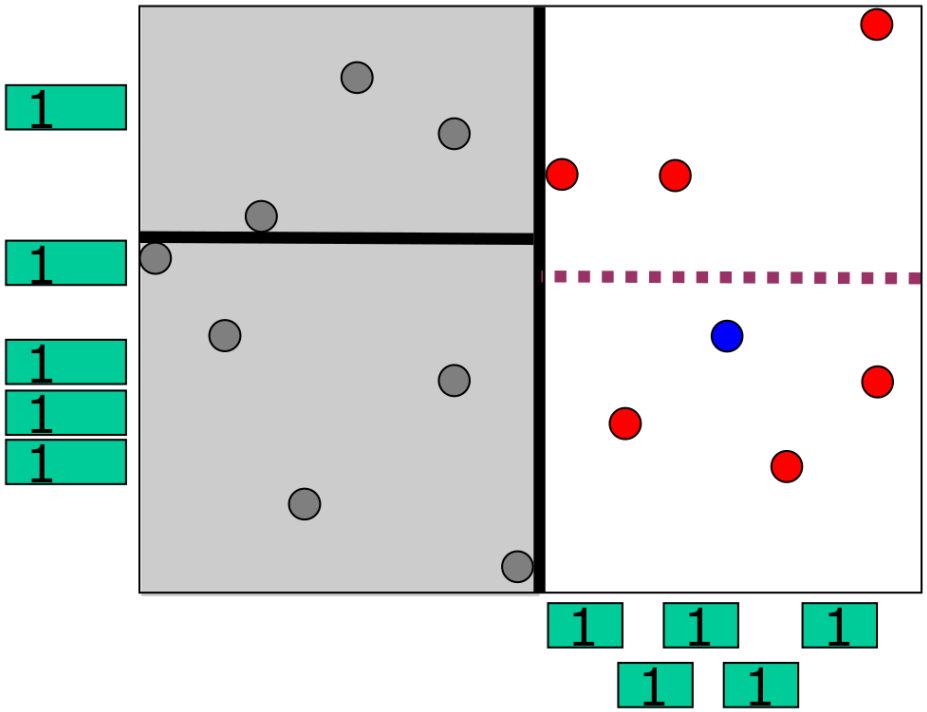
1) We compute a measure for *each possible split* in each feature
   $\rightarrow$ here **absolute error rate** (AER)
2) Recursively repead step (1) for each subspace until AER $\rightarrow 0$

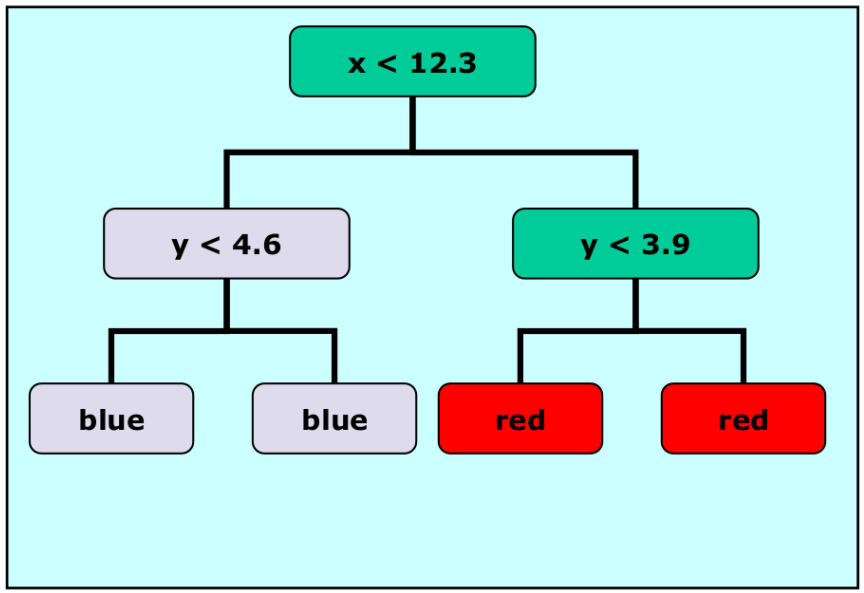




Decision tree learning

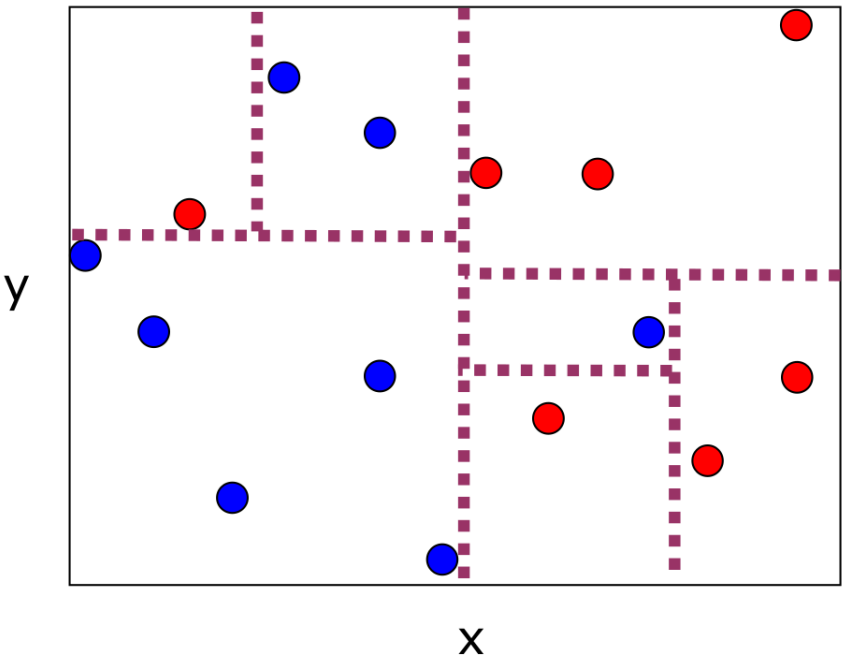
1) We compute a measure for *each possible split* in each feature
   $\rightarrow$ here **absolute error rate** (AER)
2) Recursively repead step (1) for each subspace until AER $\rightarrow 0$

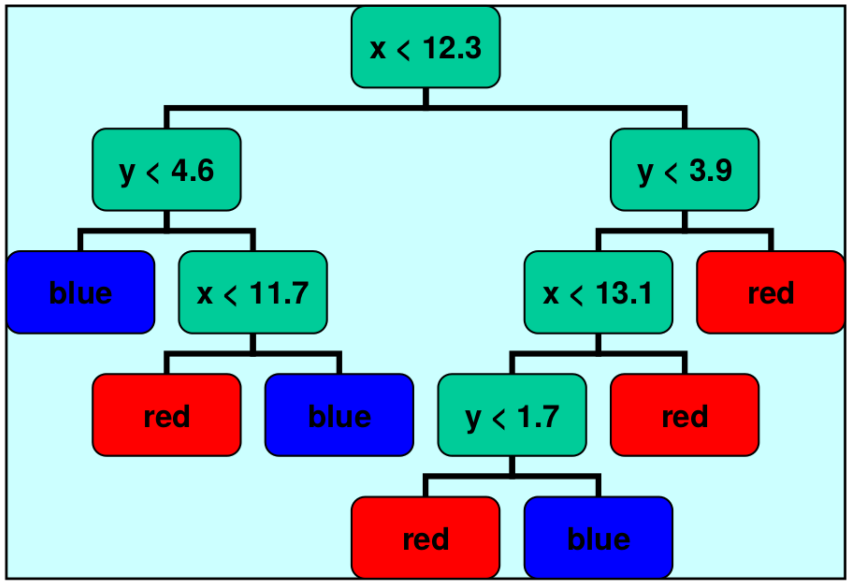




Decision tree learning

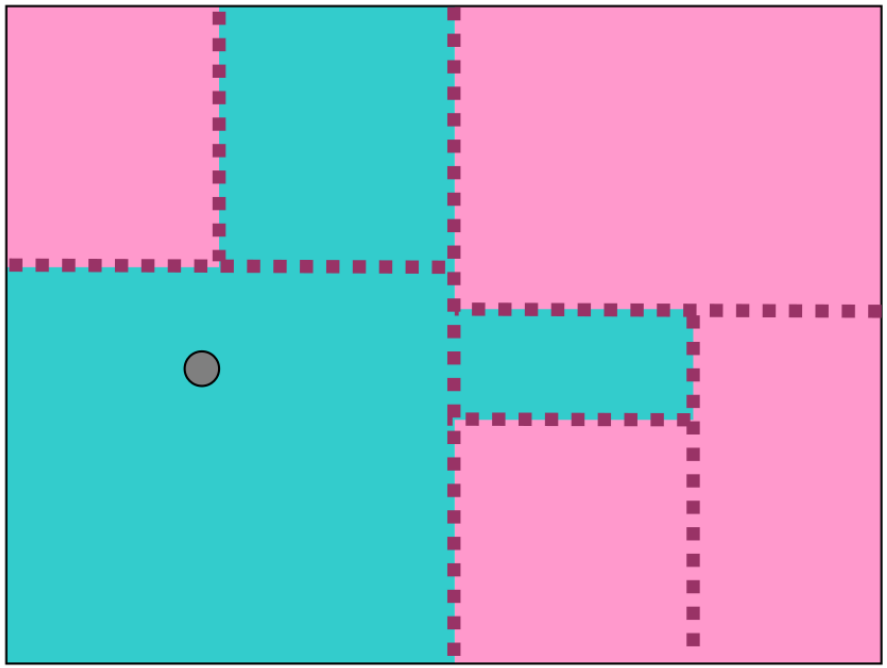
1) We compute a measure for *each possible split* in each feature
   $\rightarrow$ here **absolute error rate** (AER)
2) Recursively repead step (1) for each subspace until AER $\rightarrow 0$

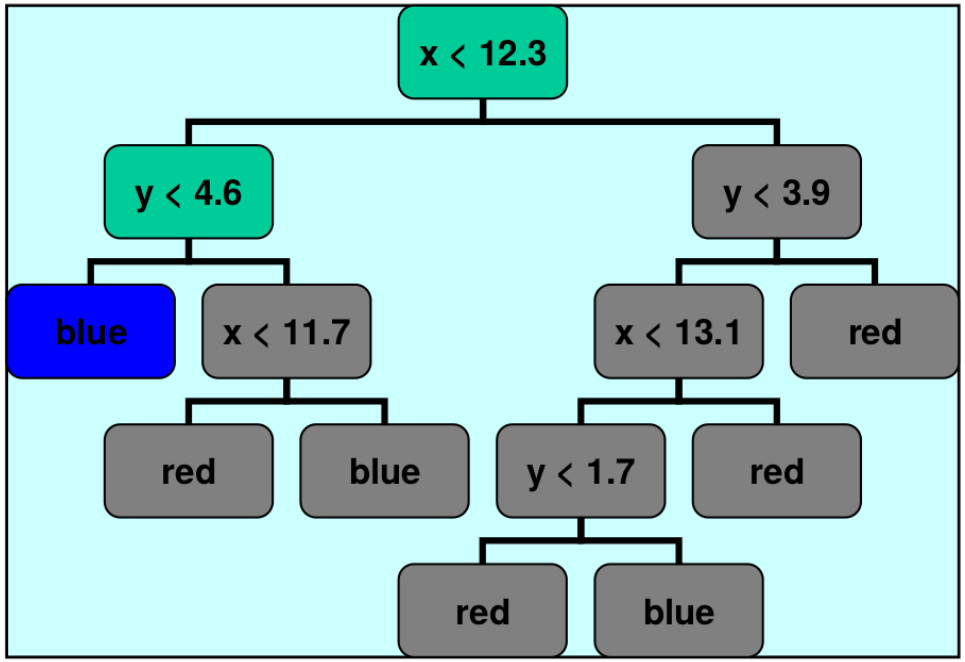




Decision tree classification

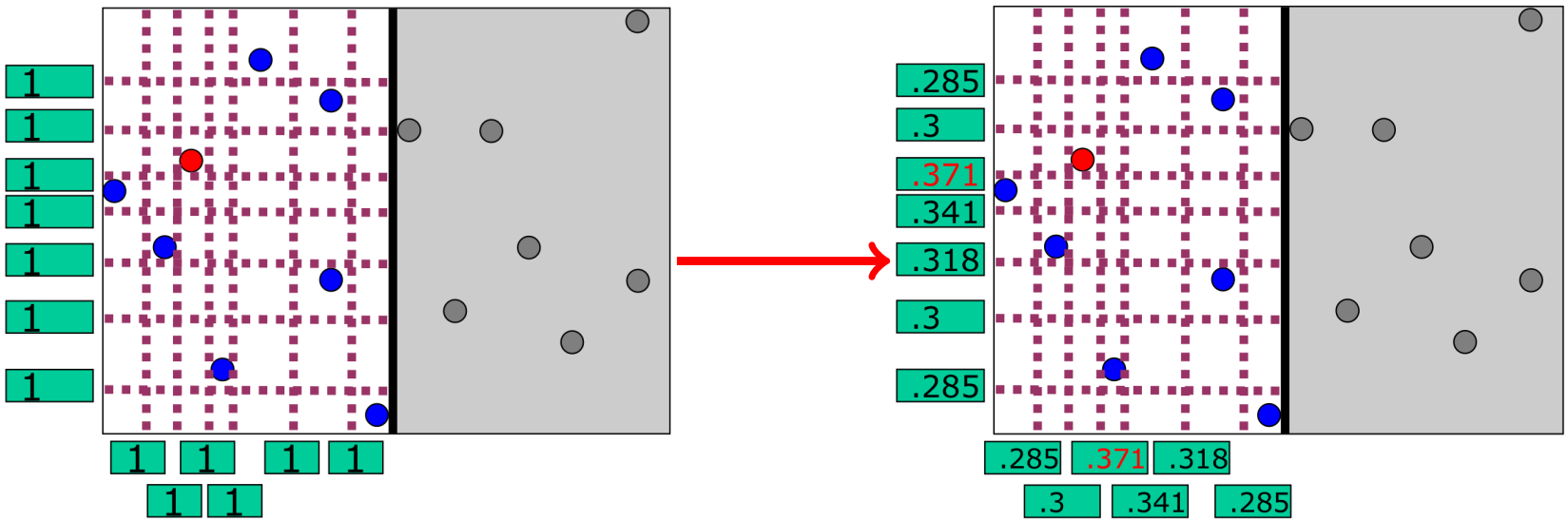
3) Classification





Decision tree improvements I

- Use more sophisticated split measures
  - *Information gain* $\leftrightarrow$ (im-)purity of splitted sub-sets
  - Gini index
- Pruning



Absolute error rate

Information gain

Decision tree improvements II

Random forest

- Ensemble of DTs
- For each tree use:
  - Random sub-sample (=*bootstrapping*)
  - Random number of the original features
  - → large number of rather shallow trees
- Classify data by majority voting of individual trees

Boosted DT

- Sequential ensemble of evolving DTs
- Assign weights to training samples → initially equal weights
- Adjust weights & give more importance to misclassified samples
- Subsequent predictors thus should focus on those

Decision tree improvements II

Random forest

- Ensemble of DTs
- For each tree use:
  - Random sub-sample (=*bootstrapping*)
  - Random number of the original features $\rightarrow$ large number of rather shallow trees
- Classify data by majority voting of individual trees

Boosted DT

- Sequential ensemble of evolving DTs
- Assign weights to training samples $\rightarrow$ initially equal weights
- Adjust weights & give more importance to misclassified samples
- Subsequent predictors thus should focus on those

Decision tree improvements II

Random forest

- Ensemble of DTs
- For each tree use:
  - Random sub-sample (=*bootstrapping*)
  - Random number of the original features $\rightarrow$ large number of rather shallow trees
- Classify data by majority voting of individual trees

Boosted DT

- Sequential ensemble of evolving DTs
- Assign weights to training samples $\rightarrow$ initially equal weights
- Adjust weights & give more importance to misclassified samples
- Subsequent predictors thus should focus on those

Decision tree improvements II

Random forest

- Ensemble of DTs
- For each tree use:
  - Random sub-sample (=*bootstrapping*)
  - Random number of the original features
    $\rightarrow$ large number of rather shallow trees
- Classify data by majority voting of individual trees

Boosted DT

- Sequential ensemble of evolving DTs
- Assign weights to training samples $\rightarrow$ initially equal weights
- Adjust weights & give more importance to misclassified samples
- Subsequent predictors thus should focus on those

Decision tree improvements II

Random forest

- Ensemble of DTs
- For each tree use:
  - Random sub-sample (=*bootstrapping*)
  - Random number of the original features
    $\rightarrow$ large number of rather shallow trees
- Classify data by majority voting of individual trees

Boosted DT

- Sequential ensemble of evolving DTs
- Assign weights to training samples $\rightarrow$ initially equal weights
- Adjust weights & give more importance to misclassified samples
- Subsequent predictors thus should focus on those

Decision tree improvements II

Random forest

- Ensemble of DTs
- For each tree use:
  - Random sub-sample (=*bootstrapping*)
  - Random number of the original features
    $\rightarrow$ large number of rather shallow trees
- Classify data by majority voting of individual trees

Boosted DT

- Sequential ensemble of evolving DTs
- Assign weights to training samples $\rightarrow$ initially equal weights
- Adjust weights & give more importance to misclassified samples
- Subsequent predictors thus should focus on those

Decision tree improvements II

Random forest

- Ensemble of DTs
- For each tree use:
  - Random sub-sample (=*bootstrapping*)
  - Random number of the original features
    $\rightarrow$ large number of rather shallow trees
- Classify data by majority voting of individual trees

Boosted DT

- Sequential ensemble of evolving DTs
- Assign weights to training samples $\rightarrow$ initially equal weights
- Adjust weights & give more importance to misclassified samples
- Subsequent predictors thus should focus on those

Decision tree improvements II

Random forest

- Ensemble of DTs
- For each tree use:
  - Random sub-sample (=*bootstrapping*)
  - Random number of the original features
    $\rightarrow$ large number of rather shallow trees
- Classify data by majority voting of individual trees

Boosted DT

- Sequential ensemble of evolving DTs
- Assign weights to training samples $\rightarrow$ initially equal weights
- Adjust weights & give more importance to misclassified samples
- Subsequent predictors thus should focus on those

Decision tree improvements II

Random forest

- Ensemble of DTs
- For each tree use:
  - Random sub-sample (=*bootstrapping*)
  - Random number of the original features
    $\rightarrow$ large number of rather shallow trees
- Classify data by majority voting of individual trees

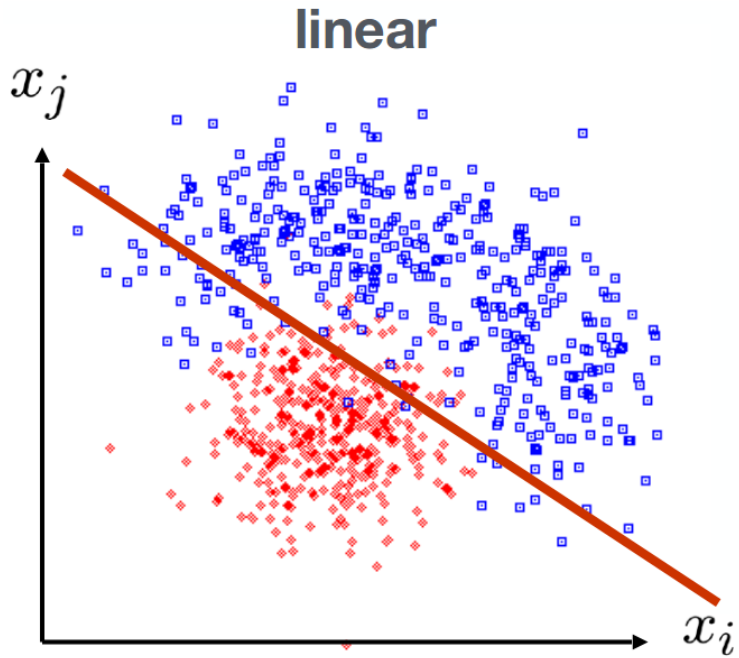
Boosted DT

- Sequential ensemble of evolving DTs
- Assign weights to training samples $\rightarrow$ initially equal weights
- Adjust weights & give more importance to misclassified samples
- Subsequent predictors thus should focus on those

Linear cuts

More degrees of freedom than rectangular cut

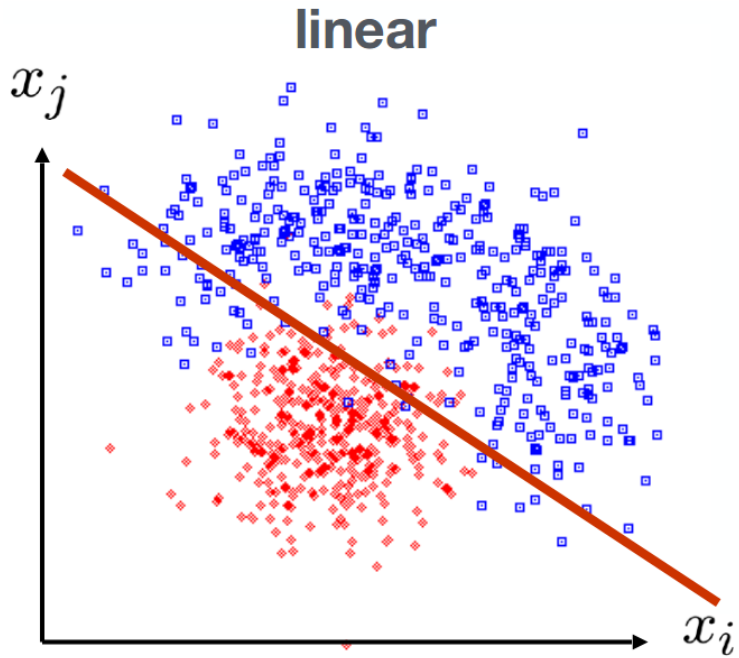
- Simple white box methods
- Very fast classification
- Can become very powerful by using *kernel trick*



Linear cuts

More degrees of freedom than rectangular cut

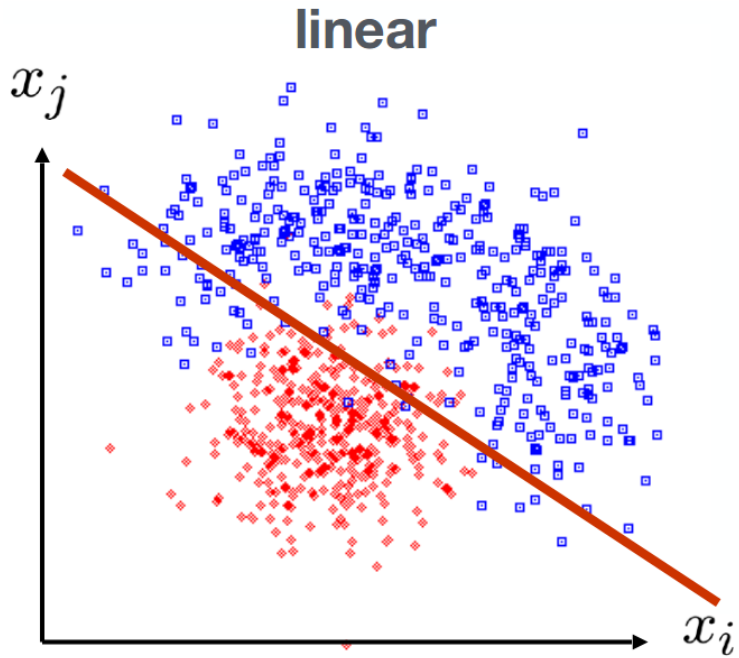
- Simple white box methods
- Very fast classification
- Can become very powerful by using *kernel trick*



Linear cuts

More degrees of freedom than rectangular cut

- Simple white box methods
- Very fast classification
- Can become very powerful by using *kernel trick*



Linear models

Takes a *linear function* of its inputs $\boldsymbol{x} = (x_1, \ldots x_n)$ to base its decision on.

$$y : \mathbb{R}^n \to \mathbb{R} \mid \boldsymbol{x} \mapsto y = f(\boldsymbol{w} \cdot \boldsymbol{x}) = f(\sum_j w_j x_j)$$

$\boldsymbol{w} \ldots$ weight vector

Simplest case

$$y = f(x) = \Theta(x) = \begin{cases} 0 & x < 0 \quad \text{signal} \\ 1 & x \geq 0 \quad \text{background} \end{cases}$$

$\rightarrow$ Function can be approximated by **single layer perceptron**

Linear models

Takes a *linear function* of its inputs $\boldsymbol{x} = (x_1, \ldots x_n)$ to base its decision on.

$$y : \mathbb{R}^n \to \mathbb{R} \mid \boldsymbol{x} \mapsto y = f(\boldsymbol{w} \cdot \boldsymbol{x}) = f(\sum_j w_j x_j)$$

$\boldsymbol{w} \ldots$ weight vector

Simplest case

$$y = f(x) = \Theta(x) = \begin{cases} 0 & x < 0 \quad \text{signal} \\ 1 & x \geq 0 \quad \text{background} \end{cases}$$

$\rightarrow$ Function can be approximated by **single layer perceptron**

Linear models

Takes a *linear function* of its inputs $\boldsymbol{x} = (x_1, \ldots x_n)$ to base its decision on.

$$y : \mathbb{R}^n \to \mathbb{R} \mid \boldsymbol{x} \mapsto y = f(\boldsymbol{w} \cdot \boldsymbol{x}) = f(\sum_j w_j x_j)$$

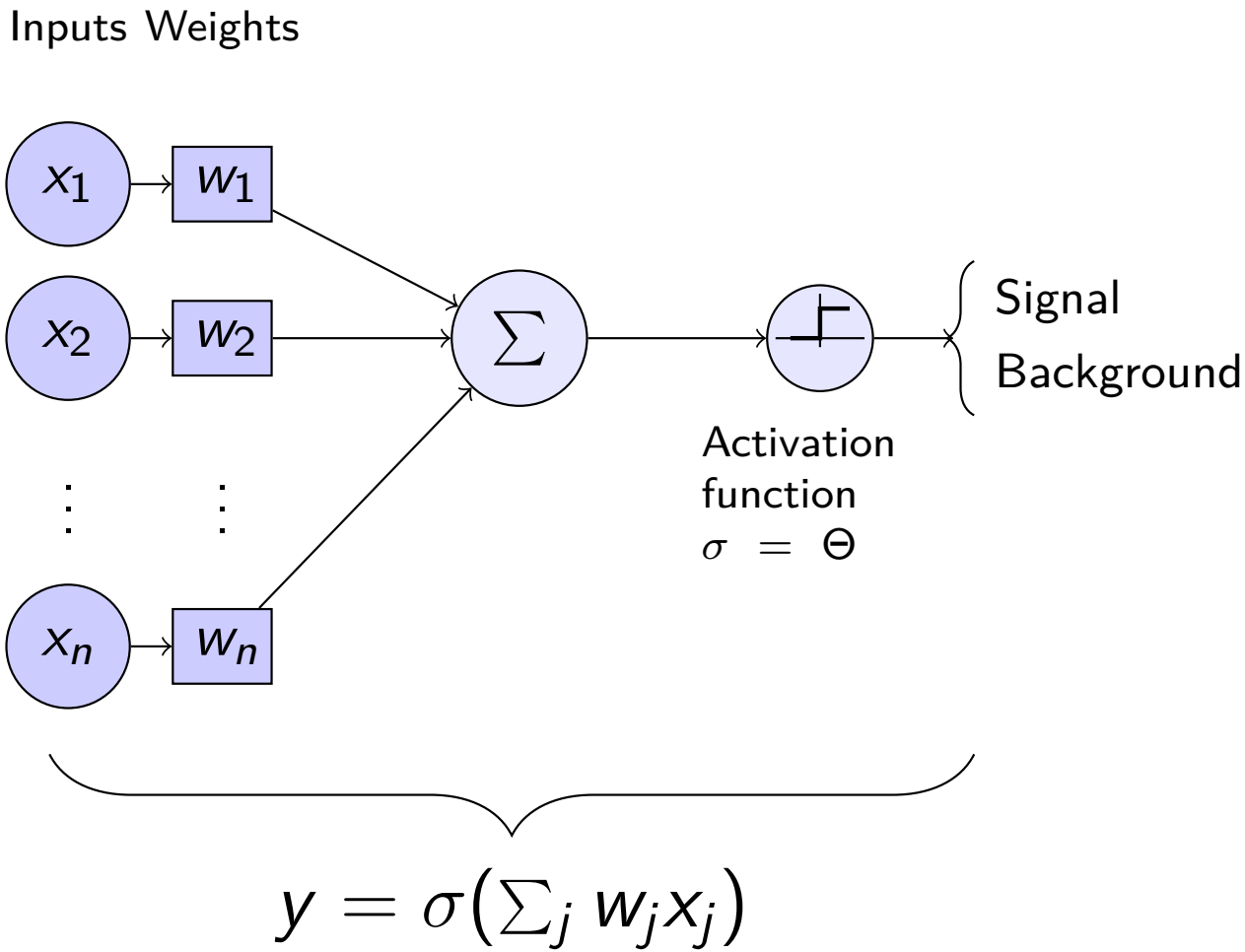
$\boldsymbol{w} \ldots$ weight vector

Simplest case

$$y = f(x) = \Theta(x) = \begin{cases} 0 & x < 0 \quad \text{signal} \\ 1 & x \geq 0 \quad \text{background} \end{cases}$$

$\rightarrow$ Function can be approximated by **single layer perceptron**

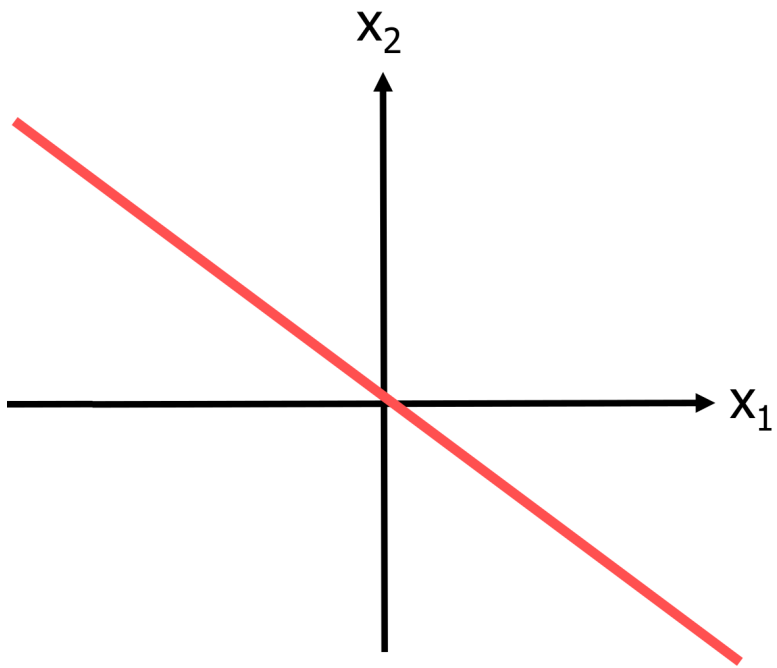
Single layer perceptron (SLP)



SLP training

Algorithm

1. Initialize weights $\boldsymbol{w}_{init}$
2. Repeat until $y = y_{target}$:
   1. Present training sample x
   2. Predict sample label $y = \Theta(x\boldsymbol{w}_{init})$ and compute error $\Delta = y_{target} - y$
   3. If $\Delta \neq 0 \rightarrow$ update weights $\boldsymbol{w}' = \boldsymbol{w} + \alpha\Delta\boldsymbol{x}$

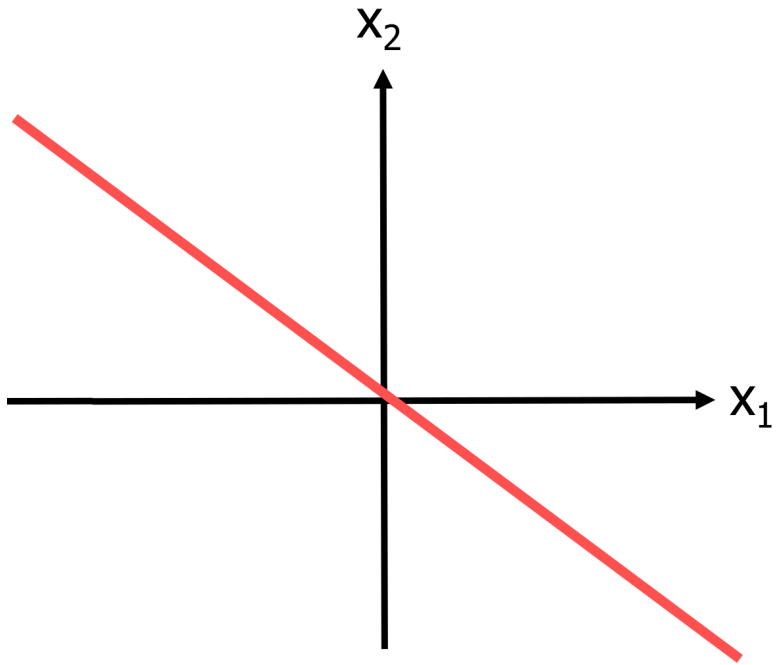


$$y = w_1 x_1 + w_2 x_2$$

SLP training

Algorithm

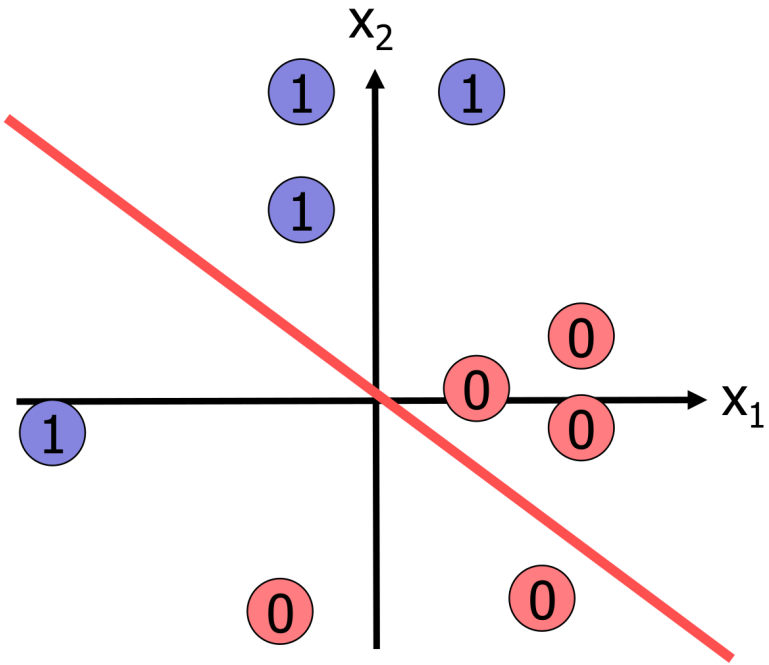
1. Initialize weights $\boldsymbol{w}_{init}$
2. Repeat until $y = y_{target}$:
   1. Present training sample $\boldsymbol{x}$
   2. Predict sample label $y = \Theta(\boldsymbol{x}\boldsymbol{w}_{init})$ and compute error $\Delta = y_{target} - y$
   3. If $\Delta \neq 0 \rightarrow$ update weights $\boldsymbol{w}' = \boldsymbol{w} + \alpha\Delta\boldsymbol{x}$



SLP training

Algorithm

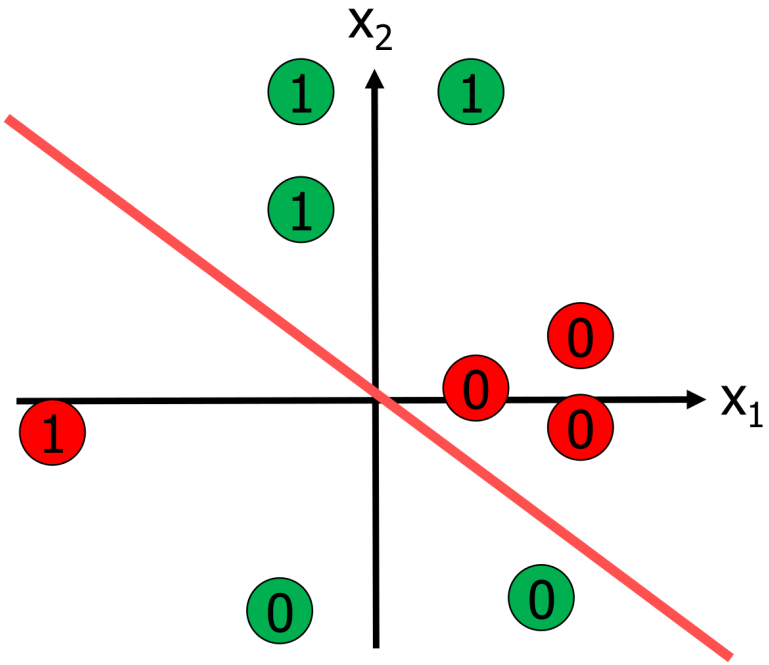
1. Initialize weights $\boldsymbol{w}_{init}$
2. Repeat until $y = y_{target}$:
   1. Present training sample $\boldsymbol{x}$
   2. Predict sample label $y = \Theta(\boldsymbol{x}\boldsymbol{w}_{init})$ and compute error $\Delta = y_{target} - y$
   3. If $\Delta \neq 0 \rightarrow$ update weights $\boldsymbol{w}' = \boldsymbol{w} + \alpha\Delta\boldsymbol{x}$



SLP training

Algorithm

1. Initialize weights $\boldsymbol{w}_{init}$
2. Repeat until $y = y_{target}$:
   1. Present training sample $\boldsymbol{x}$
   2. Predict sample label $y = \Theta(\boldsymbol{x}\boldsymbol{w}_{init})$ and compute error $\Delta = y_{target} - y$
   3. If $\Delta \neq 0 \rightarrow$ update weights $\boldsymbol{w}' = \boldsymbol{w} + \alpha\Delta\boldsymbol{x}$

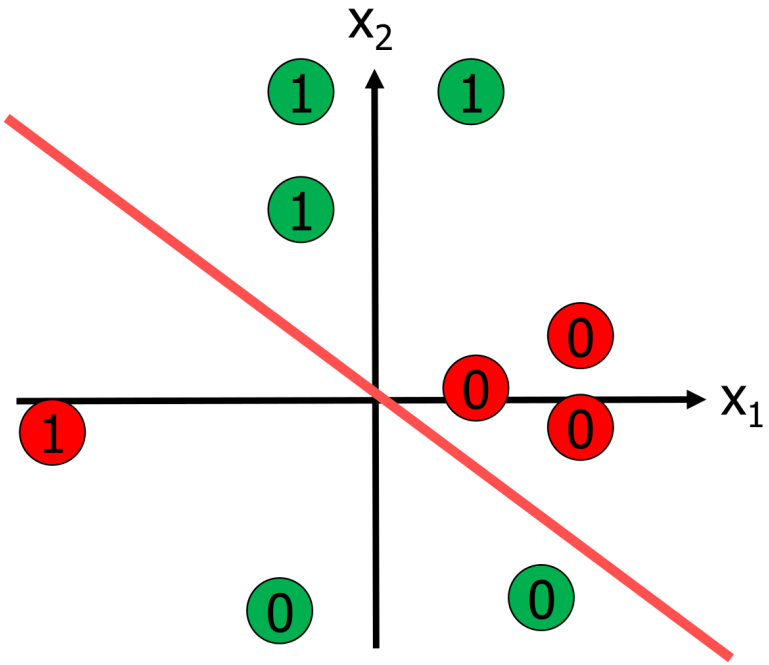


SLP training

Algorithm

1. Initialize weights $\boldsymbol{w}_{init}$
2. Repeat until $y = y_{target}$:
   1. Present training sample $\boldsymbol{x}$
   2. Predict sample label $y = \Theta(\boldsymbol{x}\boldsymbol{w}_{init})$ and compute error $\Delta = y_{target} - y$
   3. If $\Delta \neq 0 \rightarrow$ update weights $\boldsymbol{w}' = \boldsymbol{w} + \alpha\Delta\boldsymbol{x}$

$\alpha \ldots$ learning rate

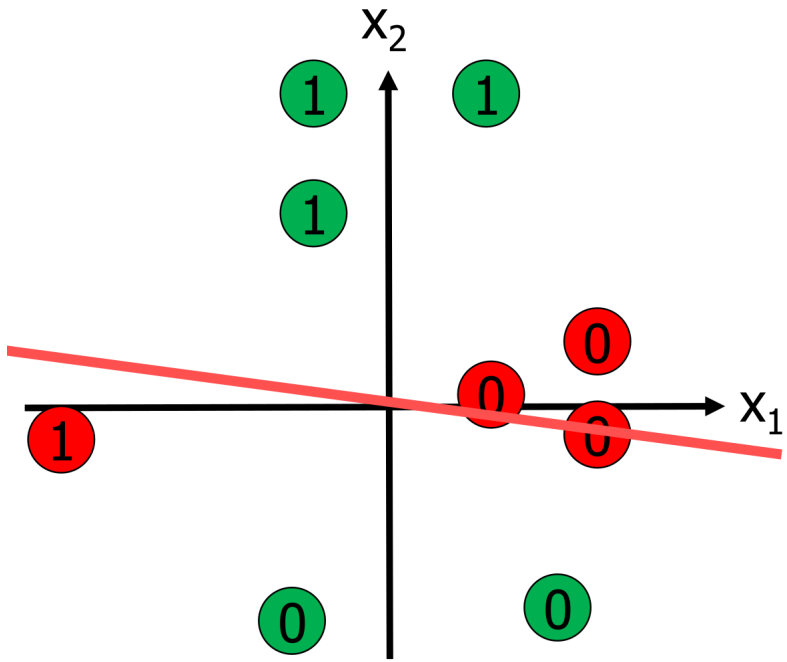


SLP training

Algorithm

1. Initialize weights $\boldsymbol{w}_{init}$
2. Repeat until $y = y_{target}$:
   1. Present training sample $\boldsymbol{x}$
   2. Predict sample label $y = \Theta(\boldsymbol{x}\boldsymbol{w}_{init})$ and compute error $\Delta = y_{target} - y$
   3. If $\Delta \neq 0 \rightarrow$ update weights $\boldsymbol{w}' = \boldsymbol{w} + \alpha\Delta\boldsymbol{x}$

$\alpha \ldots$ learning rate

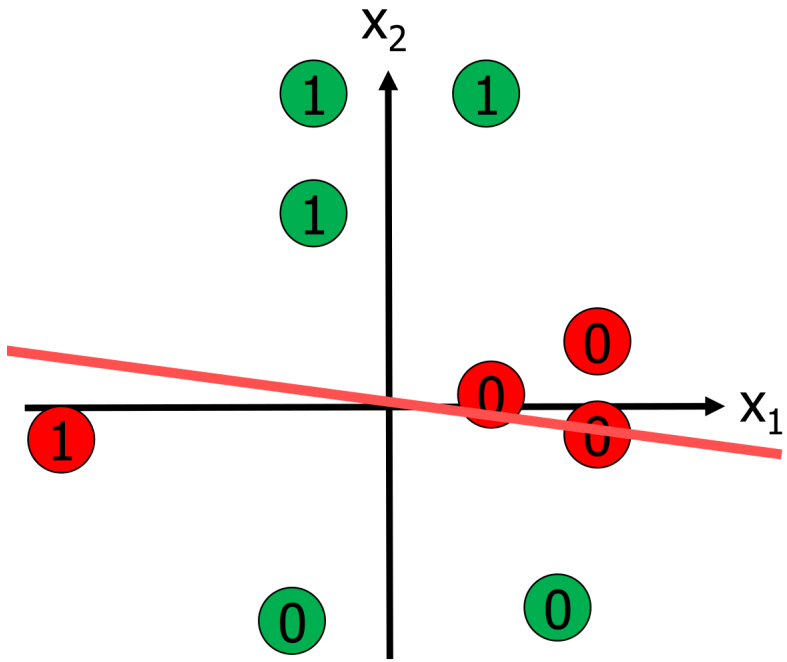


SLP training

Algorithm

1. Initialize weights $\boldsymbol{w}_{init}$
2. Repeat until $y = y_{target}$:
   1. Present training sample $\boldsymbol{x}$
   2. Predict sample label $y = \Theta(\boldsymbol{x}\boldsymbol{w}_{init})$ and compute error $\Delta = y_{target} - y$
   3. If $\Delta \neq 0 \rightarrow$ update weights $\boldsymbol{w}' = \boldsymbol{w} + \alpha\Delta\boldsymbol{x}$

$\alpha \ldots$ learning rate

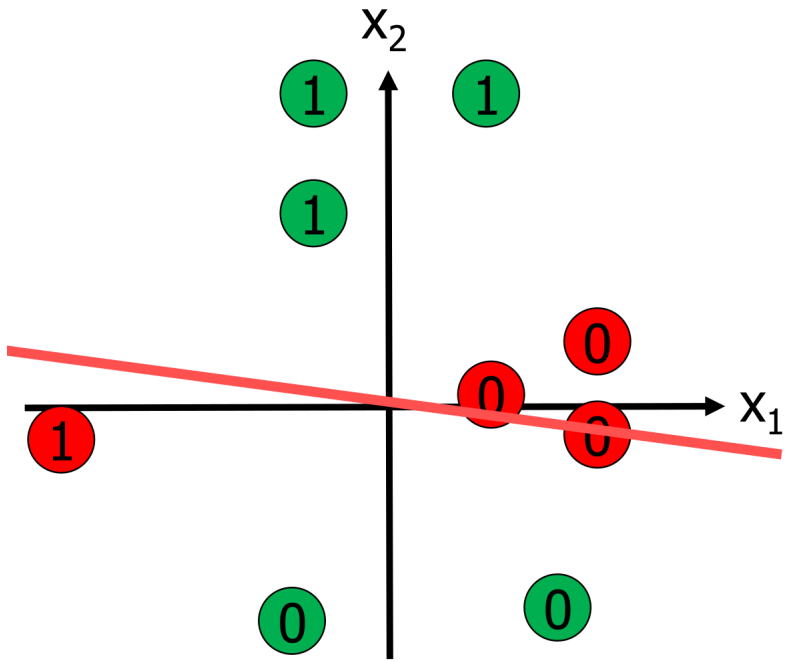


SLP training

Algorithm

1. Initialize weights $\boldsymbol{w}_{init}$
2. Repeat until $y = y_{target}$:
   1. Present training sample $\boldsymbol{x}$
   2. Predict sample label $y = \Theta(\boldsymbol{x}\boldsymbol{w}_{init})$ and compute error $\Delta = y_{target} - y$
   3. If $\Delta \neq 0 \rightarrow$ update weights $\boldsymbol{w}' = \boldsymbol{w} + \alpha\Delta\boldsymbol{x}$

$\alpha \ldots$ learning rate

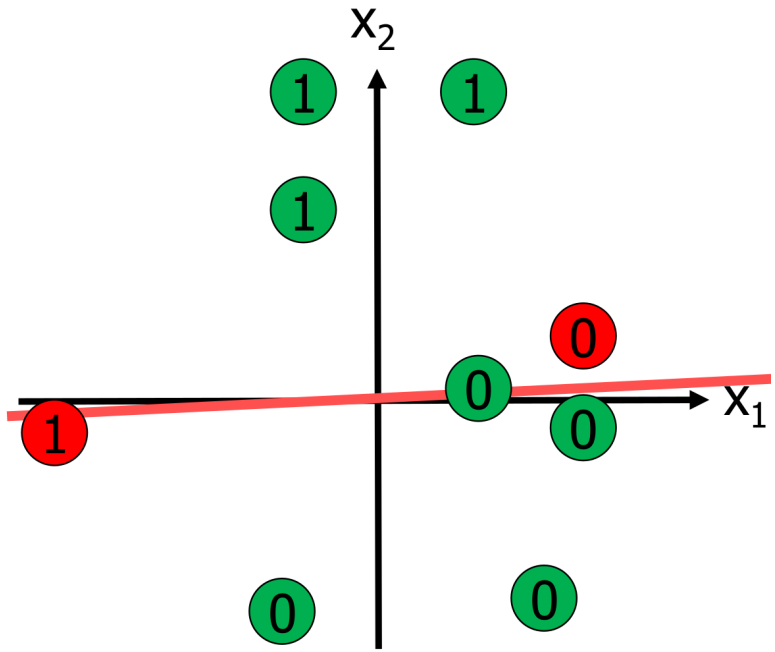


SLP training

Algorithm

1. Initialize weights $\boldsymbol{w}_{init}$
2. Repeat until $y = y_{target}$:
   1. Present training sample $\boldsymbol{x}$
   2. Predict sample label $y = \Theta(\boldsymbol{x}\boldsymbol{w}_{init})$ and compute error $\Delta = y_{target} - y$
   3. If $\Delta \neq 0 \rightarrow$ update weights $\boldsymbol{w}' = \boldsymbol{w} + \alpha\Delta\boldsymbol{x}$

$\alpha \ldots$ learning rate

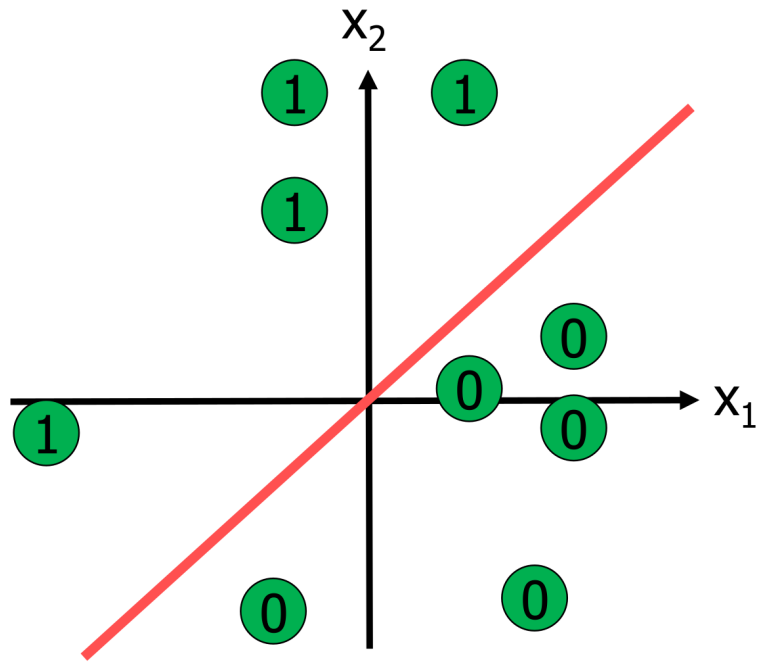


SLP training

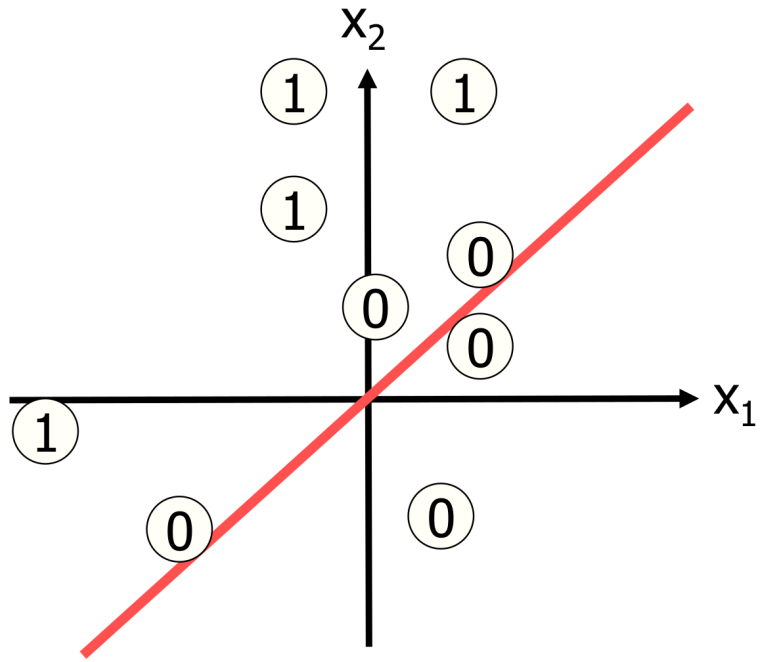
Algorithm

1. Initialize weights $\boldsymbol{w}_{init}$
2. Repeat until $y = y_{target}$:
   1. Present training sample $\boldsymbol{x}$
   2. Predict sample label $y = \Theta(\boldsymbol{x}\boldsymbol{w}_{init})$ and compute error $\Delta = y_{target} - y$
   3. If $\Delta \neq 0 \rightarrow$ update weights $\boldsymbol{w}' = \boldsymbol{w} + \alpha\Delta\boldsymbol{x}$

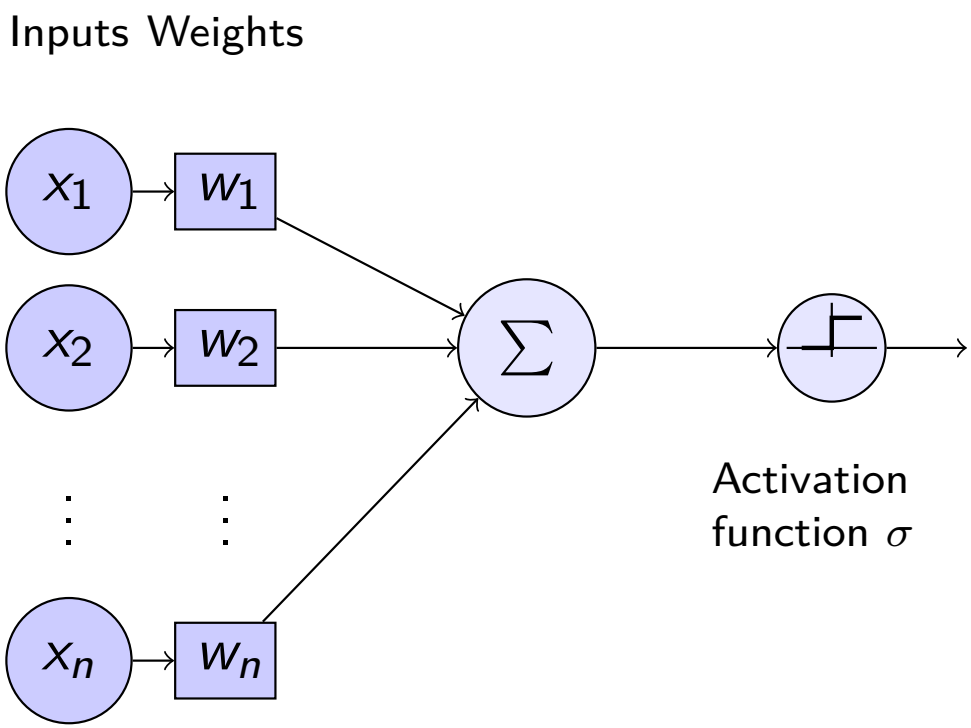
$\alpha \ldots$ learning rate

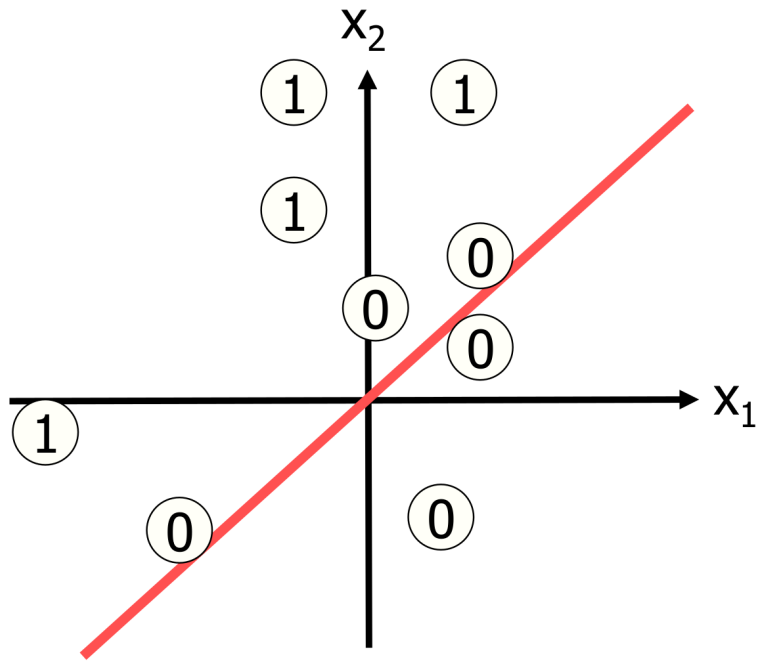


SLP bias term

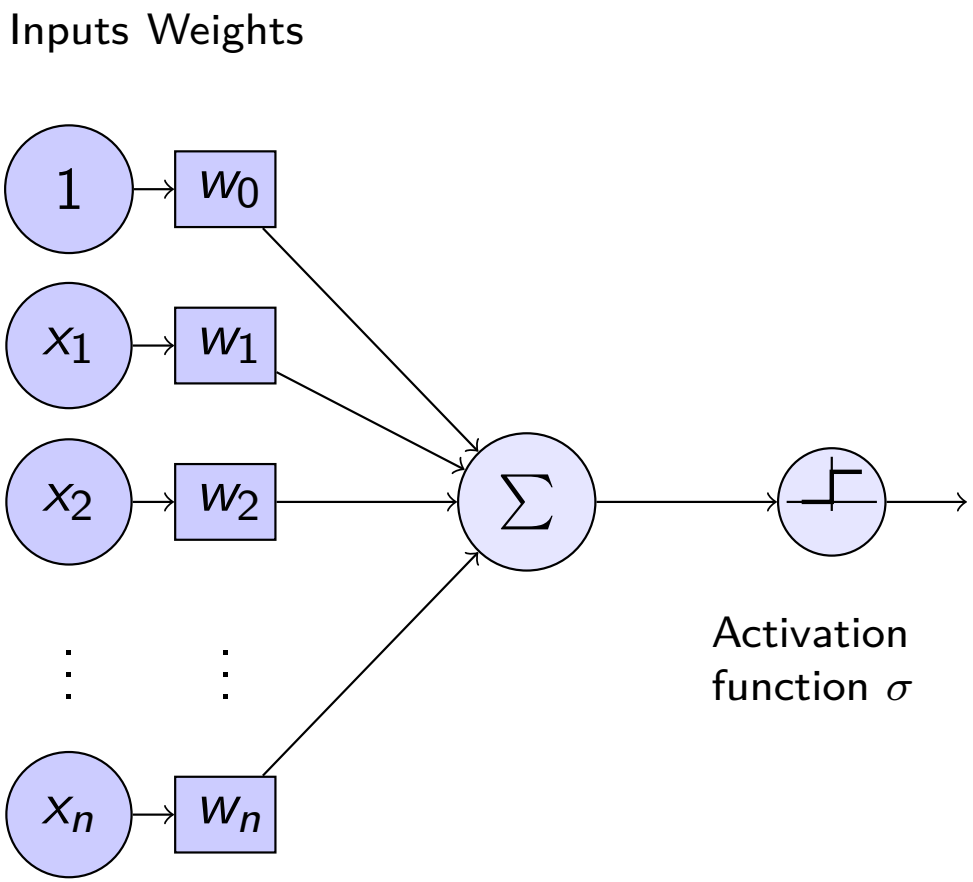


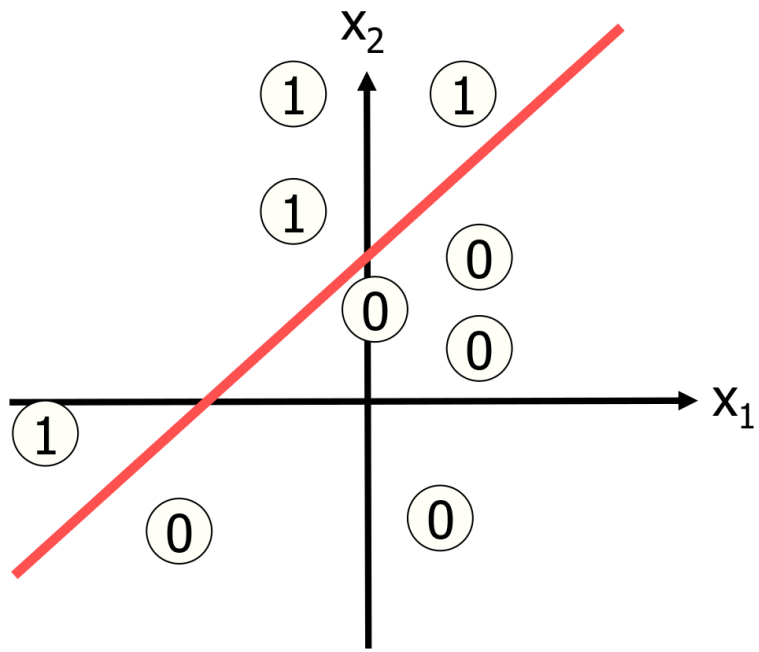
SLP bias term



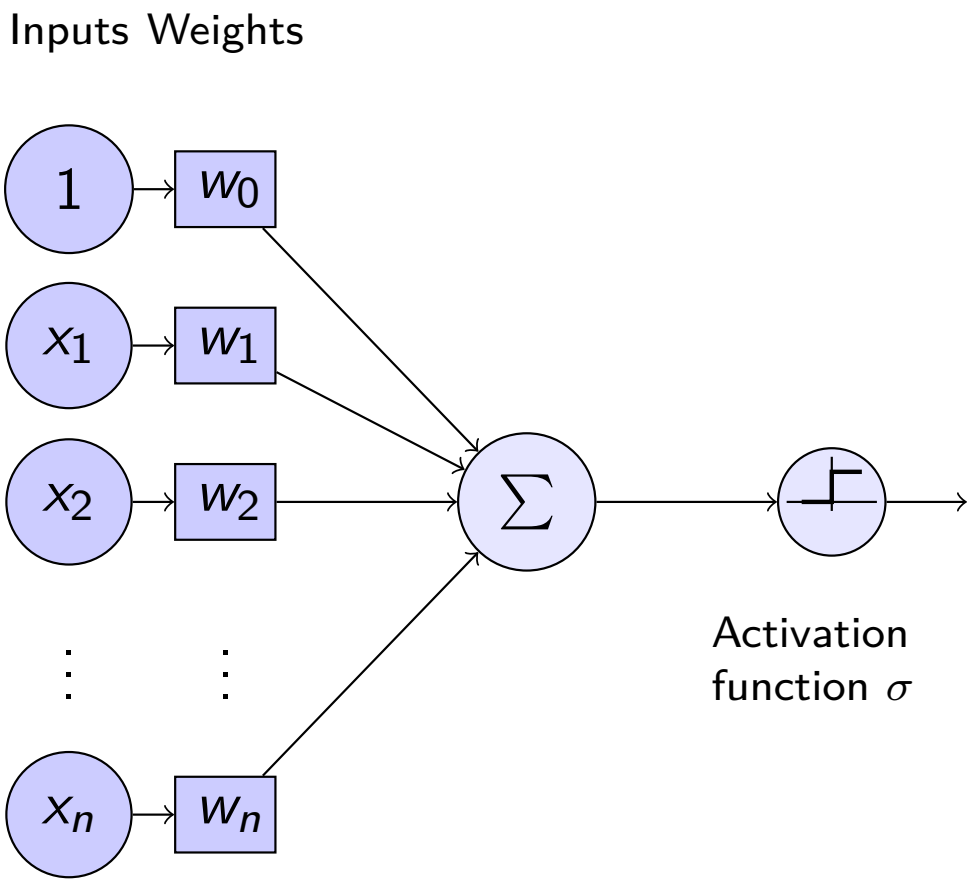


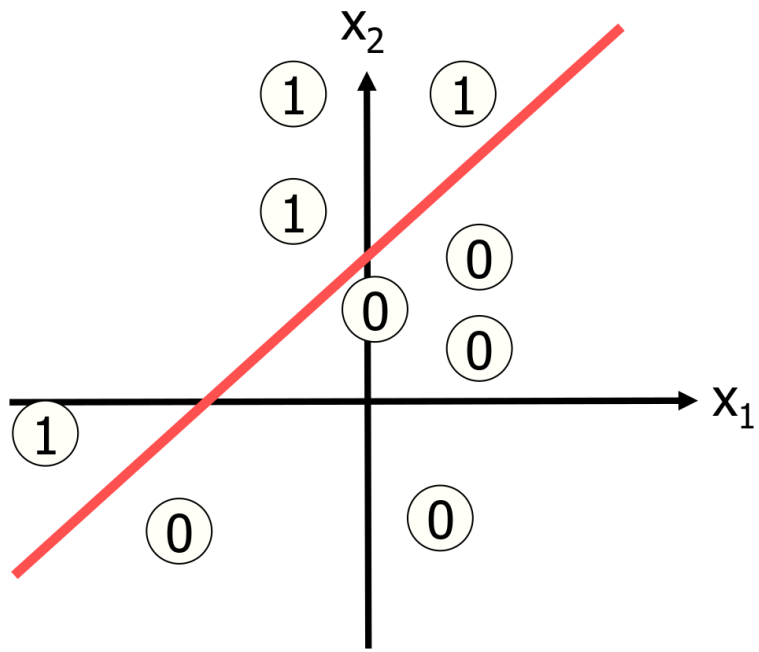
SLP bias term



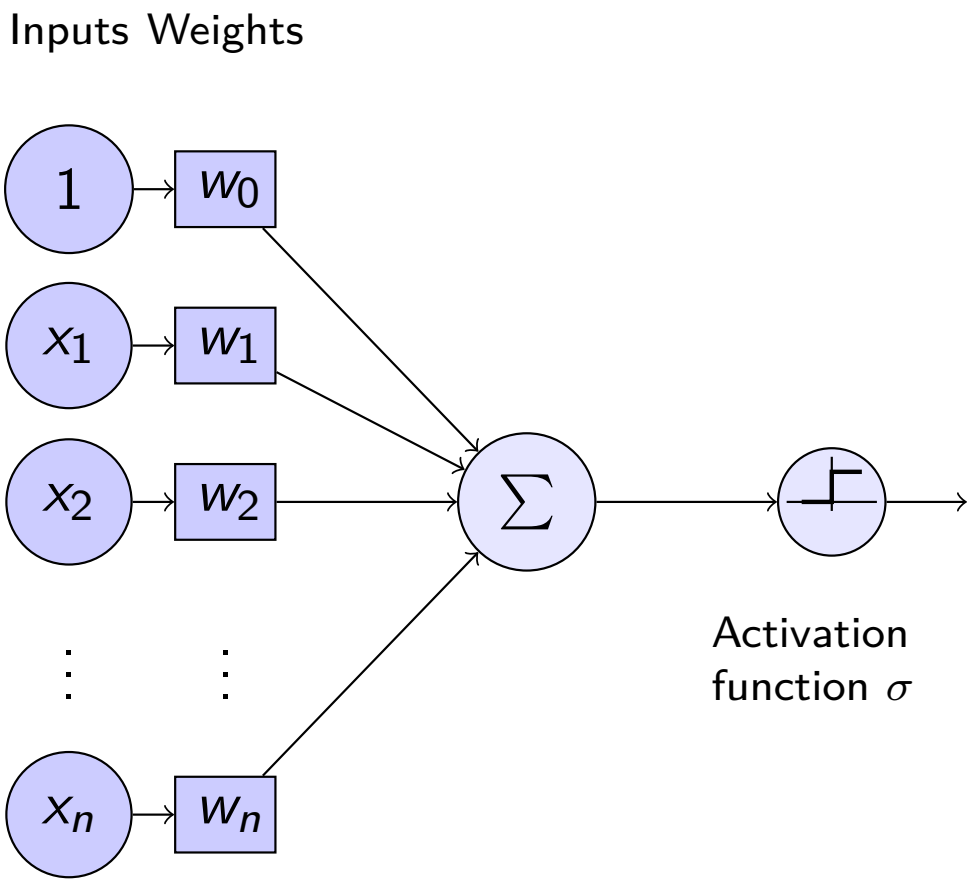


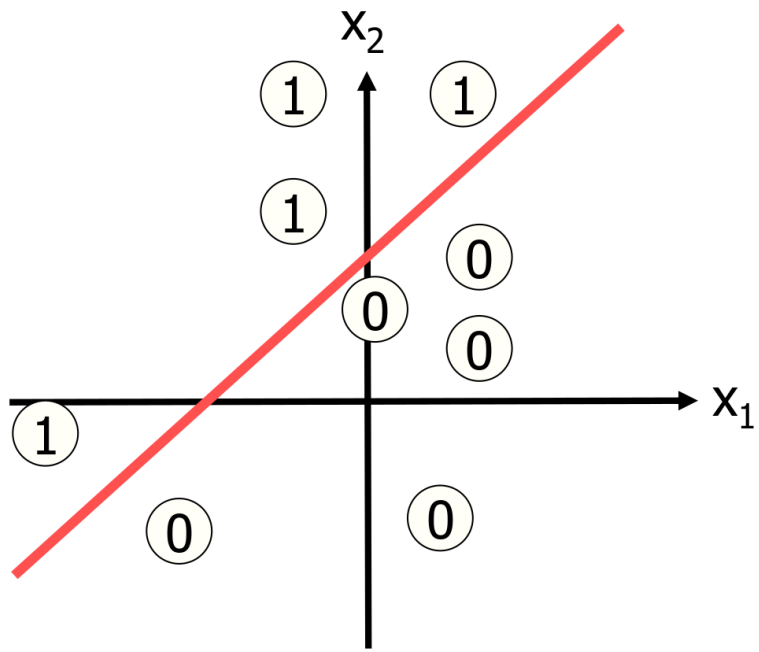
SLP bias term





SLP bias term

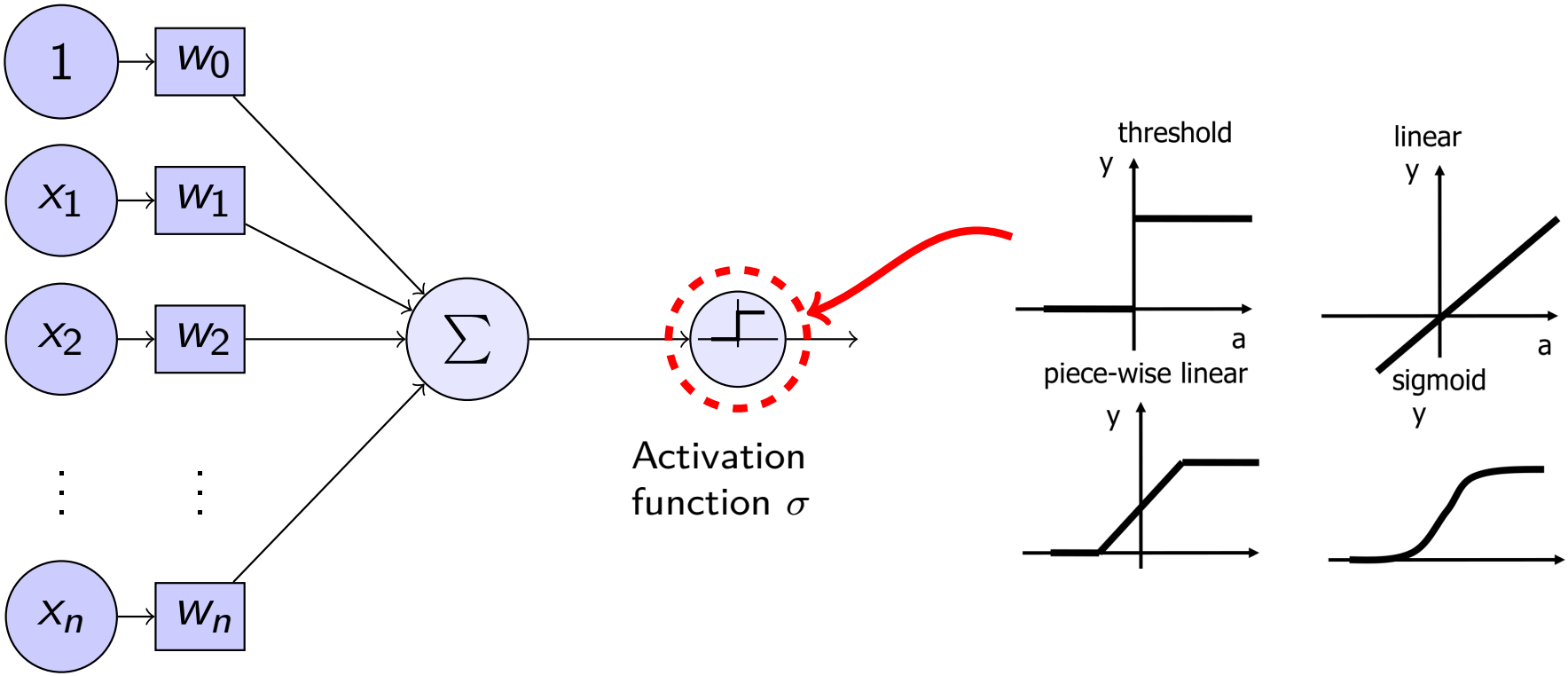




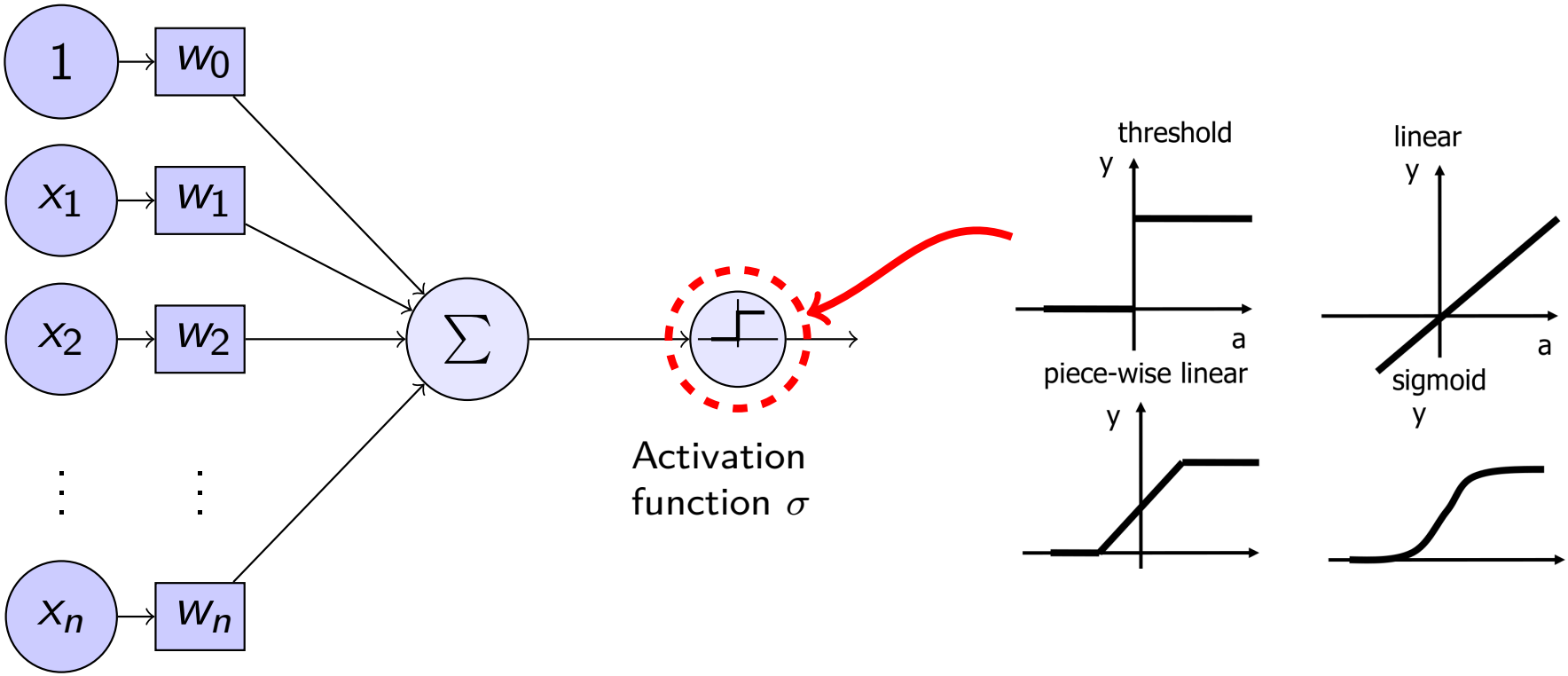
Bias weight w_0 learned just as the other weights

$$y = \sigma(w_0 + \sum_{j=1}^{n} w_j x_j)$$

SLP activation functions

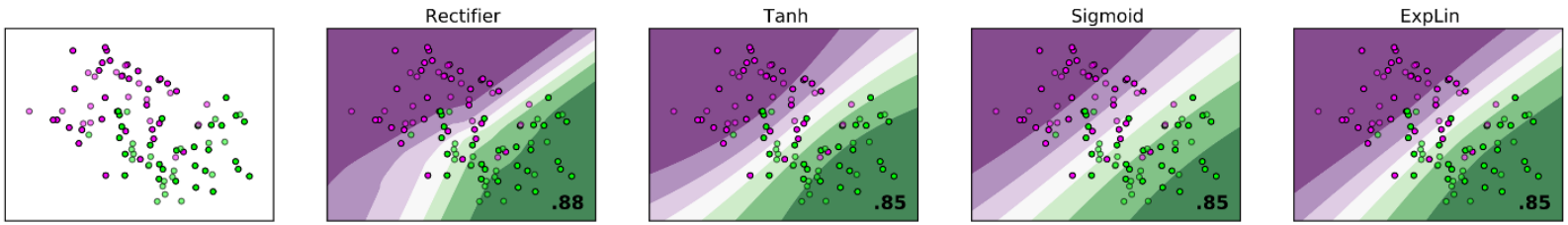


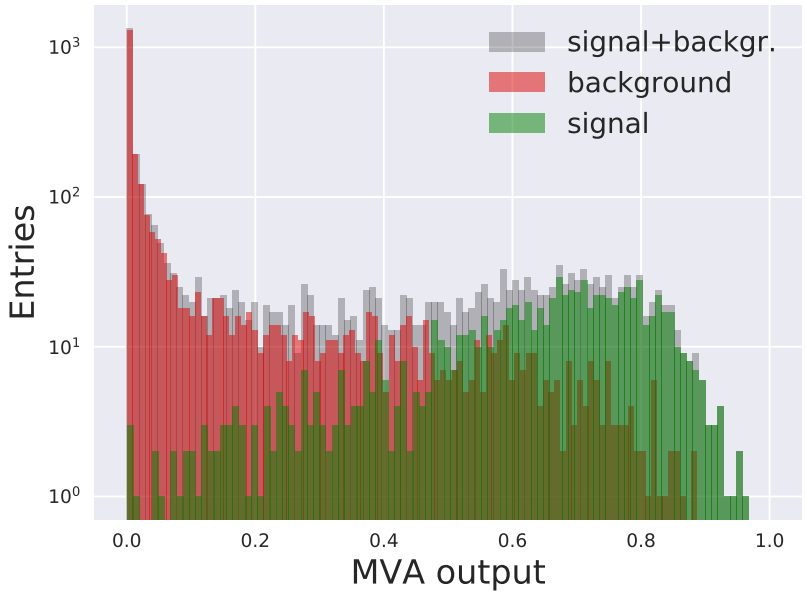
SLP activation functions



Often used: *sigmoid* $\rightarrow$ output $\in (0, 1)$

SLP improvement INon-linear activation functions output





Improving linear methods

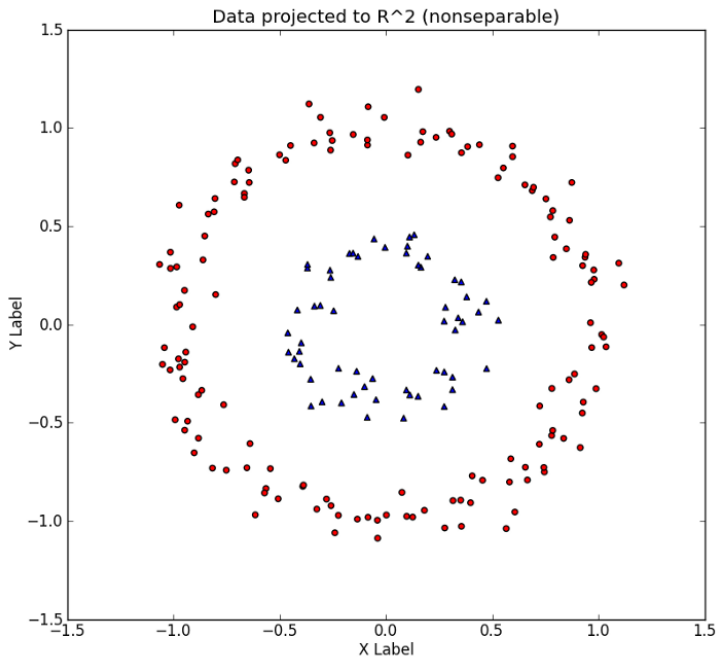
Kernel trick

Map data to a higher dimensional space where linear hyperplane can again be found

Improving linear methods

Kernel trick

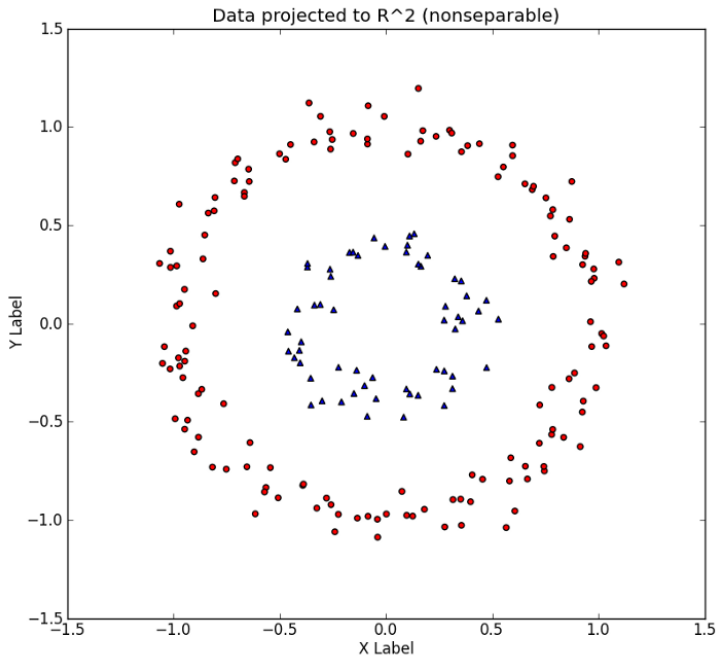
Map data to a higher dimensional space where linear hyperplane can again be found

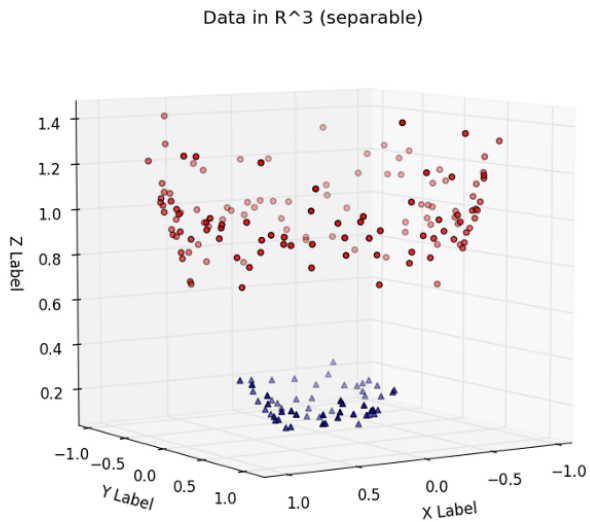


Improving linear methods

Kernel trick

Map data to a higher dimensional space where linear hyperplane can again be found





Non-linear models

Assumtion

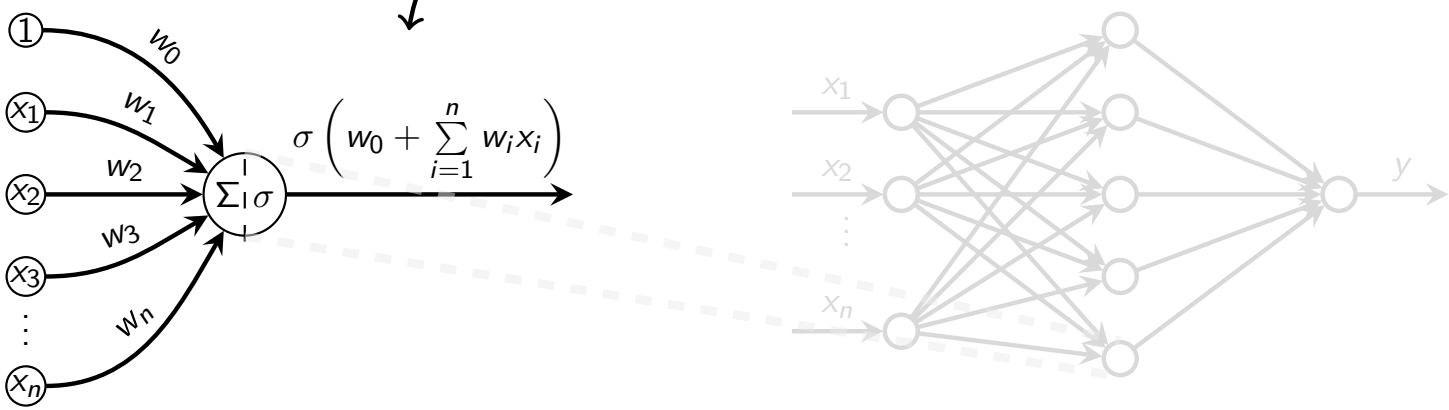
Take *kernel trick* idea: map inputs $\boldsymbol{x} = (x_1, \dots x_n)$ to higher dimensional space $\mathbb{R}^m$ with $m > n$ where linear seperation is possible:

$$y : \mathbb{R}^n \to \mathbb{R}^m \to \mathbb{R} = \sigma\left(w_0^2 + \sum_{i=1}^{m}\left[w_i^{(2)} + \sigma\left(w_{0,i}^{(1)} + \sum_{k=1}^{n} w_{k,i}^{(1)}\ x_k\right)\right]\right)$$
$$= \sigma\left[w_0^{(2)} + \boldsymbol{w}^{(2)} \cdot \sigma(\boldsymbol{w}_0^{(1)} + \boldsymbol{W}^{(1)} \cdot \boldsymbol{x})\right]$$

$\to$ *m single layer perceptrons*

Visualisaion

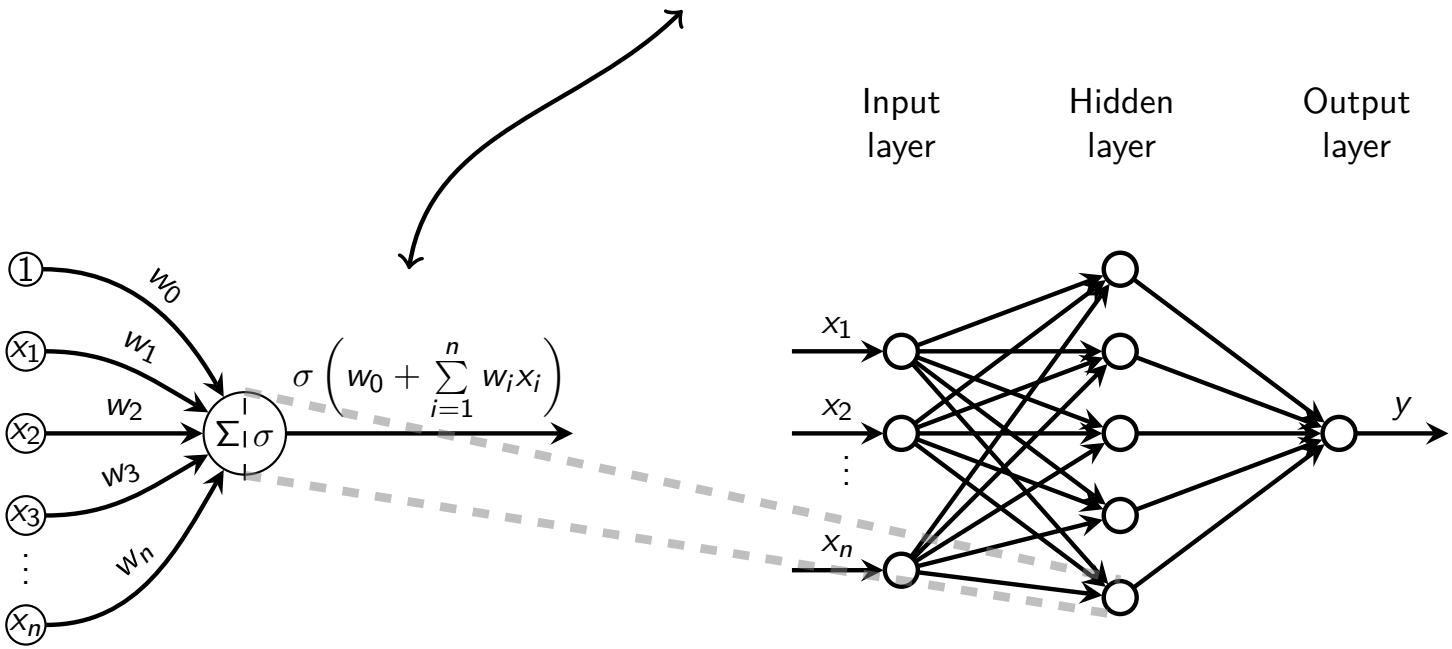
$$y = \sigma[w_0^{(2)} + \overbrace{\boldsymbol{w}^{(2)}}^{\mathbb{R}^m} \cdot \underbrace{\sigma(\overbrace{\boldsymbol{w}_0^{(1)}}^{\mathbb{R}^m} + \overbrace{\boldsymbol{W}^{(1)}}^{\mathbb{R}^{m\times n}} \cdot \boldsymbol{x})}_{\text{Output of hidden layer: } \mathbb{R}^m}]$$



$\rightarrow$ The hidden layer *learns* a representation so that the data is linearly separable

Visualisaion

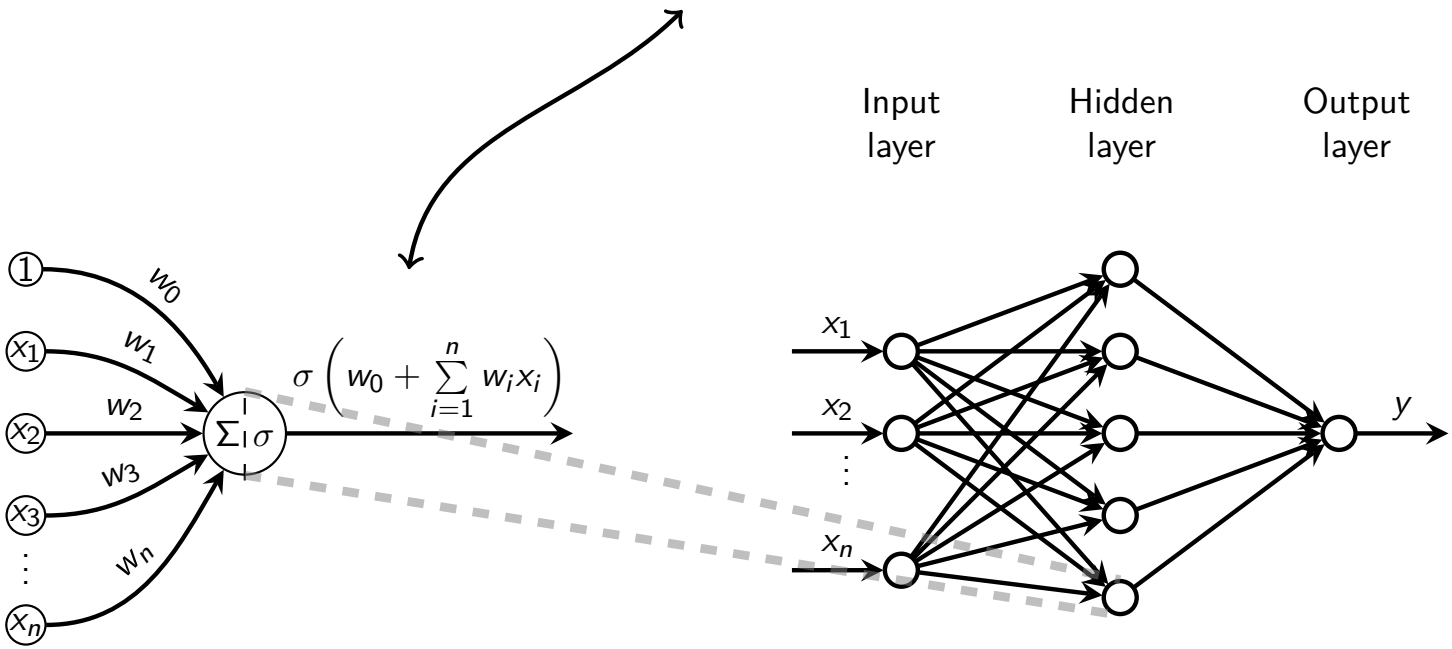
$$y = \sigma[w_0^{(2)} + \overbrace{\boldsymbol{w}^{(2)}}^{\mathbb{R}^m} \cdot \underbrace{\sigma(\overbrace{\boldsymbol{w}_0^{(1)}}^{\mathbb{R}^m} + \overbrace{\boldsymbol{W}^{(1)}}^{\mathbb{R}^{m\times n}} \cdot \boldsymbol{x})}_{\text{Output of hidden layer: } \mathbb{R}^m}]$$



$\rightarrow$ The hidden layer *learns* a representation so that the data is linearly separable

Visualisaion

$$y = \sigma[w_0^{(2)} + \overbrace{\boldsymbol{w}^{(2)}}^{\mathbb{R}^m} \cdot \underbrace{\sigma(\overbrace{\boldsymbol{w}_0^{(1)}}^{\mathbb{R}^m} + \overbrace{\boldsymbol{W}^{(1)}}^{\mathbb{R}^{m\times n}} \cdot \boldsymbol{x})}_{\text{Output of hidden layer: } \mathbb{R}^m}]$$



$\rightarrow$ The hidden layer *learns* a representation so that the data is linearly separable

Visualising the hidden layer

Effects of adding a hidden layer

Input vector $\boldsymbol{x}$ undergoes *linear transformation* by multiplication with weight matrix $\boldsymbol{W}$. The bias term $\boldsymbol{w}_0$ causes a *translation*.

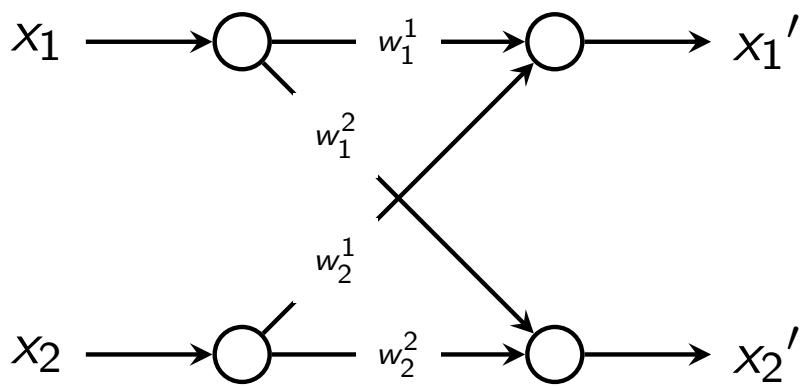
$$y = \sigma[w_0^{(2)} + \boldsymbol{w}^{(2)} \cdot \underbrace{\sigma(\boldsymbol{w}_0^{(1)} + \boldsymbol{W}^{(1)} \cdot \boldsymbol{x})}_{\tilde{\boldsymbol{x}}}]$$

Visualising the hidden layer

Effects of adding a hidden layer

Input vector $\boldsymbol{x}$ undergoes *linear transformation* by multiplication with weight matrix $\boldsymbol{W}$. The bias term $\boldsymbol{w}_0$ causes a *translation*.

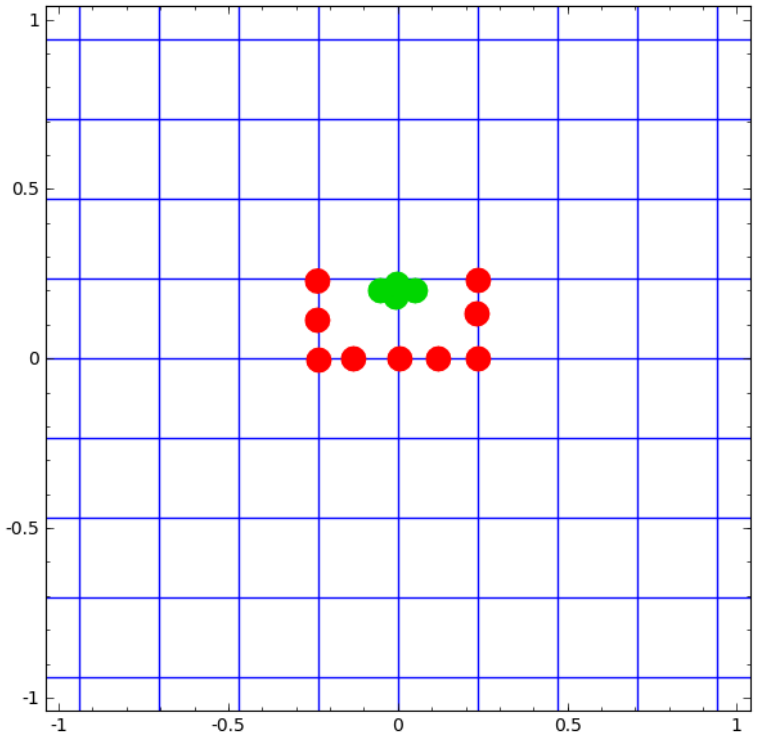
$$\tilde{\boldsymbol{x}} = \sigma(\boldsymbol{w}_0^{(1)} + \boldsymbol{W}^{(1)} \cdot \boldsymbol{x}) \quad \text{with } e.g. \ \boldsymbol{W} = \begin{pmatrix} w_1^1 & w_1^2 \\ w_2^1 & w_2^2 \end{pmatrix}$$



Visualising the hidden layer

1. Linear transformation: $\boldsymbol{W}\boldsymbol{x}$
2. Translation: $\boldsymbol{w}_0$
3. Non-linear activation: σ

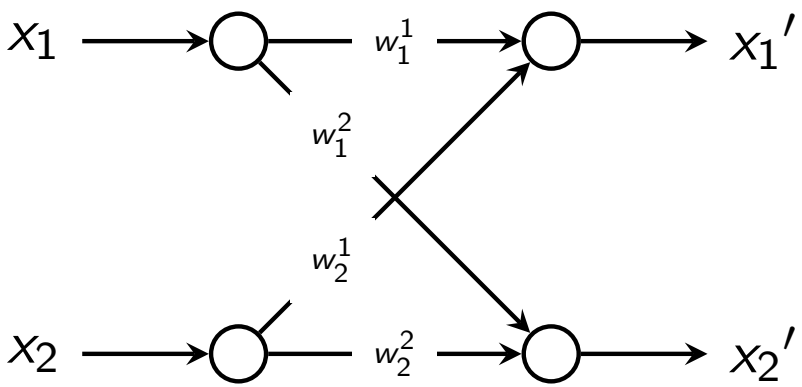
$$\tilde{\boldsymbol{x}} = \boldsymbol{x}$$

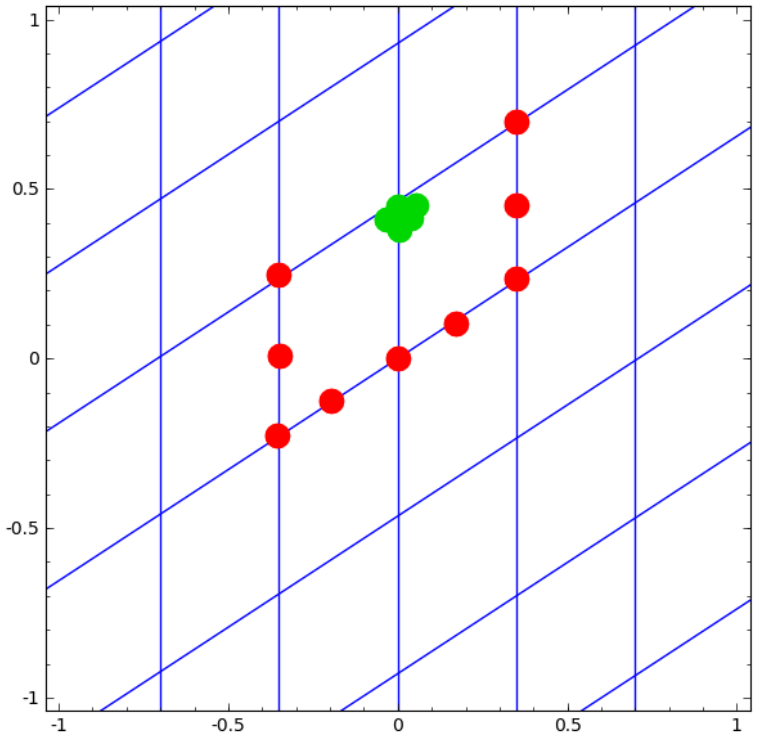


Visualising the hidden layer

1. Linear transformation: $\boldsymbol{W}\boldsymbol{x}$
2. Translation: $\boldsymbol{w}_0$
3. Non-linear activation: σ

$$\tilde{\boldsymbol{x}} = (\boldsymbol{W}^{(1)} \cdot \boldsymbol{x})$$

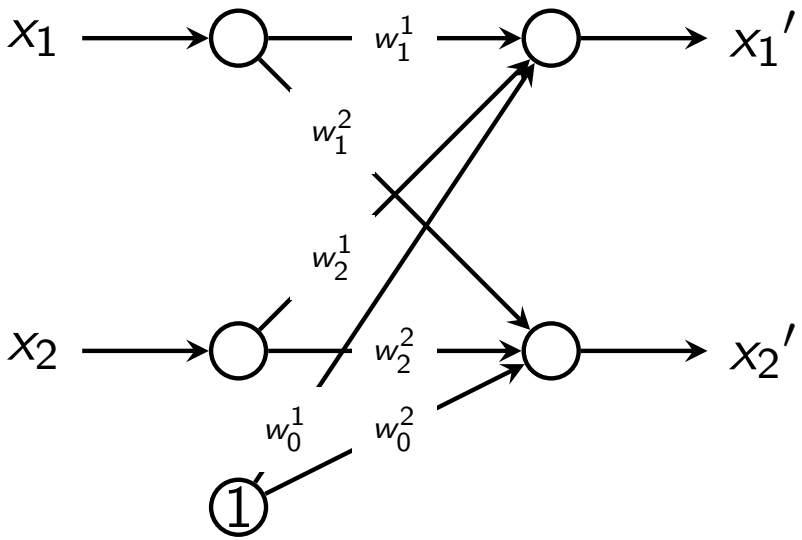


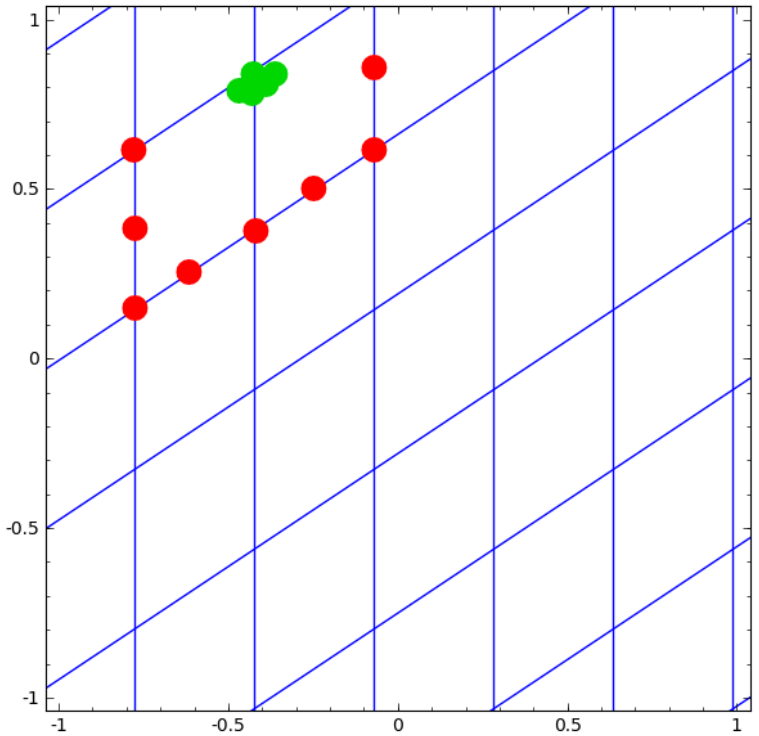


Visualising the hidden layer

1. Linear transformation: $\boldsymbol{W}\boldsymbol{x}$
2. Translation: $\boldsymbol{w}_0$
3. Non-linear activation: σ

$$\tilde{\boldsymbol{x}} = (\boldsymbol{w}_0^{(1)} + \boldsymbol{W}^{(1)} \cdot \boldsymbol{x})$$

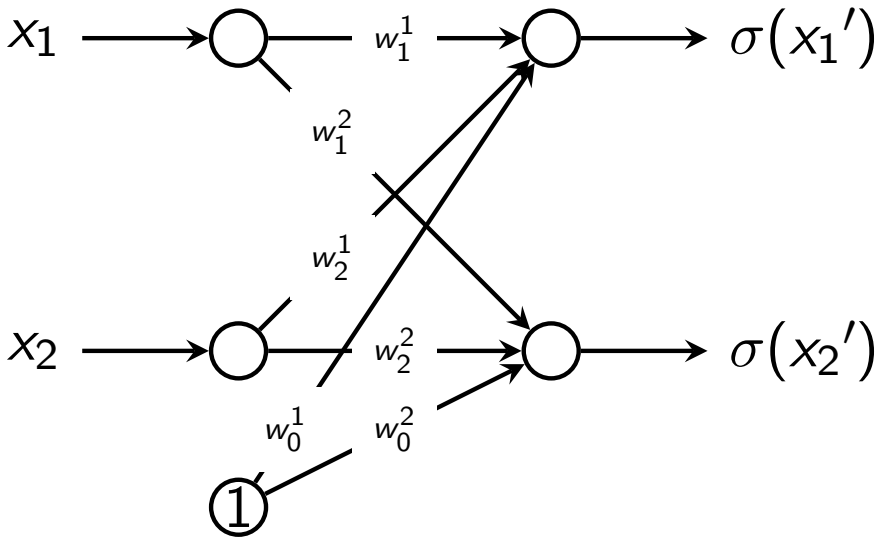


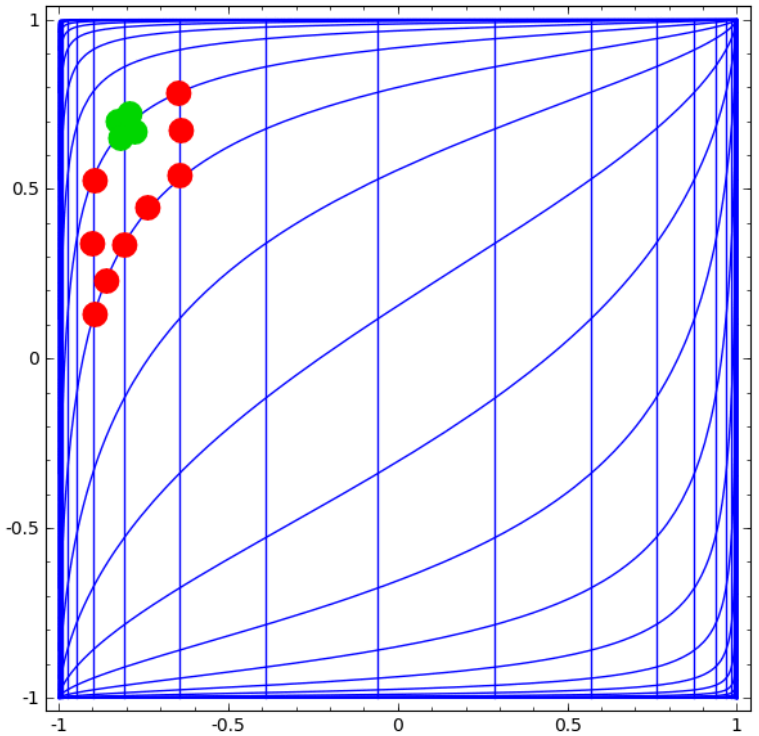


Visualising the hidden layer

1. Linear transformation: $\boldsymbol{W}\boldsymbol{x}$
2. Translation: $\boldsymbol{w}_0$
3. Non-linear activation: σ

$$\tilde{\boldsymbol{x}} = \sigma(\boldsymbol{w}_0^{(1)} + \boldsymbol{W}^{(1)} \cdot \boldsymbol{x})$$

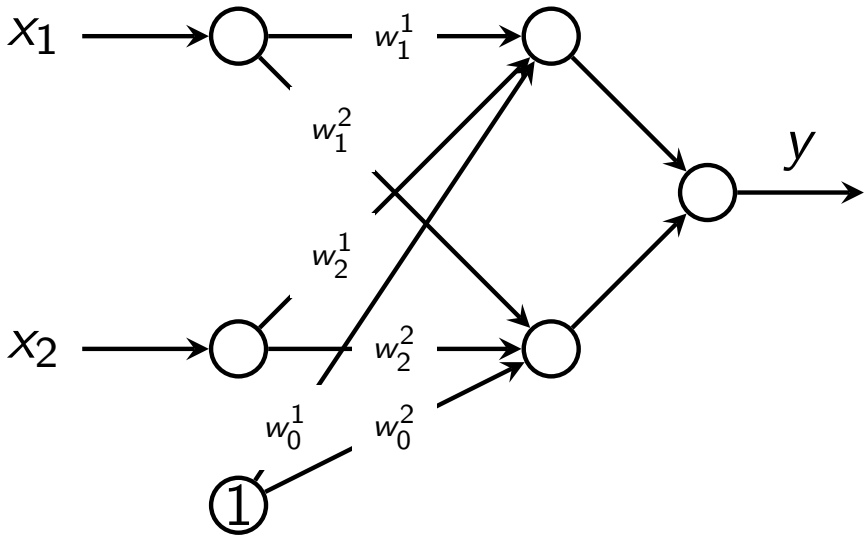


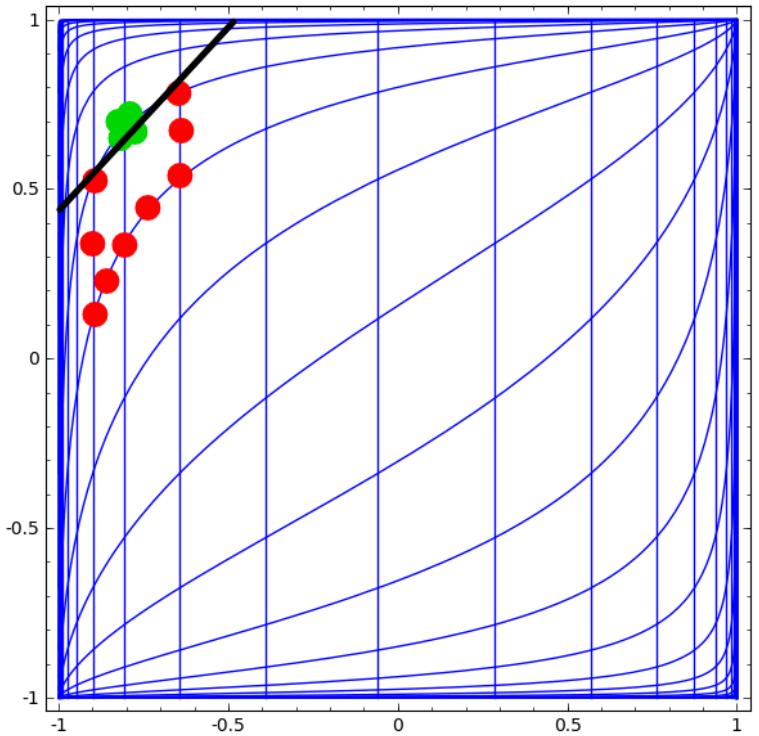


Visualising the hidden layer

1. Linear transformation: $\boldsymbol{W}\boldsymbol{x}$
2. Translation: $\boldsymbol{w}_0$
3. Non-linear activation: σ

$$y = \sigma(w_0^{(2)} + \boldsymbol{w}^{(2)} \cdot \tilde{\boldsymbol{x}})$$





Update weights via backpropergation

Backpropergation

Adapt weights by going backwards in the network

- Initialize weigts
- Evaluate $y_{predict} = f(\boldsymbol{x})$ and calculate $\Delta = (y_{true} - y_{predict})$
- Update weights: Weight adaption often denoted via loss $L \propto \Delta$
  $\rightarrow$ minimize loss $\rightarrow$ find best fit

Update weights via backpropergation

Backpropergation

Adapt weights by going backwards in the network

- Initialize weigts
- Evaluate $y_{predict} = f(\boldsymbol{x})$ and calculate $\Delta = (y_{true} - y_{predict})$
- Update weights: Weight adaption often denoted via loss $L \propto \Delta$ $\rightarrow$ minimize loss $\rightarrow$ find best fit

Update weights via backpropergation

Backpropergation

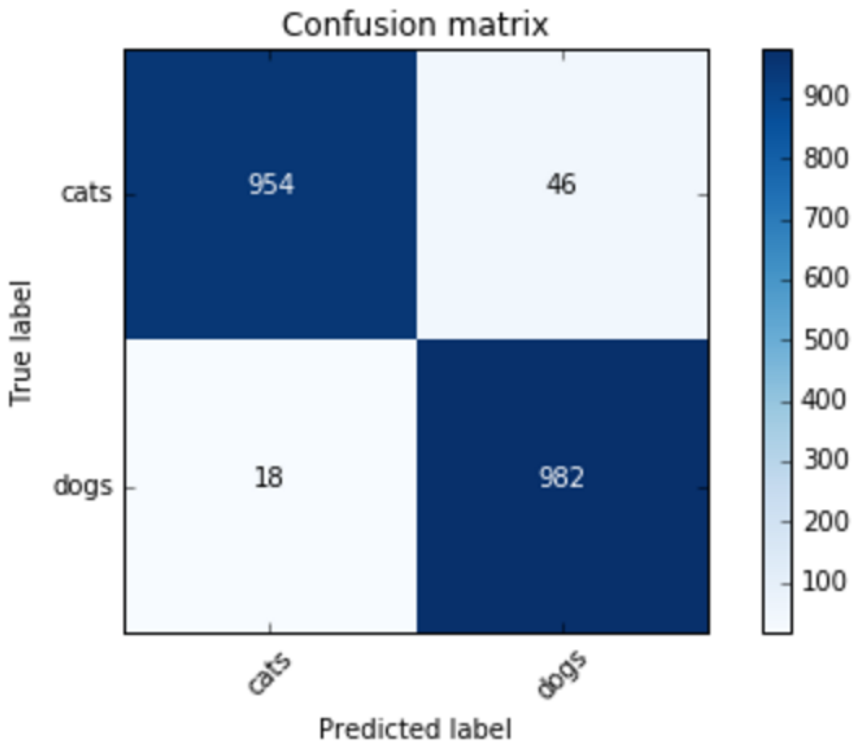
Adapt weights by going backwards in the network

- Initialize weigts
- Evaluate $y_{predict} = f(\boldsymbol{x})$ and calculate $\Delta = (y_{true} - y_{predict})$
- Update weights: Weight adaption often denoted via loss $L \propto \Delta$
  $\rightarrow$ minimize loss $\rightarrow$ find best fit

Quantify MVA output

Evaluate classifier performance

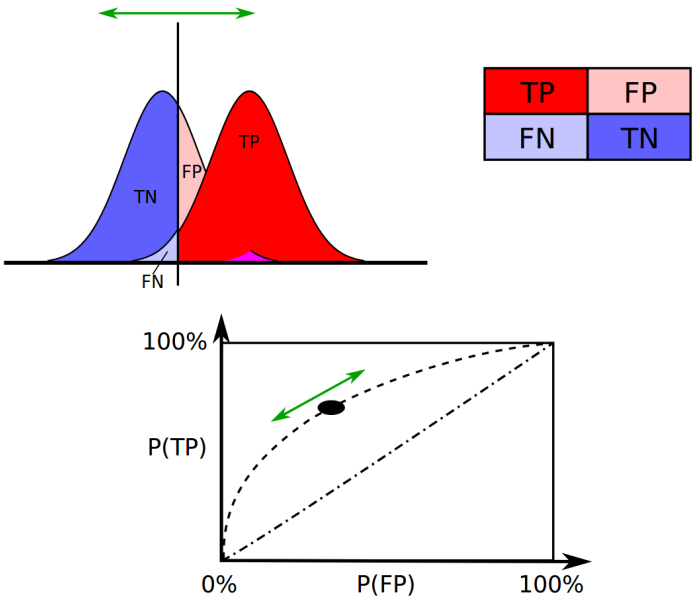
- Compare predicted class labels with the true ones → confusion matrix



Quantify MVA output

Evaluate classifier performance

- ROC curve: continuously scan y & plot background acceptance (FPR) vs. signal efficiency (TPR)



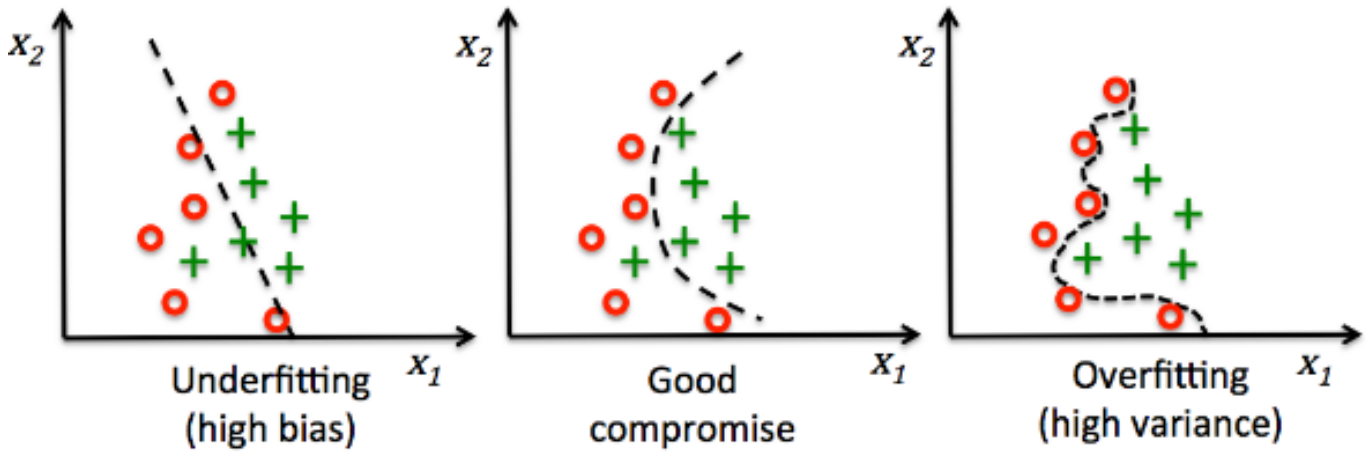
$$\text{FPR} = \text{FP}/(\text{FP}+\text{TN})$$

$$\text{TPR} = \text{TP}/\ (\text{TP} + \text{FN})$$

ROC-AUC: probability that classifier ranks randomly chosen positive sample higher than randomly chosen negative one

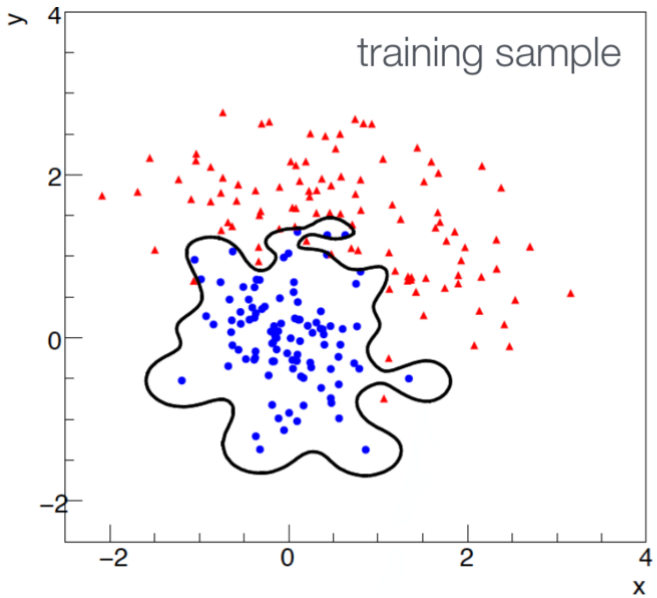
Bias-variance tradeof

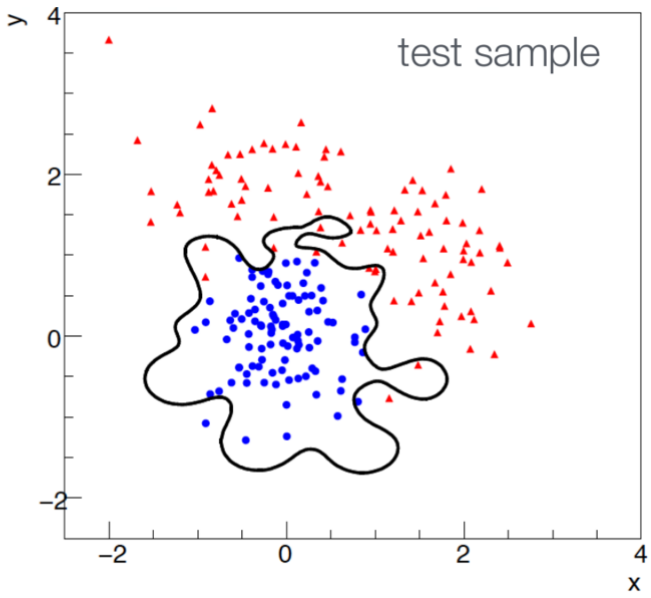
- Small variance: Classifiers with low degrees of freedom are less prone to statistical fluctuations
  $\rightarrow$ different training samples result in similar classification boundaries
- However: if data contain features that a model with few degrees of freedom cannot describe, a bias is introduced



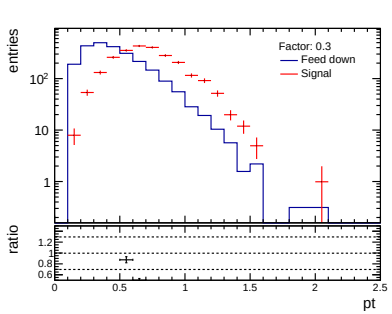
Bias-variance tradeof

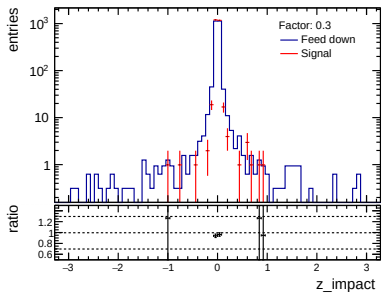
- Small variance: Classifiers with low degrees of freedom are less prone to statistical fluctuations
  $\rightarrow$ different training samples result in similar classification boundaries
- However: if data contain features that a model with few degrees of freedom cannot describe, a bias is introduced

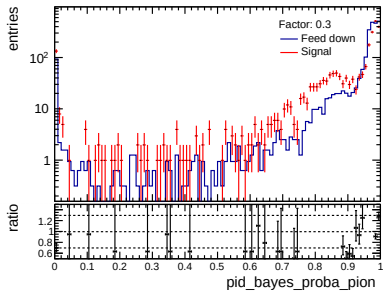




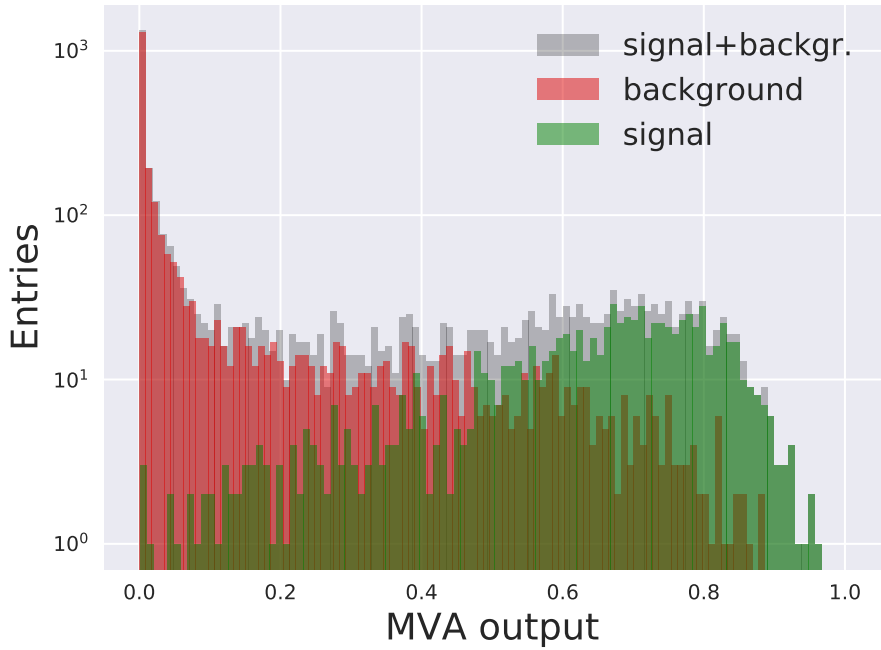
Results



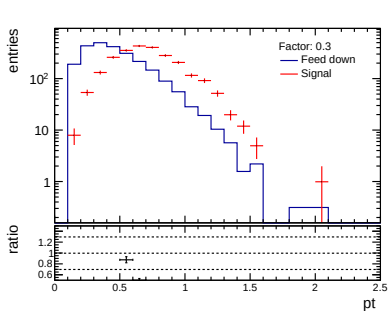


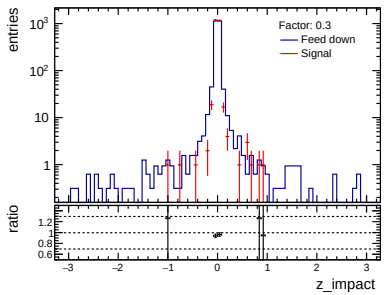


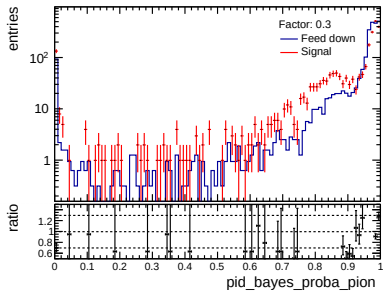
$$\mathbb{R}^n \rightarrow \mathbb{R}$$



Results

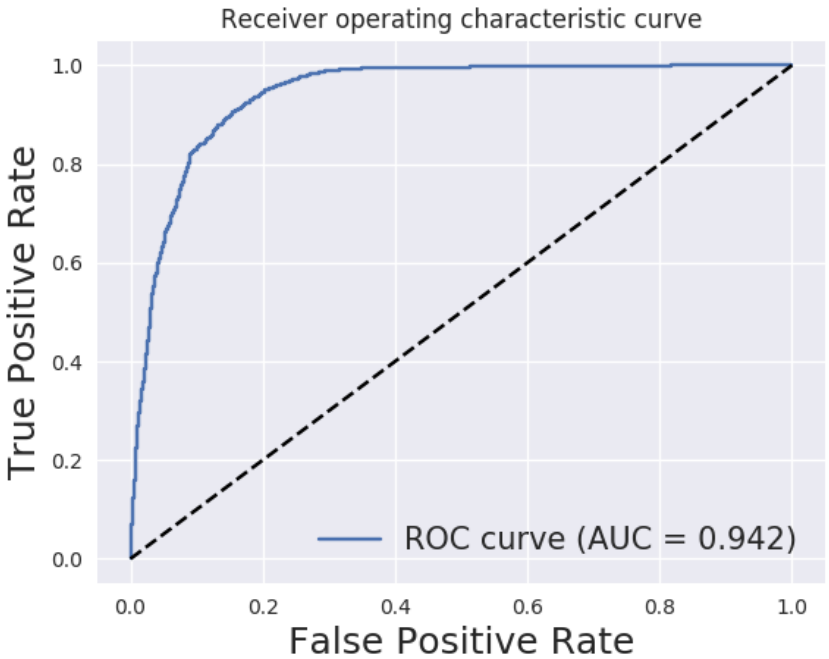




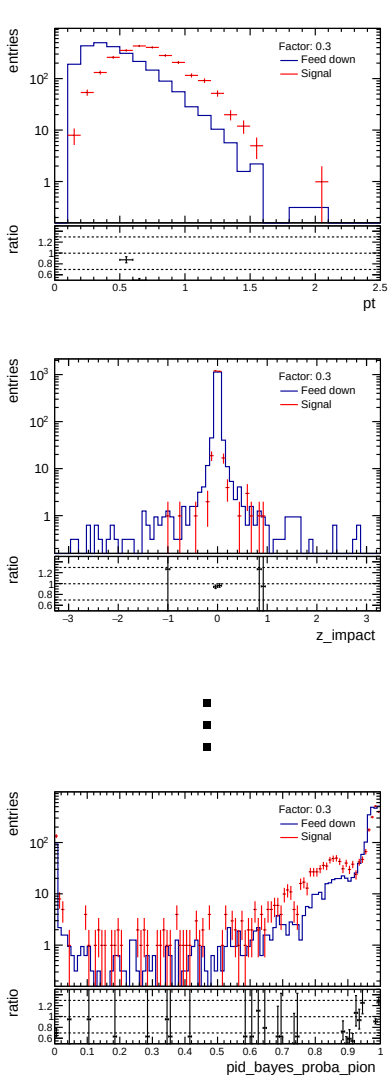


$$\mathbb{R}^n \rightarrow \mathbb{R}$$

ROC train

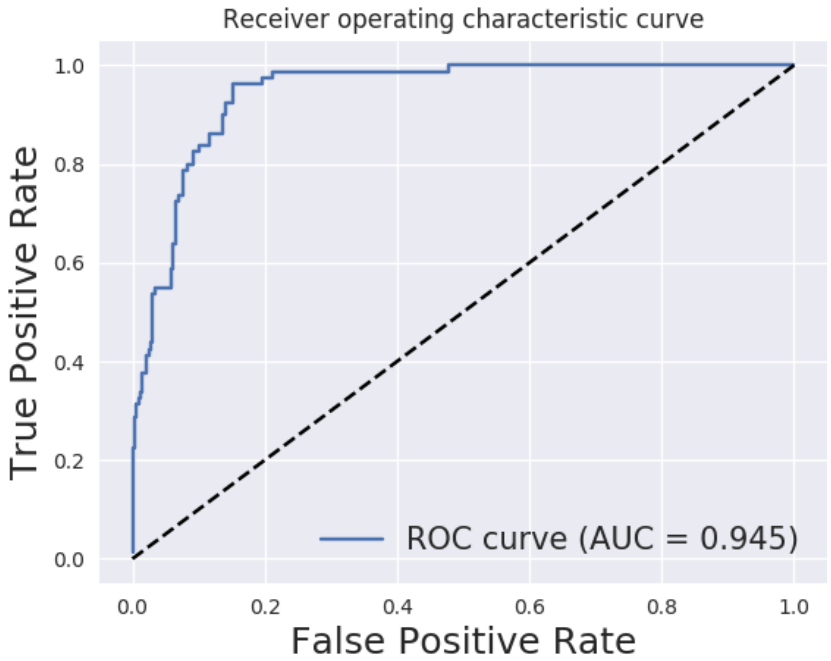


Results

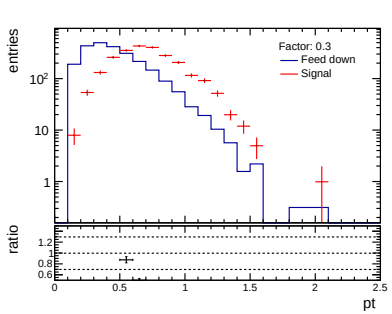


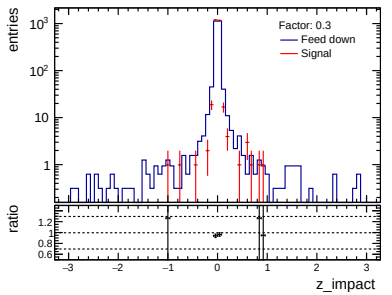
$\mathbb{R}^n \rightarrow \mathbb{R}$

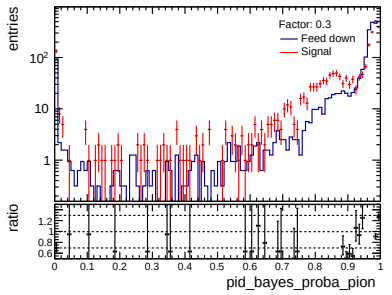
ROC test



Results

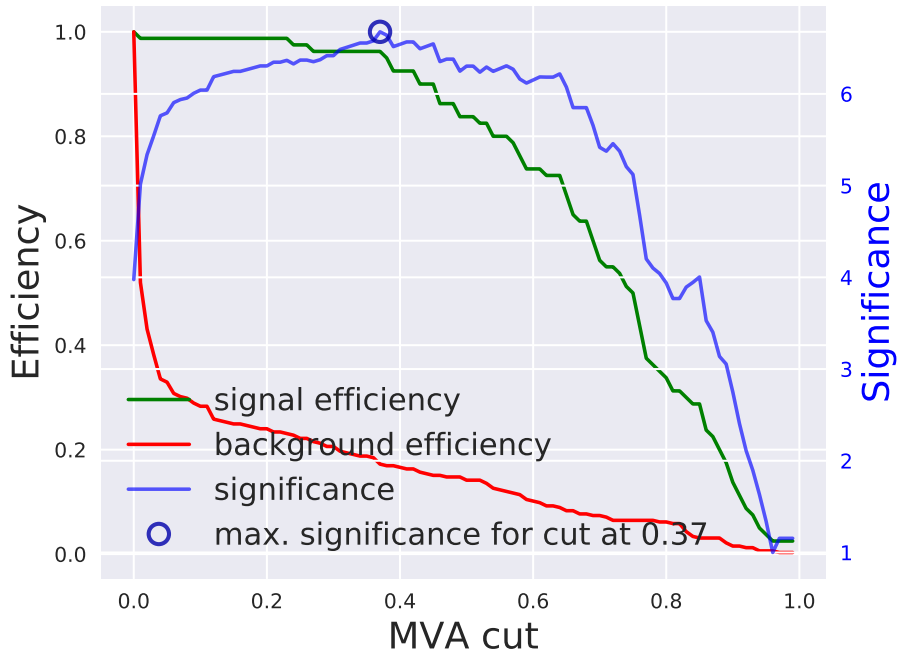




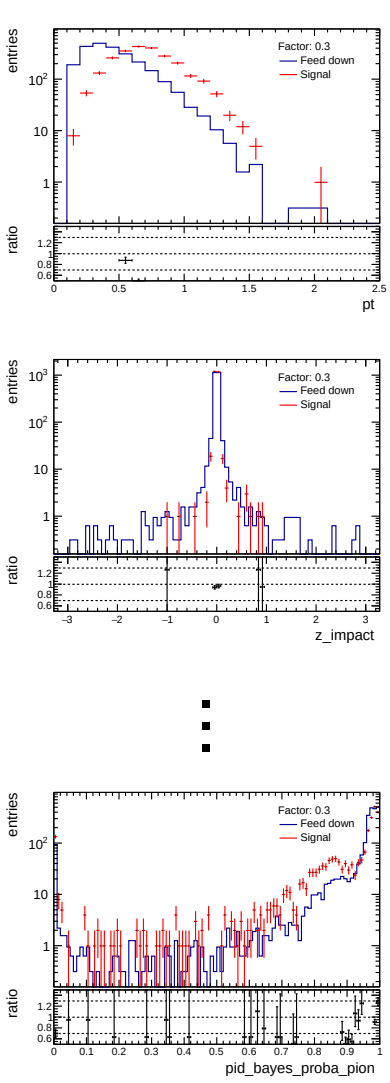


$$\mathbb{R}^n \to \mathbb{R}$$

Significance $S = N_{sig} / \sqrt{N_{sig} + N_{BG}}$

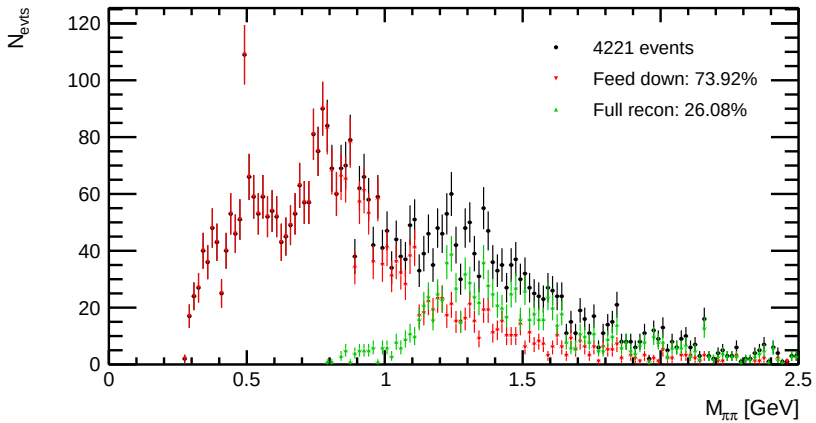


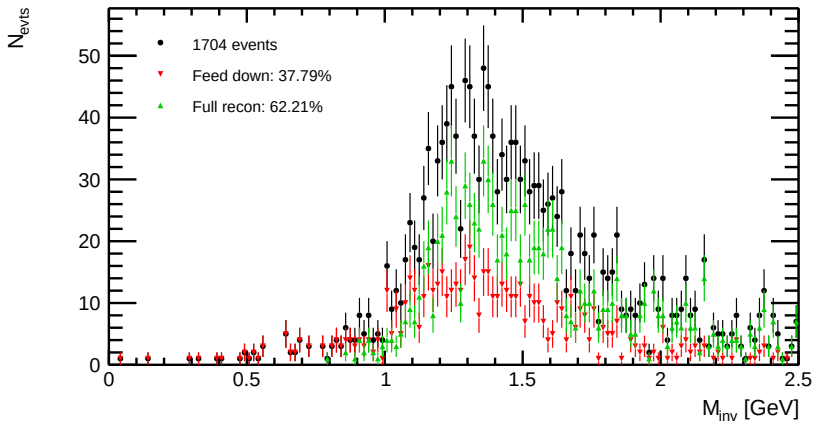
Results



$\mathbb{R}^n \rightarrow \mathbb{R}$

no MVA cut





with MVA cut