Summer School ML

IV Linear Models

Prof. Dr.-Ing. Janis Keuper

KI Labor Südbaden

INSTITUTE FOR MACHINE LEARNING AND ANALYTICS

Hochschule Offenburg
offenburg.university

KI Labor Südbaden

Basic Types of Machine Learning Algorithms

Supervised Learning

Unsupervised Learning

Reinforcement Learning

- Labeled data
- Direct and quantitative evaluation

- Learn model from „ground truth“ examples
- Predict unseen examples

Recall Classification

Supervised Learning: Annotated Training Data

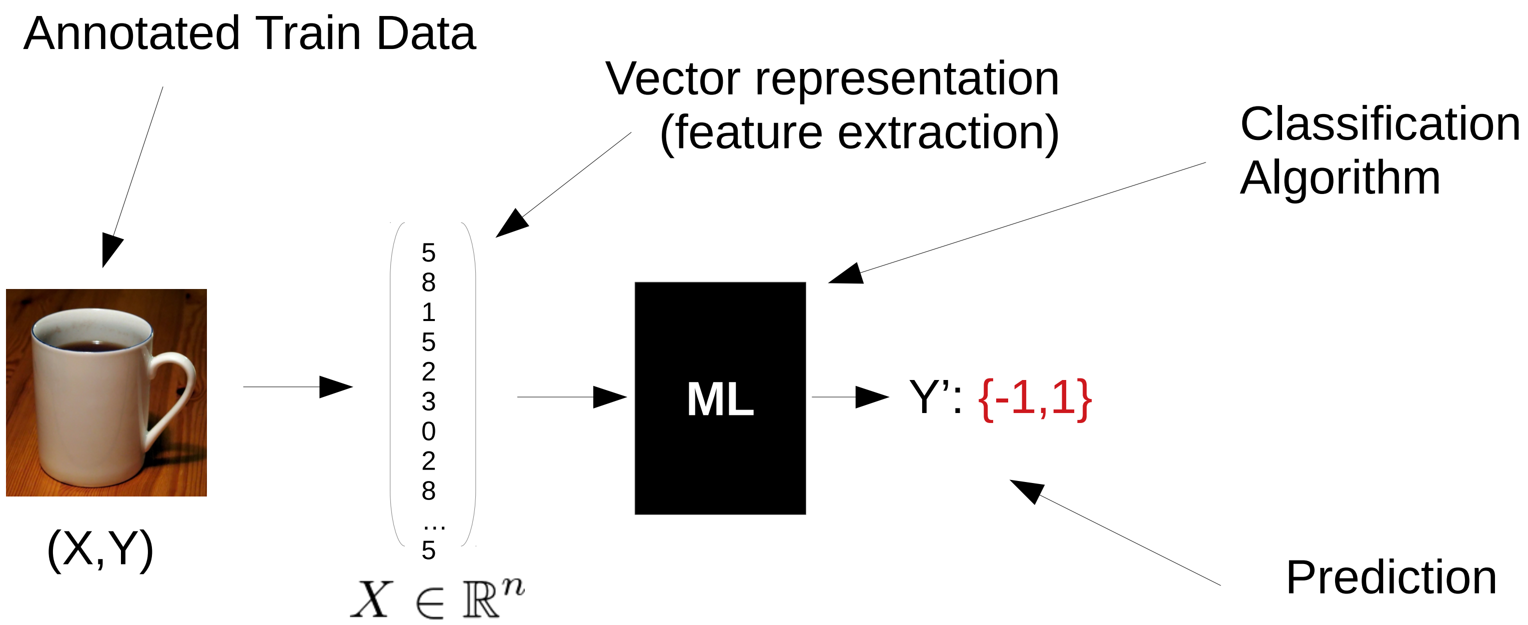
(X,Y)

$$\begin{pmatrix} 5 \\ 8 \\ 1 \\ 5 \\ 2 \\ 3 \\ 0 \\ 2 \\ 8 \\ \dots \\ 5 \end{pmatrix} \rightarrow \boxed{\text{ML}} \rightarrow Y': \{-1,1\}$$

$X \in \mathbb{R}^n$

Recall Classification

Supervised Learning: Annotated Training Data



Recall Classification

LEARNING: is a optimization problem → Finding the best function separating

$$min(\|f(X,Y),Y'\|)$$

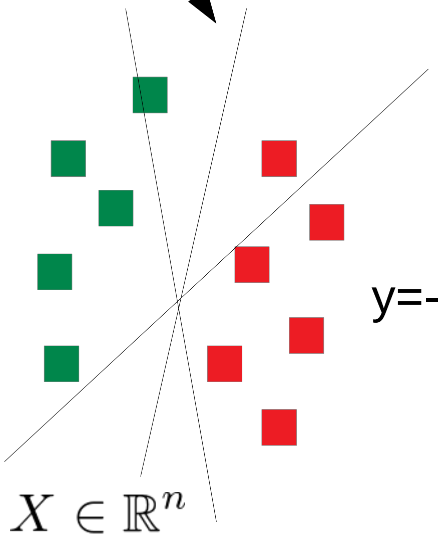
(X,Y)

$$\begin{pmatrix} 5 \\ 8 \\ 1 \\ 5 \\ 2 \\ 3 \\ 0 \\ 2 \\ 8 \\ \dots \\ 5 \end{pmatrix}$$

$X \in \mathbb{R}^n$

ML → Y’: {-1,1}

y=1



y=-1

$X \in \mathbb{R}^n$

Example: MNIST

KI Labor Südbaden

The MNIST hand written digits classification Problem

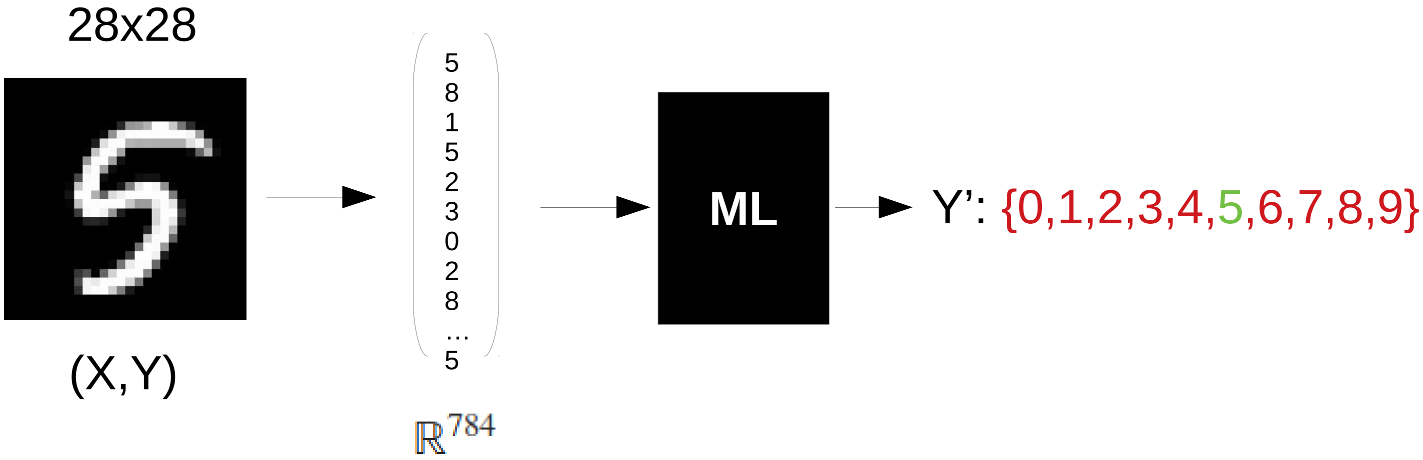
The MNIST database (Modified National Institute of Standards and Technology database) is a large database of handwritten digits that is commonly used for training various image processing systems.

Data specs:

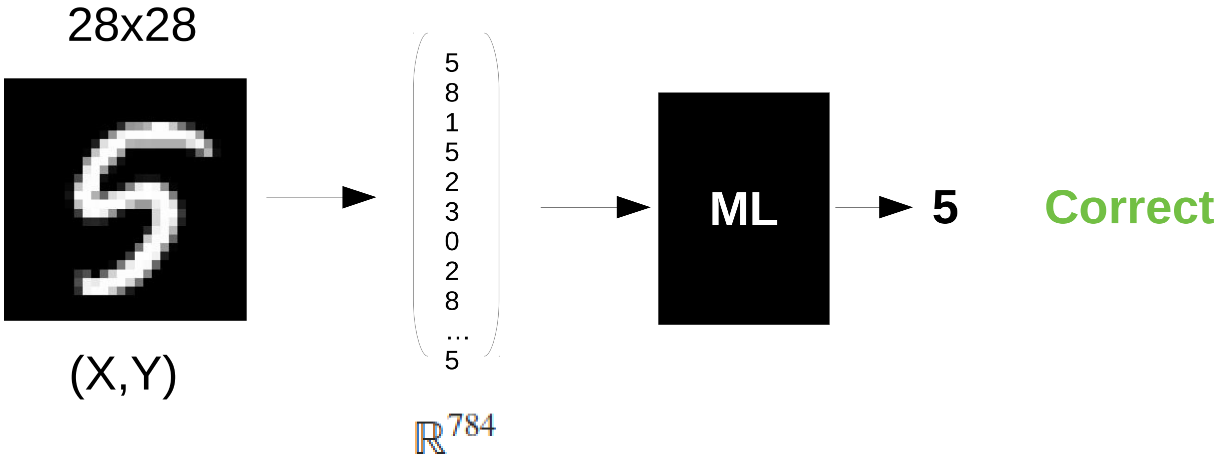
- 10 Classes (digest 0-9)
- 28x28 gray scale images
- 60000 train samples
- 10000 test samples

Example: MNIST

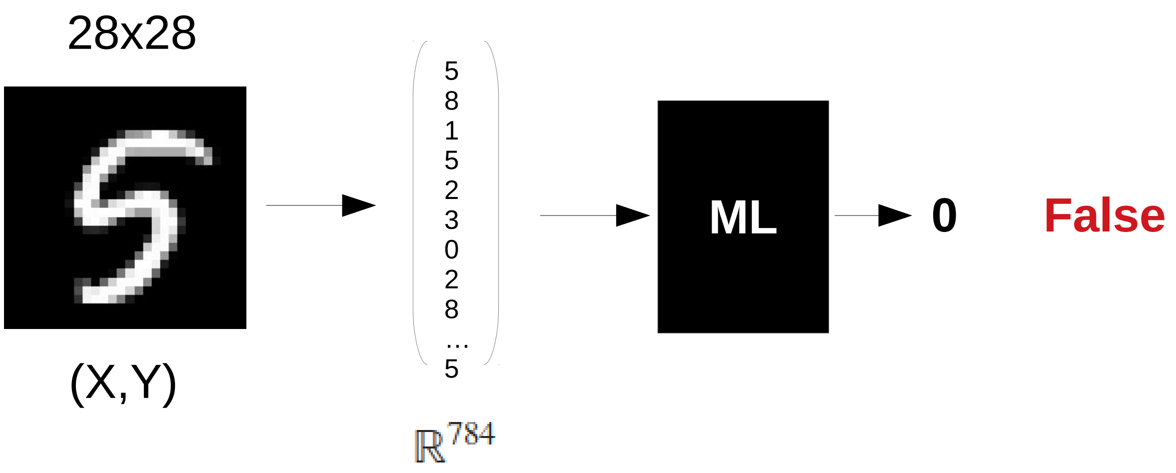
The MNIST hand written digits classification Problem



Evaluation



Evaluation



Recall: Evaluation

Basic evaluation of a model:

Train error: measure of how well the model predicts the given labels

$$Err_{train} := \frac{1}{|X_{train}|} \sum_{x_i \in X_{train}} |f(x_i) - y_i|$$

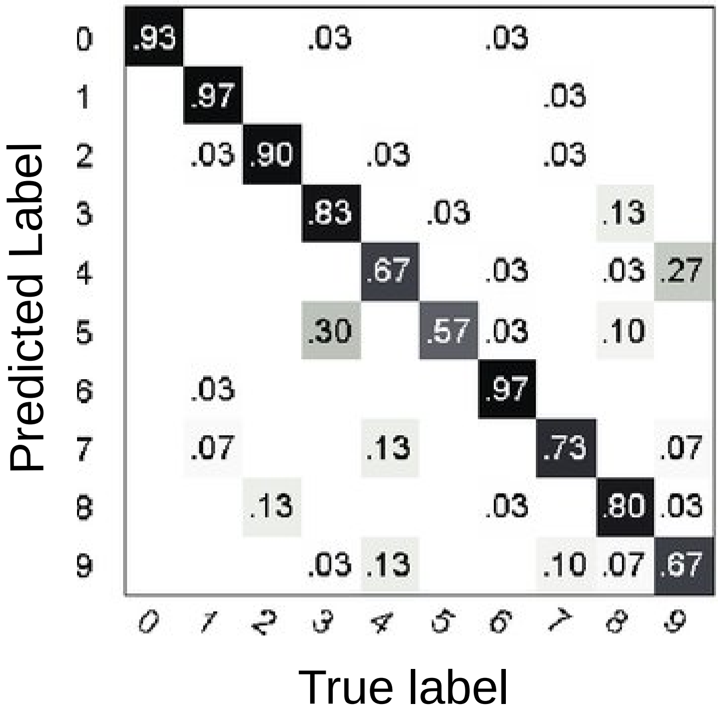
low train error is the necessary condition for a "good" model

Test error: **same as train error: low test error is the sufficient condition**

$$Err_{test} := \frac{1}{|X_{test}|} \sum_{x_i \in X_{test}} |f(x_i) - y_i|$$

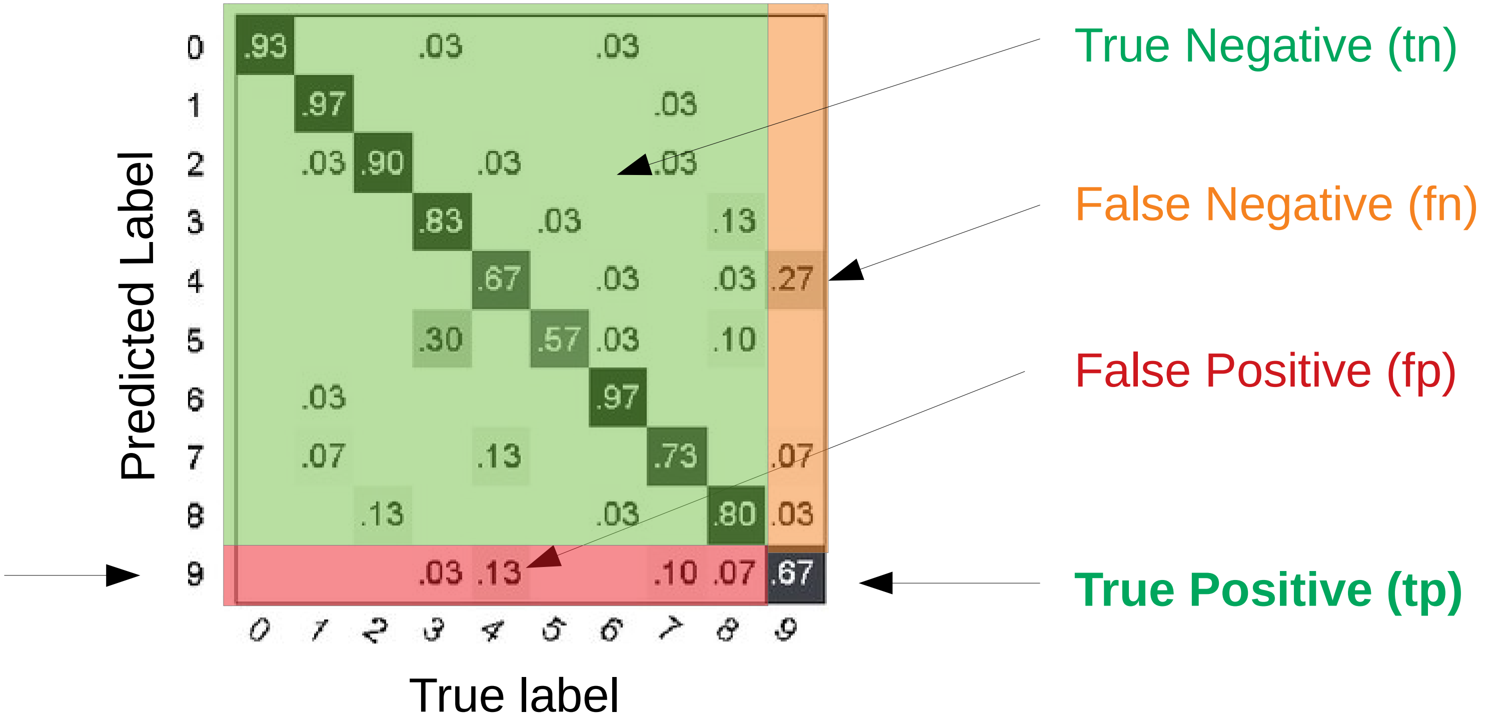
Evaluation

Confusion Matrix and True and False Positives/Negatives



Evaluation

Confusion Matrix and True and False Positives/Negatives
Example for true digit “9”



Evaluation

Accuracy:

Most commonly used error metric:

Correct samples

$$\text{Accuracy} = \frac{tp + tn}{tp + tn + fp + fn}$$

All samples

Evaluation

Problems with Accuracy Unbalanced classes:

If the prior probability of one class is much higher than others, *fp* will have little impact.

$$\text{Accuracy} = \frac{tp + tn}{tp + tn + fp + fn}$$

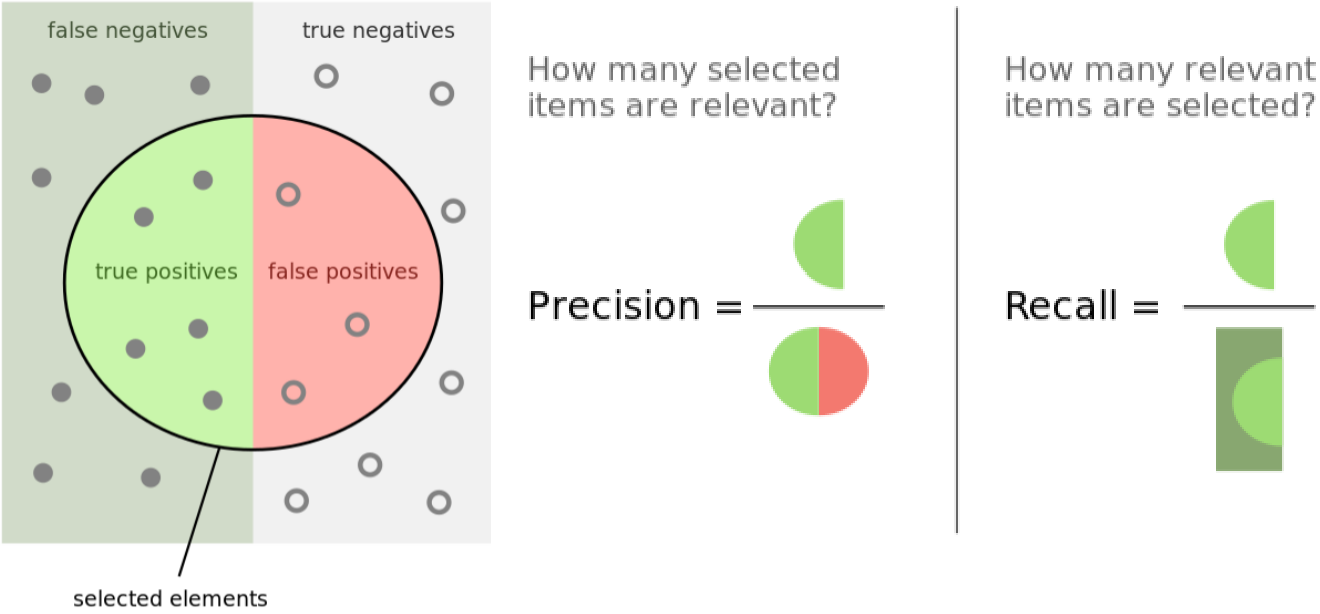
Extreme example: if 90% of the digits are “1”, classifying every digit to “1” will will have 90% accuracy!

Evaluation

Precision and Recall

$$\text{Precision} = \frac{tp}{tp + fp}$$

$$\text{Recall} = \frac{tp}{tp + fn}$$



[image by wikipedia]

Evaluation

F-Measure or balanced F-score is the harmonic mean of precision and recall:

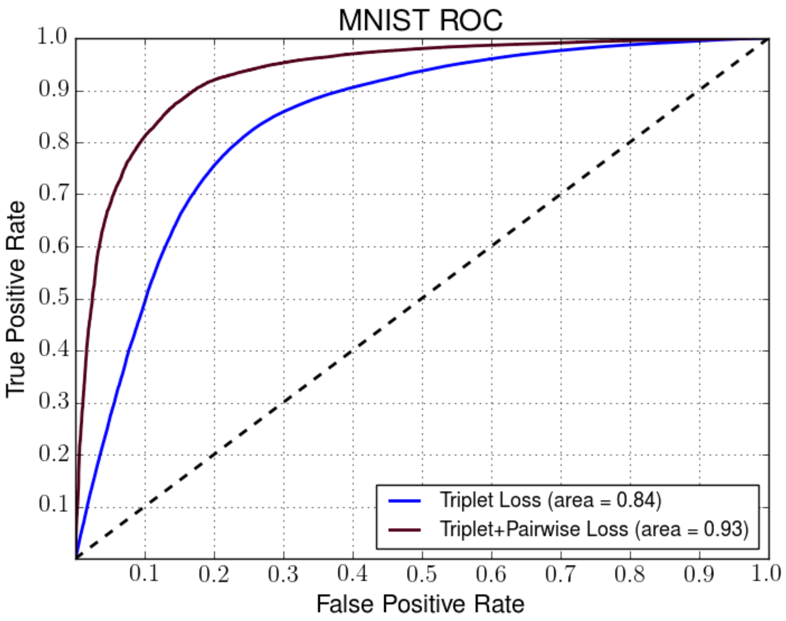
$$F = 2 \cdot \frac{\text{precision} \cdot \text{recall}}{\text{precision} + \text{recall}}$$

Evaluation

Receiver Operating Characteristic Curve

A receiver operating characteristic curve, or ROC curve, is a graphical plot that illustrates the diagnostic ability of a **binary classifier** system as its discrimination threshold is varied.

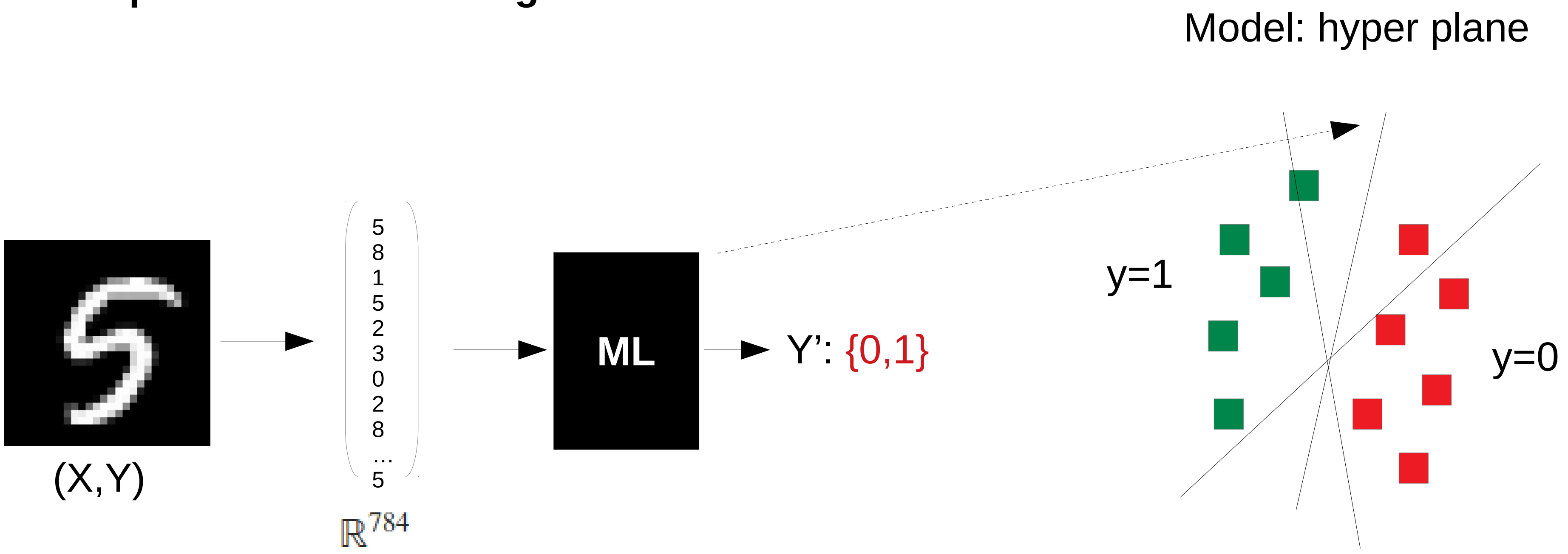
Example: comparing two different models



Linear Classifier

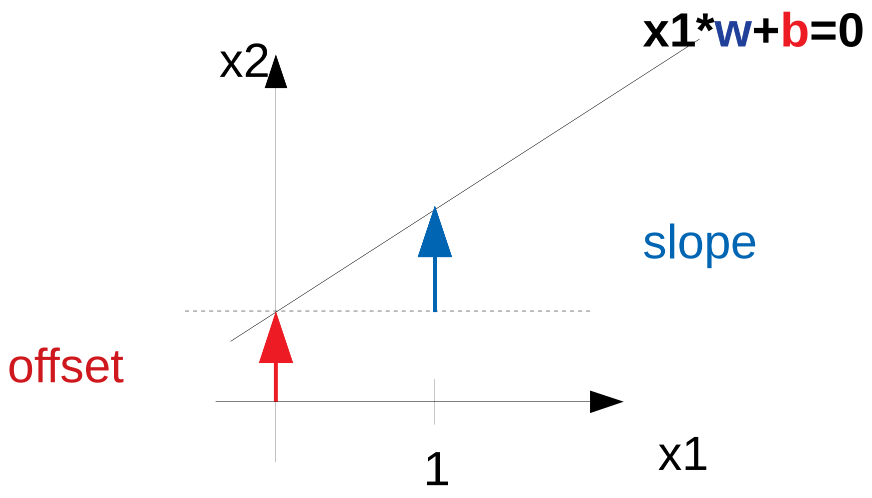
A Simple Linear Model: binary classification

Example: “5” vs “other digit”

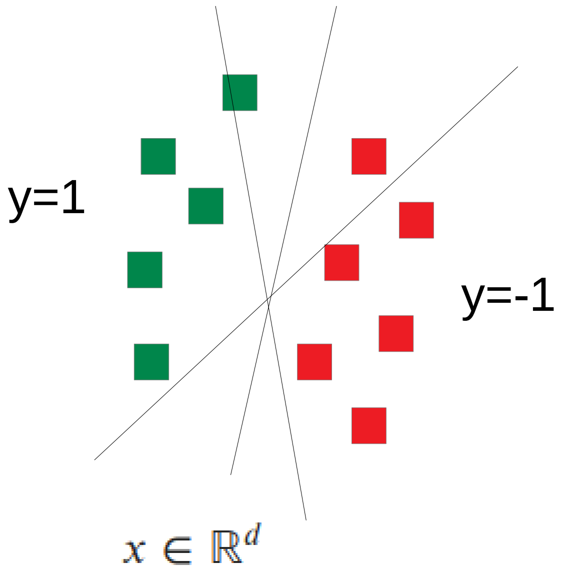


Linear Classifier

Parameterization of a hyper plane (here 2D)



Model: hyper plane



Linear Classifier

A Simple Linear Model: binary classification

Parameterization of prediction function f with d-dimensional data as:

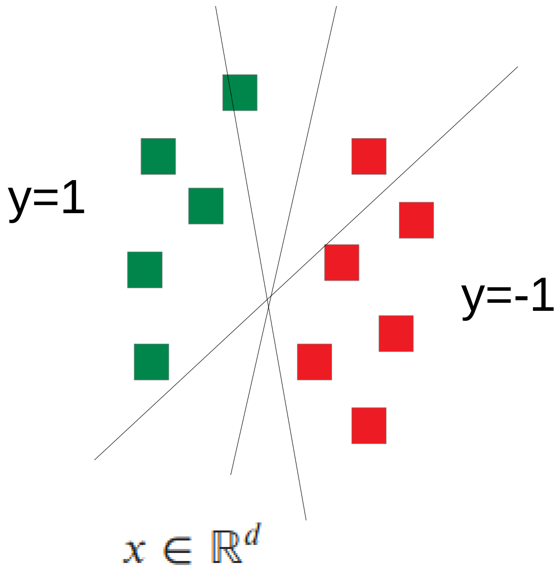
$$f(x) = y' = sgn(w^T x + b) = sgn(\sum_{j=0}^{d} x_j w_j + b_j)$$

With data samples $x \in \mathbb{R}^d$

Model parameters $w \in \mathbb{R}^d$

$$\operatorname{sgn}(x) := \begin{cases} +1 & \text{falls} \quad x > 0 \\ 0 & \text{falls} \quad x = 0 \\ -1 & \text{falls} \quad x < 0 \end{cases}$$

Model: hyper plane



Linear Classifier

A Simple Linear Model: binary classification

Parameterization of prediction function f with d-dimensional data as:

$$f(x) = y' = sgn(w^T x + b) = sgn(\sum_{j=0}^{d} x_j w_j + b_j)$$

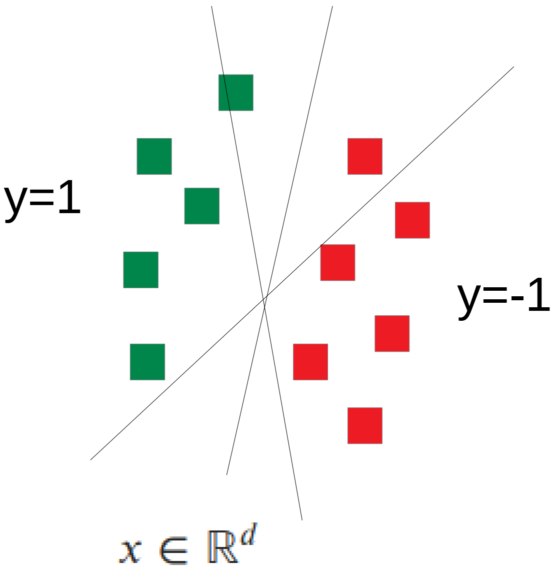
With data samples $x \in \mathbb{R}^d$

Model parameters $w \in \mathbb{R}^d$

$$\operatorname{sgn}(x) := \begin{cases} +1 & \text{falls} \quad x > 0 \\ 0 & \text{falls} \quad x = 0 \\ -1 & \text{falls} \quad x < 0 \end{cases}$$

How to find the parameters?

Model: hyper plane



Linear Classifier

Optimization problem to find parameters

$$\arg\min_{w} \sum_{i=0}^{N} L(y_i, w^T x_i)$$

With a differential *Loss* function like

$$L(y = 1, y') := \frac{1}{1+e^{-y'}}$$

$$L(y = 0, y') := 1 - L(y = 1, y')$$

Linear Classifier

Optimization problem to find parameters

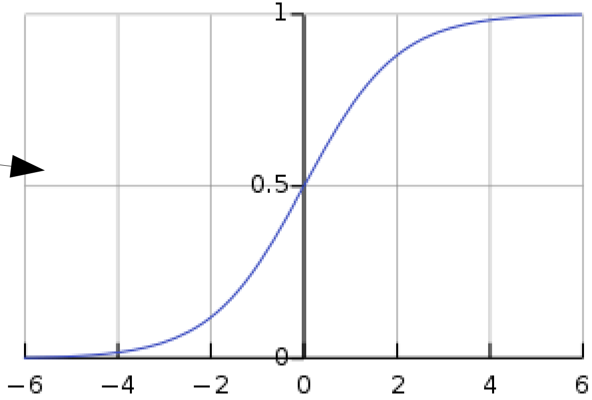
$$\arg\min_{w} \sum_{i=0}^{N} L(y_i, w^T x_i)$$

With a differential *Loss* function like: **logistic function**

Only one of many possible Loss functions, but common choice

$$L(y = 1, y') := \frac{1}{1+e^{-y'}}$$

$$L(y = 0, y') := 1 - L(y = 1, y')$$



- **pseudo probability:** Out put always between 0 and 1
- Apply **threshold** function on probability that class label =1

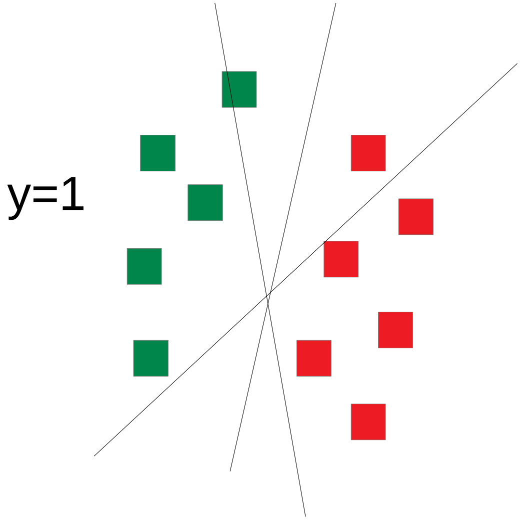
Gradient Descent Optimization

Goal: find w to minimize $\arg\min_{w} \sum_{i=0}^{N} L(y_i, w^T x_i)$

Gradient Descent Optimization

Goal: find w to minimize $\arg\min_{w} \sum_{i=0}^{N} L(y_i, w^T x_i)$

2D Example:



$x \in \mathbb{R}^2$

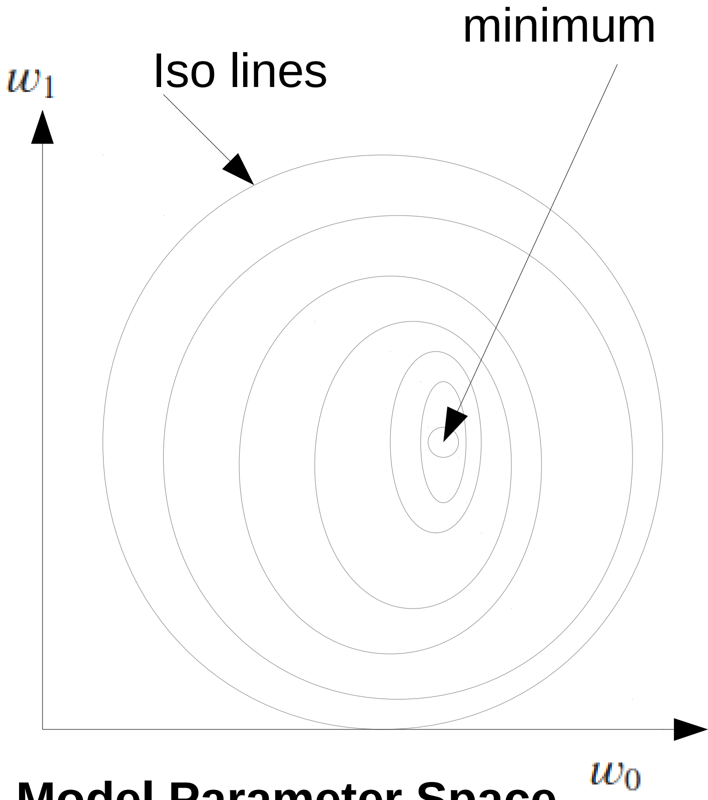
Feature Space

Gradient Descent Optimization

Goal: find w to minimize $\arg\min_{w} \sum_{i=0}^{N} L(y_i, w^T x_i)$

2D Example:

- How many (and which) parameters do we have to find?
- *L* spans a (loss) surface in the d-dimensional space of the data X **(parameter space)**
- We can evaluate *L* at each point *w*
- We can compute the gradient at each point *w* in *L* *(assuming L to be Lipschitz)*



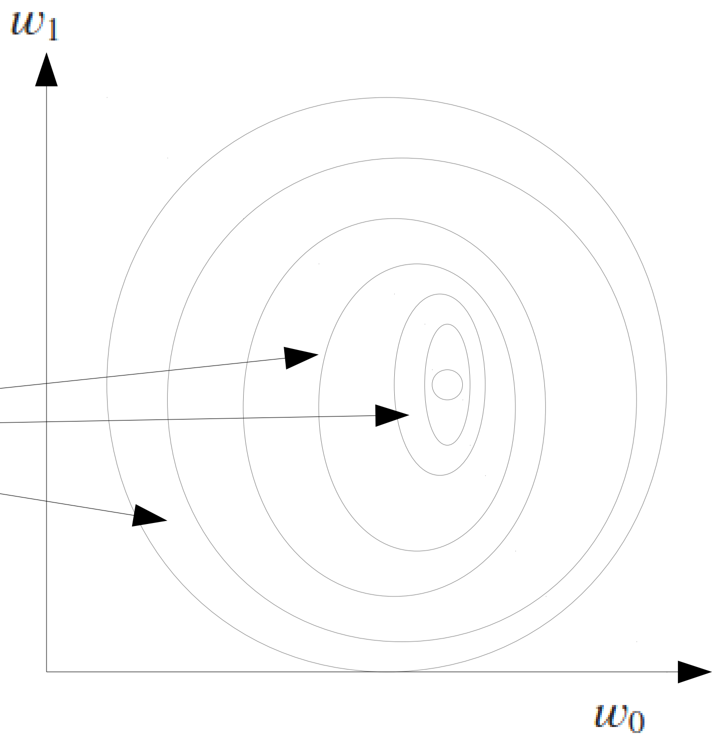
Model Parameter Space

Gradient Descent Optimization

Goal: find w to minimize $\arg\min_{w} \sum_{i=0}^{N} L(y_i, w^T x_i)$

2D Example:

- How many (and which) parameters do we have to find?
- *L* spans a (loss) surface in the d-dimensional space of the data X **(parameter space)**
- We can evaluate *L* at each point *w*
- We can compute the gradient at each point *w* in *L* *(assuming L to be Lipschitz)*



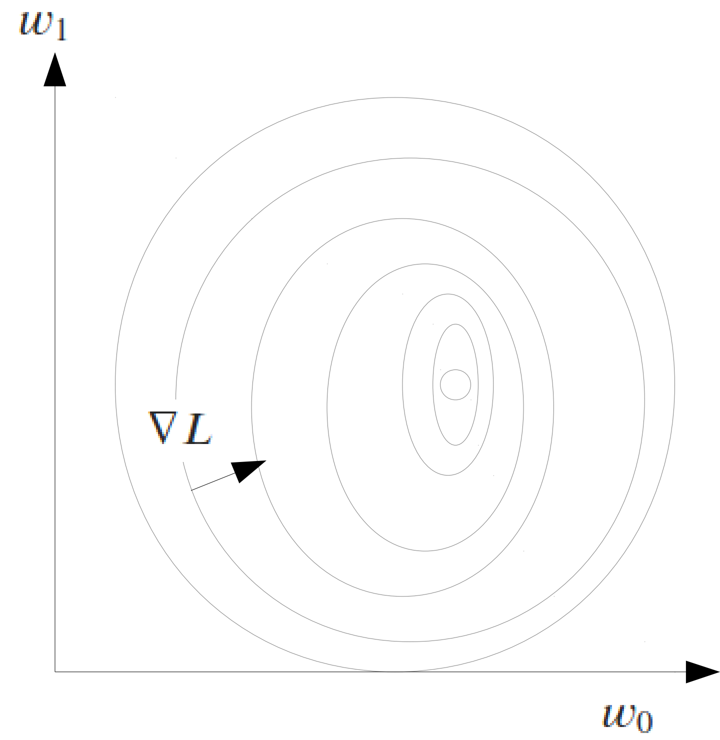
Gradient Descent Optimization

Goal: find w to minimize $\arg\min_{w} \sum_{i=0}^{N} L(y_i, w^T x_i)$

2D Example:

- How many (and which) parameters do we have to find?
- *L* spans a (loss) surface in the d-dimensional space of the data X **(parameter space)**
- We can evaluate *L* at each point *w*
- We can compute the gradient at each point *w* in *L* *(assuming L to be Lipschitz)*

$$\nabla L = \frac{dL}{dw}$$

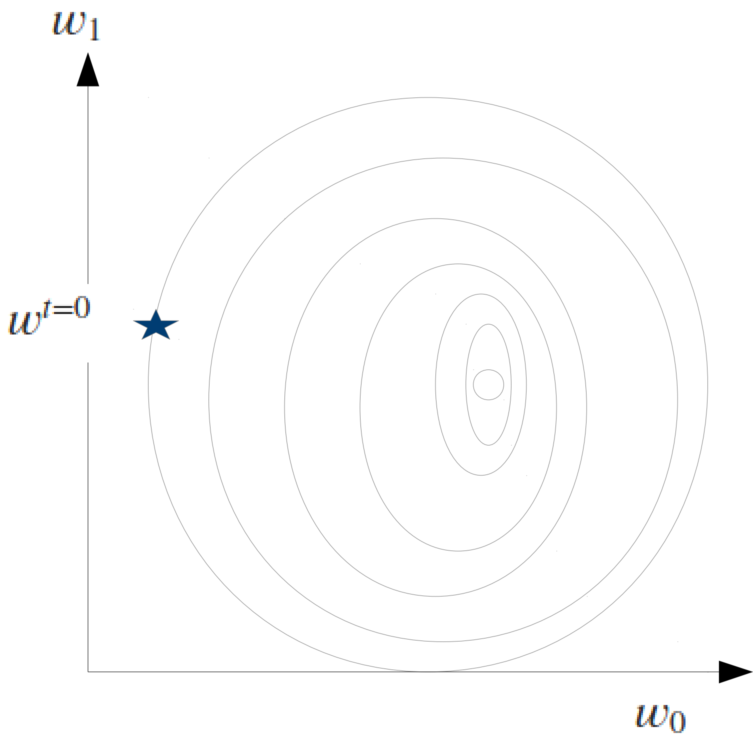


Gradient Descent Optimization

Goal: find w to minimize $\arg\min_{w} \sum_{i=0}^{N} L(y_i, w^T x_i)$

Gradient Descent Algorithm:

I. Start with random $w^{t=0}$



Gradient Descent Optimization

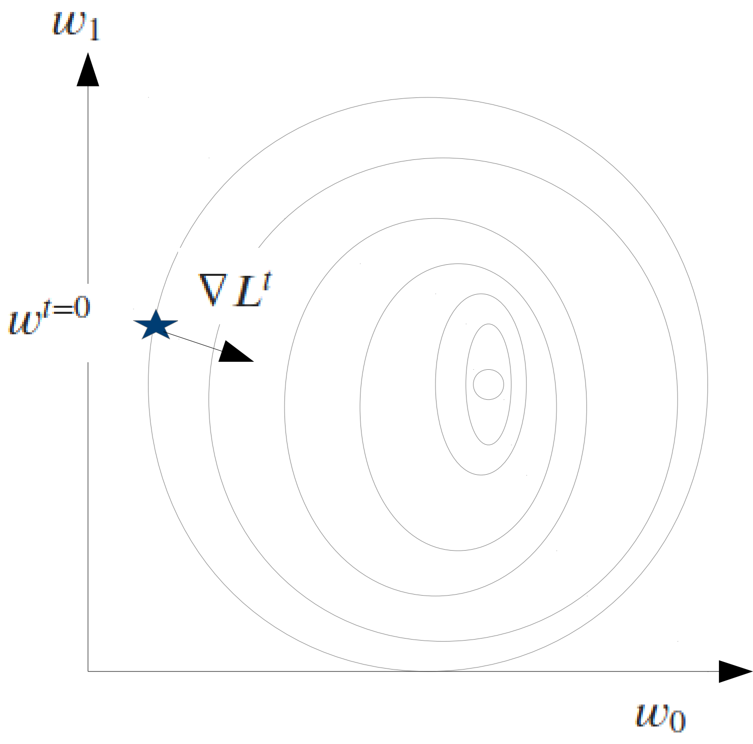
Goal: find w to minimize $\underset{w}{\arg\min} \sum_{i=0}^{N} L(y_i, w^T x_i)$

Gradient Descent Algorithm:

I. Start with random $w^{t=0}$

II. Compute gradient for all training samples

$$\nabla L^t = \sum_{i=0}^{|(X,y)|} \frac{dL(y_i, w^t x_i)}{dw^t}$$



Gradient Descent Optimization

Goal: find w to minimize $\arg\min_{w} \sum_{i=0}^{N} L(y_i, w^T x_i)$

Gradient Descent Algorithm:

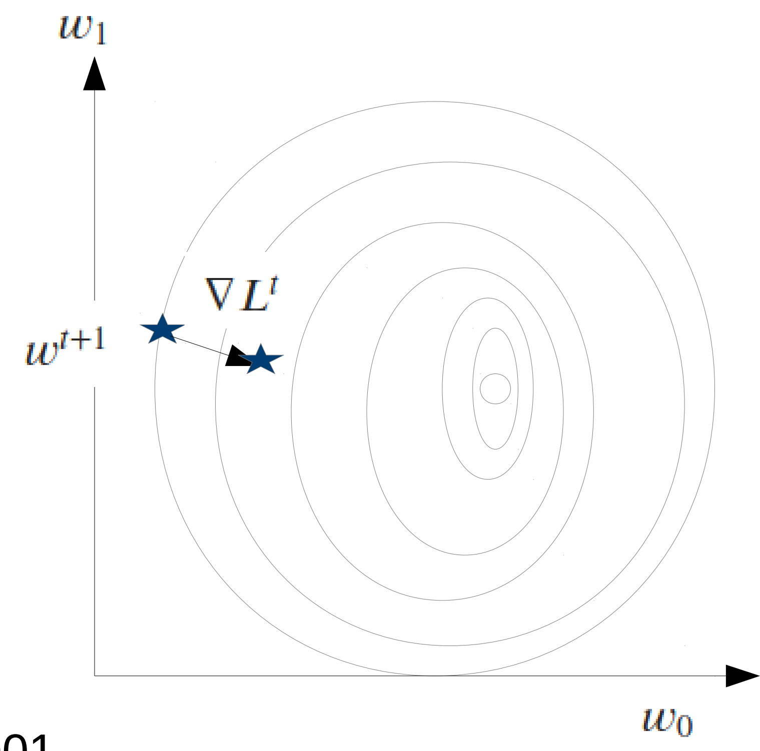
I. Start with random $w^{t=0}$

II. Compute gradient for all training samples

$$\nabla L^t = \sum_{i=0}^{|(X,y)|} \frac{dL(y_i, w^t x_i)}{dw^t}$$

III. Update parameters

$$w^{t+1} = w^t + \lambda \nabla L^t$$



Step size or Learning rate
Usually quite small scalar like 0.001

Gradient Descent Optimization

Goal: find w to minimize $\underset{w}{\arg\min} \sum_{i=0}^{N} L(y_i, w^T x_i)$

Gradient Descent Algorithm:

I. Start with random $w^{t=0}$

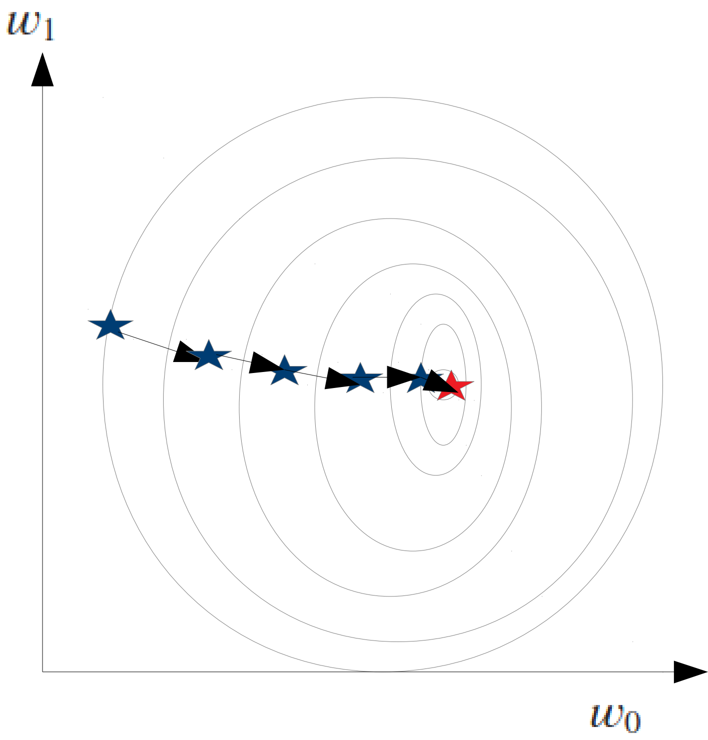
II. Compute gradient for all training samples

$$\nabla L^t = \sum_{i=0}^{|(X,y)|} \frac{dL(y_i, w^t x_i)}{dw^t}$$

III. Update parameters

$$w^{t+1} = w^t + \lambda \nabla L^t$$

IV. Repeat II-III till convergence



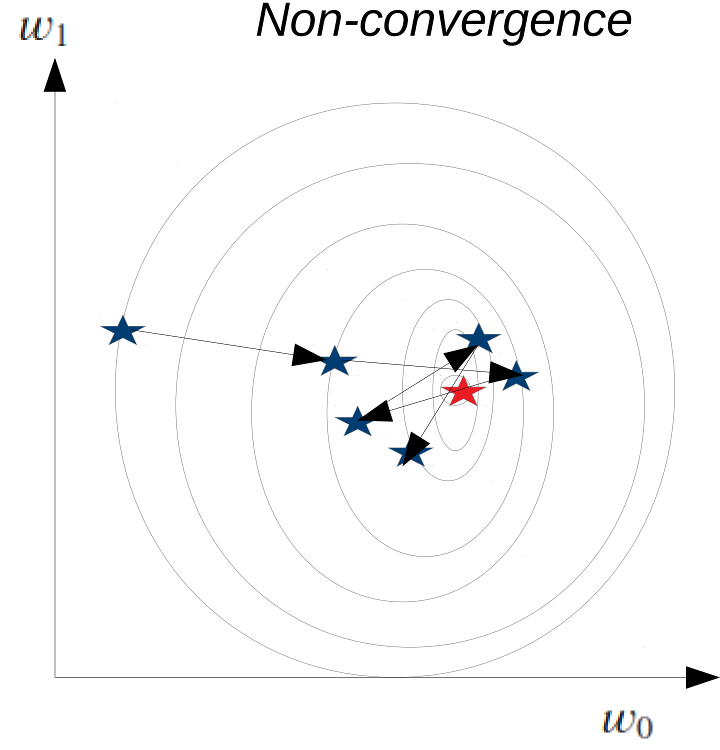
Gradient Descent Optimization

Goal: find w to minimize $\arg\min_{w} \sum_{i=0}^{N} L(y_i, w^T x_i)$

Convergence

I. Theory: need to decrease λ to guarantee convergence to minimum

Simplification: Non-convergence



Gradient Descent Optimization

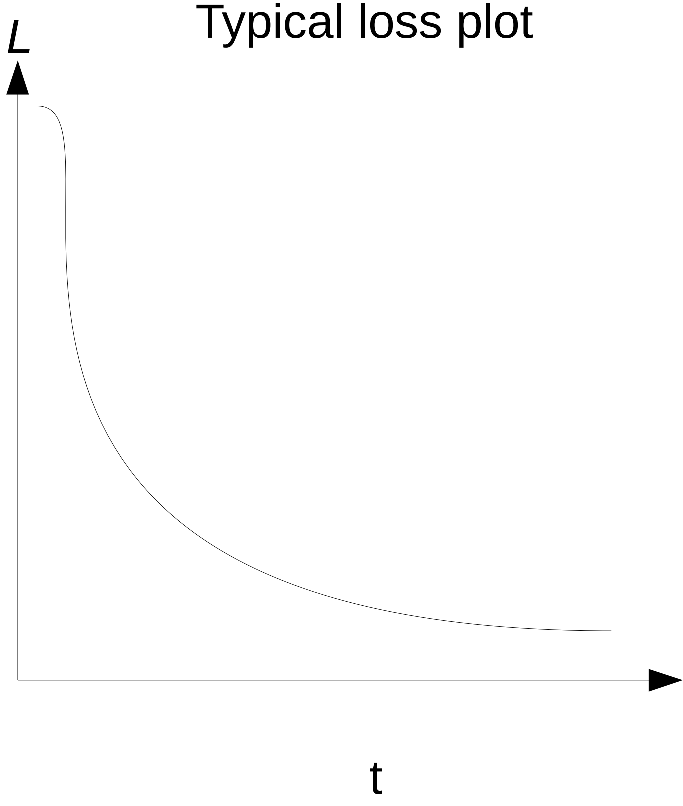
Goal: find w to minimize $\arg\min_{w} \sum_{i=0}^{N} L(y_i, w^T x_i)$

Convergence

I. Theory: need to decrease λ to guarantee convergence to minimum

II. How to know when to stop?
   I. Pre set number of iterations
   II. Loss limit
   III. Loss not changing

Typical loss plot



Multi Class Problems

What if we have more than two classes? → simple extension of our model

$$f(x) = y' = argmax(Wx)$$

Replace parameter vector by Matrix
→ One vector per class
→ Matrix vector Multiplication
→ returns vector with class-wise response
→ argmax selects maximum class label

Multi Class Problems

What if we have more than two classes? → simple extension of our model

$$f(x) = y' = argmax(Wx)$$

Optimization problem is almost the same

$$\arg\min_{w} \sum_{i=0}^{N} L(y_i, Wx_i)$$

Change *Loss* to *SOFTMAX* function to normalize sum aver all probabilities to one

$$L(y^i, y^{i\prime}) := \frac{e^{y^{i\prime}}}{\sum_j^k e^{y^{j\prime}}}$$

Use “one-hot” coding of y

Multi Class Problems

What if we have more than two classes? → simple extension of our model

$$f(x) = y' = argmax(Wx)$$

Optimization problem is almost the same

$$\arg\min_{w} \sum_{i=0}^{N} L(y_i, Wx_i)$$

Change *Loss* to *SOFTMAX* function to normalize sum aver all probabilities to one

$$L(y^i, y^{i\prime}) := \frac{e^{y^{i\prime}}}{\sum_j^k e^{y^{j\prime}}}$$

Use "one-hot" coding of y

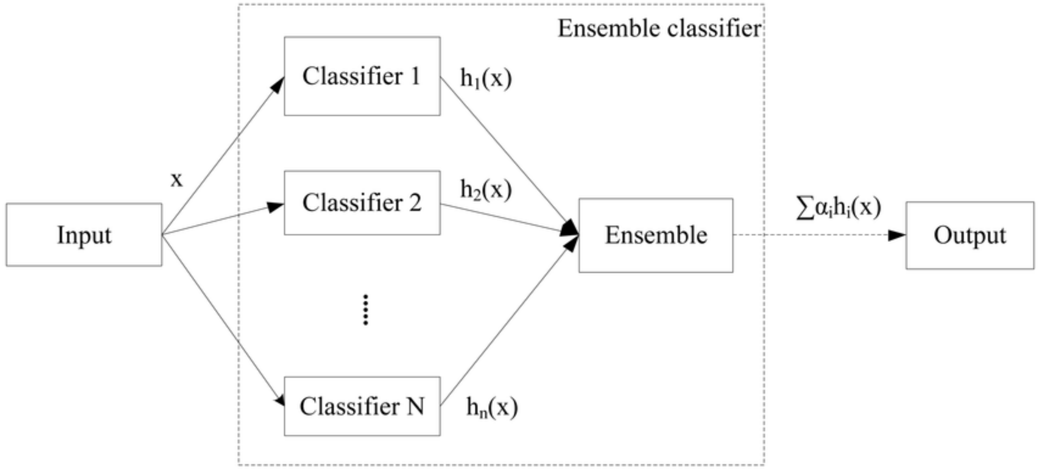
Y is now a vector with k (number of classes) entries and y^i is the kth class label

Discussion

Random Forests

Ensemble Learning

- Very popular method based on ensemble learning
  → many weak models decide together (by voting)
- Simple but powerful method
- Easy to implement and to parallelize
- Does not tend to overfit
- Build in Feature-Selection
  (next Lecture)



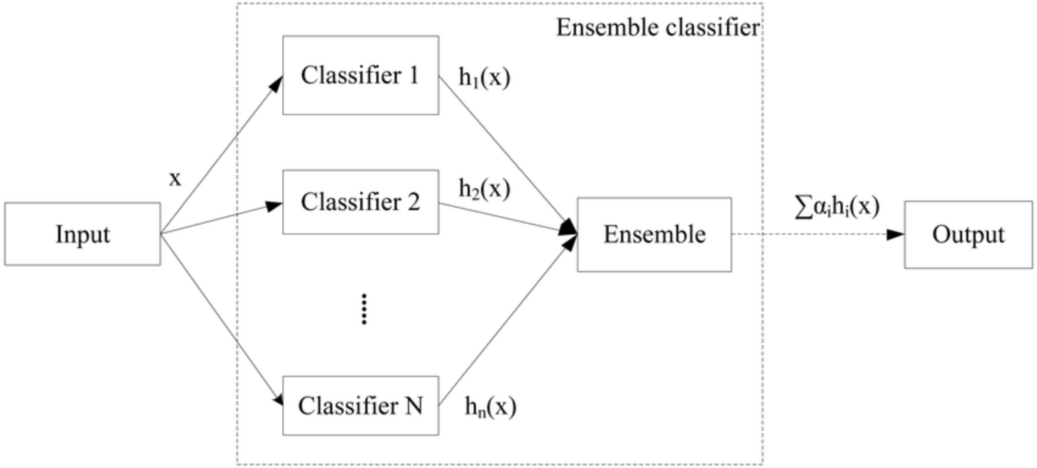
Random Forests

Ensemble Learning

- **Very popular method based on ensemble learning**
  → many weak models decide together (by voting)

Statistics: Bagging and Boosting

- **Simple but powerful method**
- **Approximating non-linear decision function by combination of piecewise linear functions**
- **Easy to implement and to parallelize**
- **Build in Feature-Selection (next Lecture)**

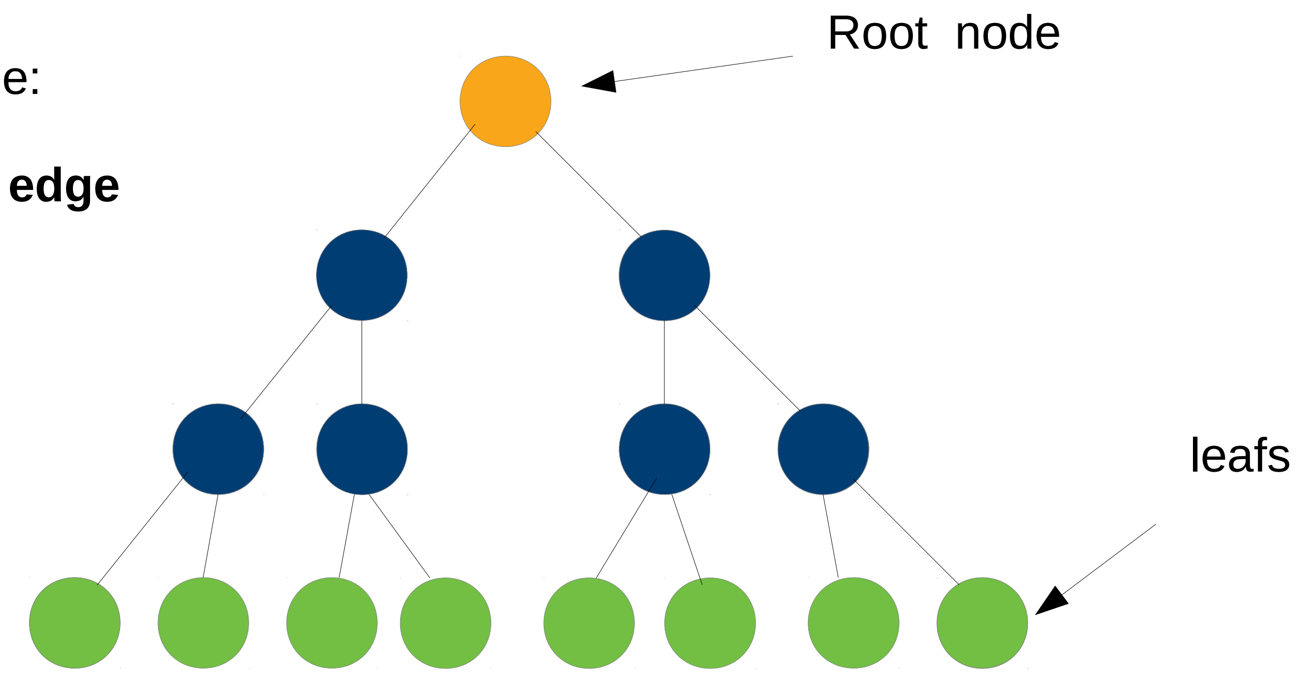


Random Forests

Decision Trees: the base classifier for Random Forests

Typically Binary Tree:

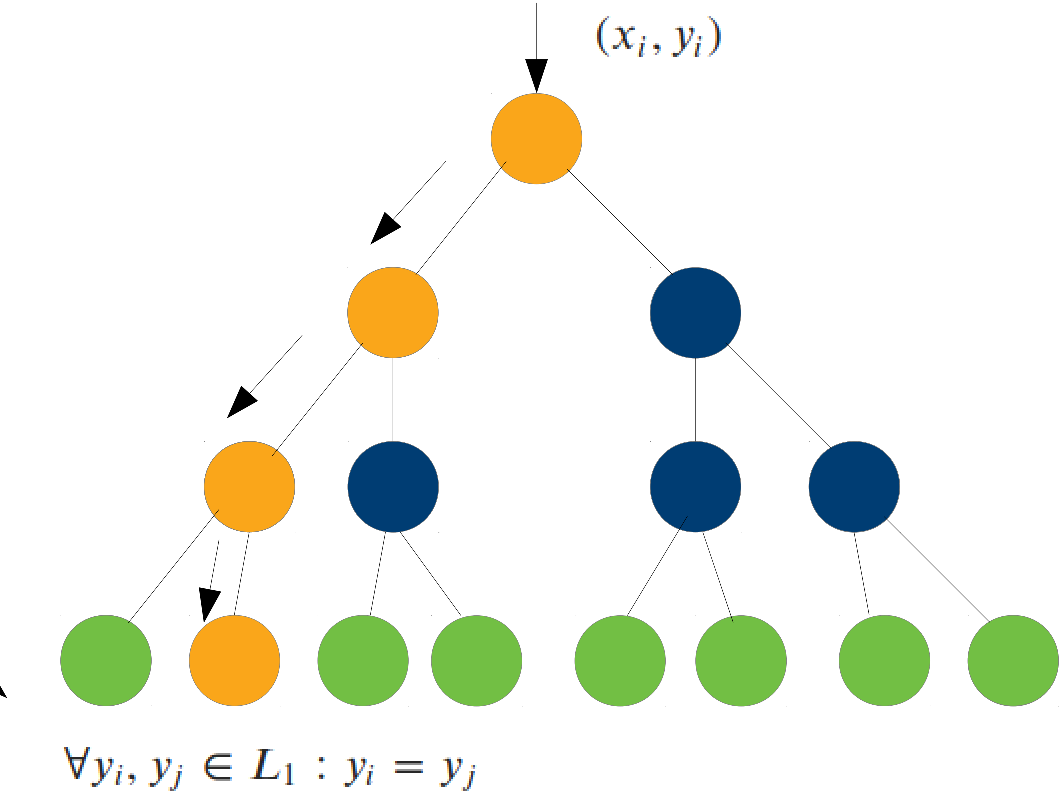
Vertex (Node) and edge



Random Forests

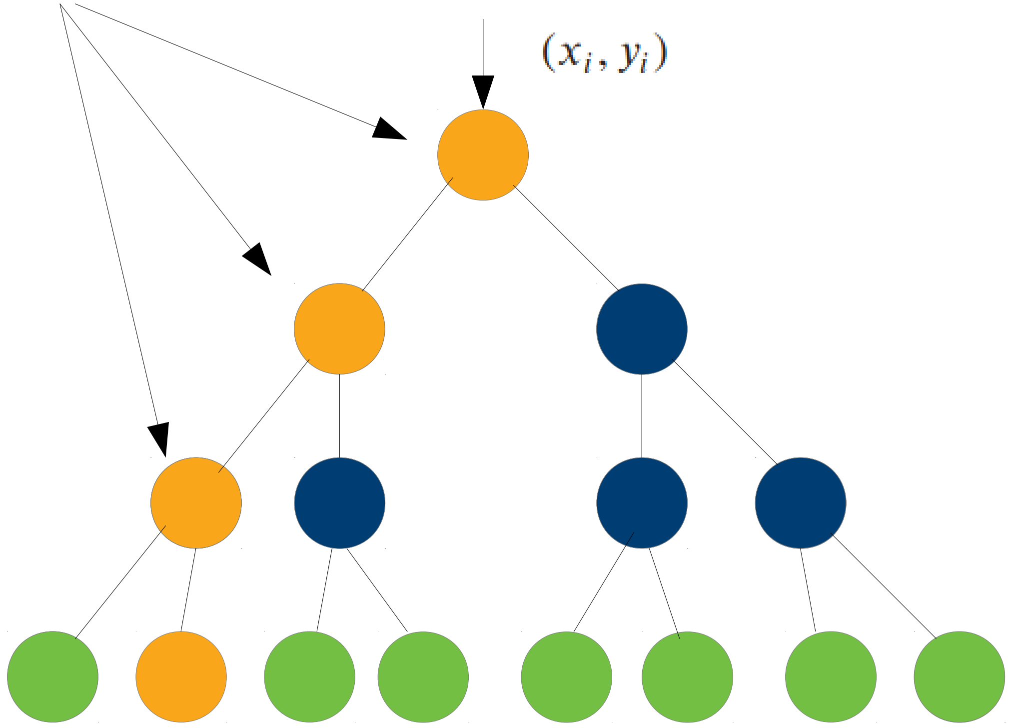
Goal: divide training data samples X at each node such that leafs have (mostly) Pure class labels

Classification: label y' is the label of the leaf node



Random Forests

All we need is a splitting function that will produce (almost) pure class labels



$\forall y_i, y_j \in L_1 : y_i = y_j$

Random Forests

KI Labor Südbaden

Top Down Training:

Assume ***k*** classes and data of dimension ***d***

- Fill tree with ALL training samples from the root down
- In each node: compute probability for all class labels in node *n*

$$p_i := p(y = i) = \frac{|(x,i)| \in X_n}{|X_n|}$$

- Compute **node purity** based on class probabilities
- Split node along one dimension such that purity of children is increasing $\leftarrow$ **optimization**

Random Forests

All we need is a splitting function that will produce (almost) pure class labels.

Entropy (a way to measure impurity):

$$Entropy = -\sum_j p_j \log_2 p_j$$

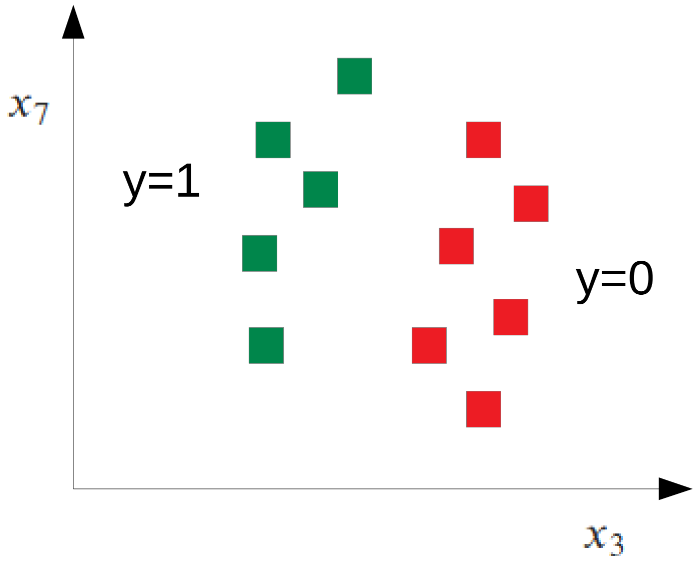
Gini index:

$$Gini = 1 - \sum_j p_j^2$$

Random Forests

Split optimization is a simple line search (Example):

I. Select a random subset of variables (feature dimensions) from the data X
e.g. x_7 x_3



Random Forests

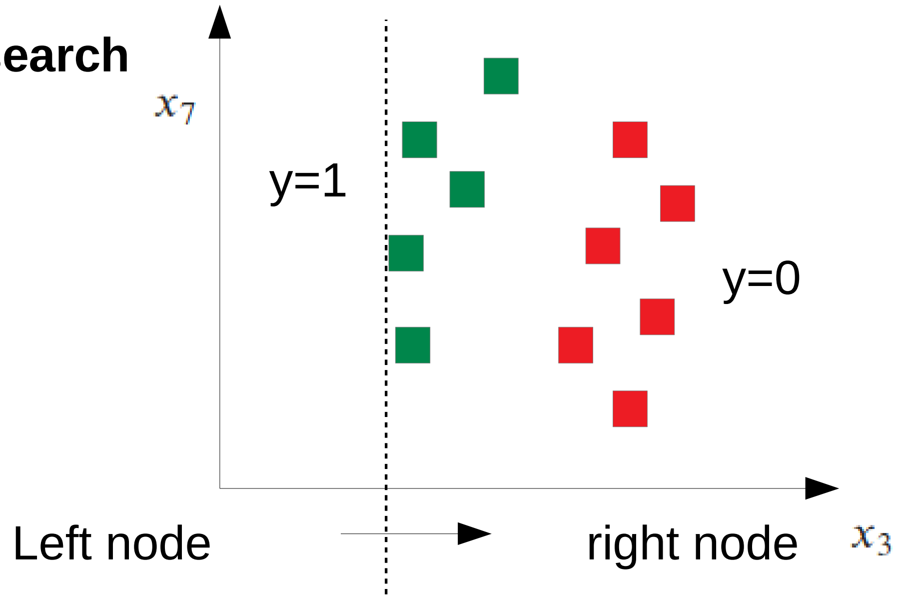
Split optimization is a simple line search (Example):

I. Select a random subset of variables (feature dimensions) from the data X
e.g. x_7 x_3

II. For each variable: find best split via line search
e.g. for x_3

Left: Undefined (div by zero)

Right: $p_1 = \frac{4}{11} = 0.36, p_0 = \frac{6}{11} = 0.54$



Random Forests

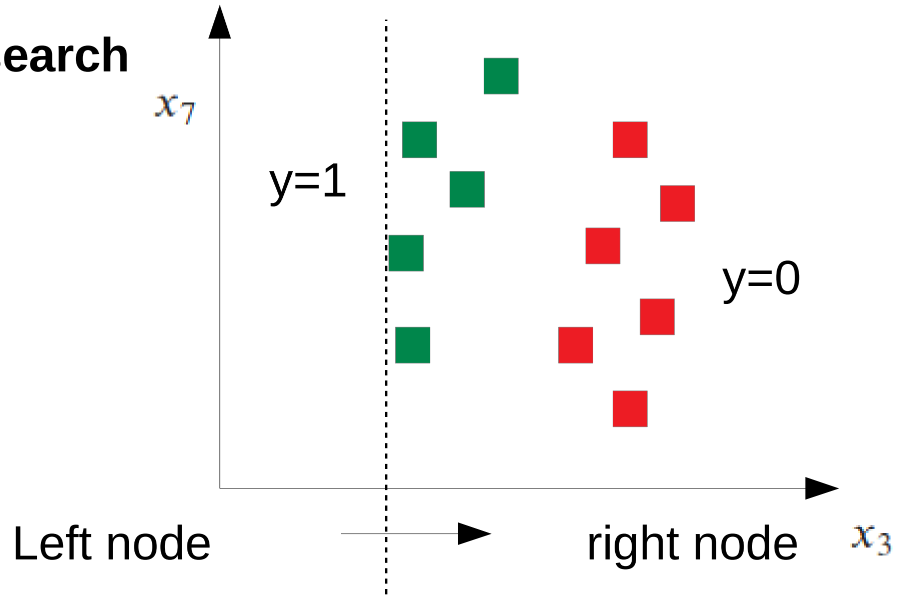
Split optimization is a simple line search (Example):

I. Select a random subset of variables (feature dimensions) from the data X
e.g. x_7 x_3

II. For each variable: find best split via line search
e.g. for x_3

Left: $gini = 1-(0^2+0^2) = 1$

Right: $gini = 1-(0.36^2+0.54^2) = 0.57$



Random Forests

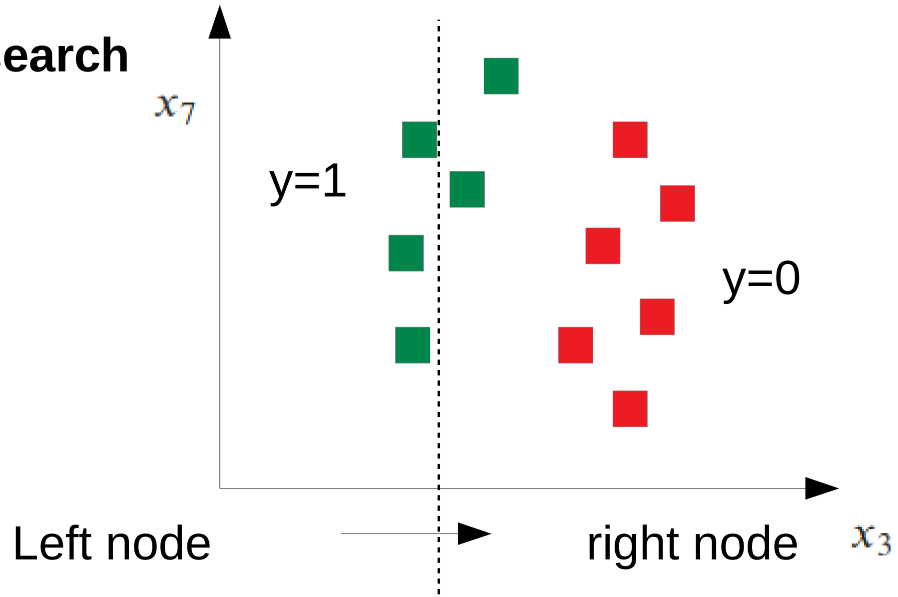
Split optimization is a simple line search (Example):

I. Select a random subset of variables (feature dimensions) from the data X
e.g. x_7 x_3

II. For each variable: find best split via line search
e.g. for x_3

Left: $p_1 = \frac{3}{3} = 1, p_0 = \frac{0}{3} = 0$

Right: $p_1 = \frac{2}{8} = 0.25, p_0 = \frac{6}{8} = 0.75$



Random Forests

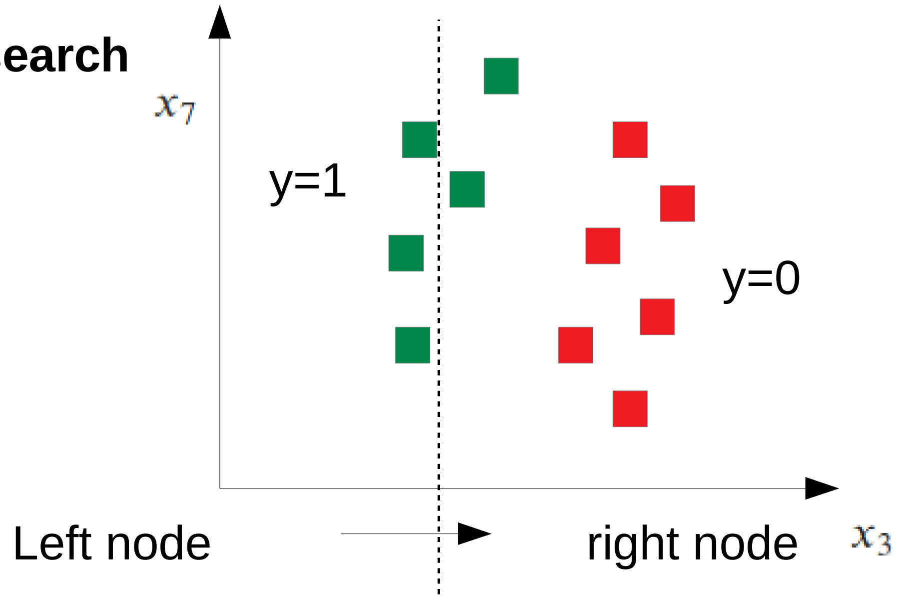
Split optimization is a simple line search (Example):

I. Select a random subset of variables (feature dimensions) from the data X
e.g. x_7 x_3

II. For each variable: find best split via line search
e.g. for x_3

Left: $gini = 1 - (1^2 + 0^2) = 0$

Right: $gini = 1 - (0.25^2 + 0.75^2) = 0.375$



Random Forests

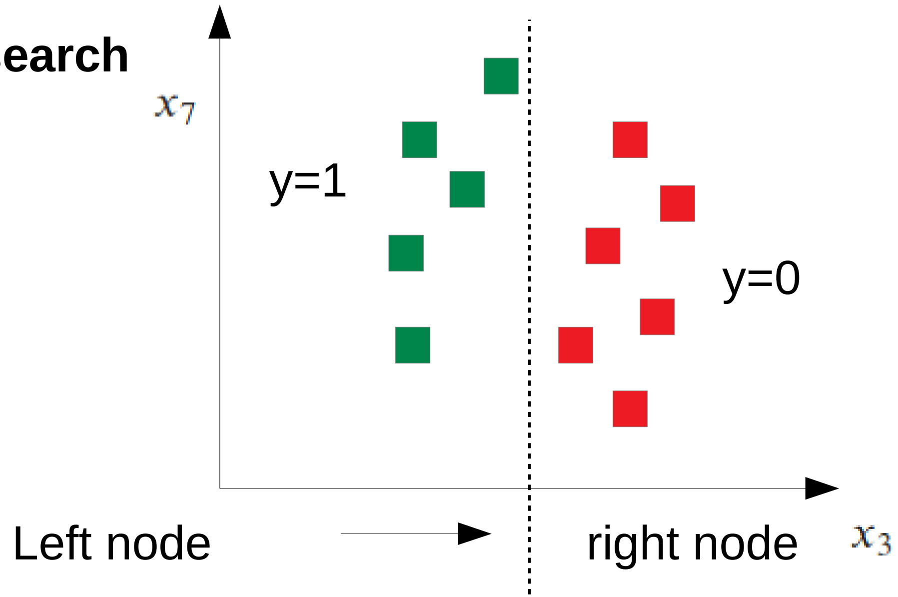
Split optimization is a simple line search (Example):

I. Select a random subset of variables (feature dimensions) from the data X
e.g. x_7 x_3

II. For each variable: find best split via line search
e.g. for x_3

Left: $p_1 = \frac{5}{5} = 1, p_0 = \frac{0}{5} = 0$

Right: $p_1 = \frac{0}{6} = 0, p_0 = \frac{6}{6} = 1$



Random Forests

Split optimization is a simple line search (Example):

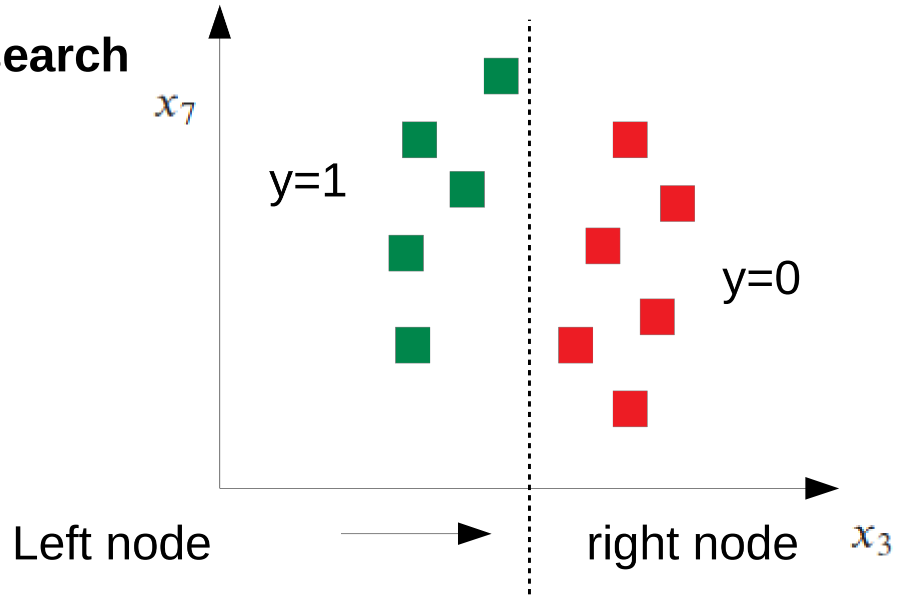
I. Select a random subset of variables (feature dimensions) from the data X
e.g. x_7 x_3

II. For each variable: find best split via line search
e.g. for x_3

Left: $gini = 1 - (1^2 + 0^2) = 0$

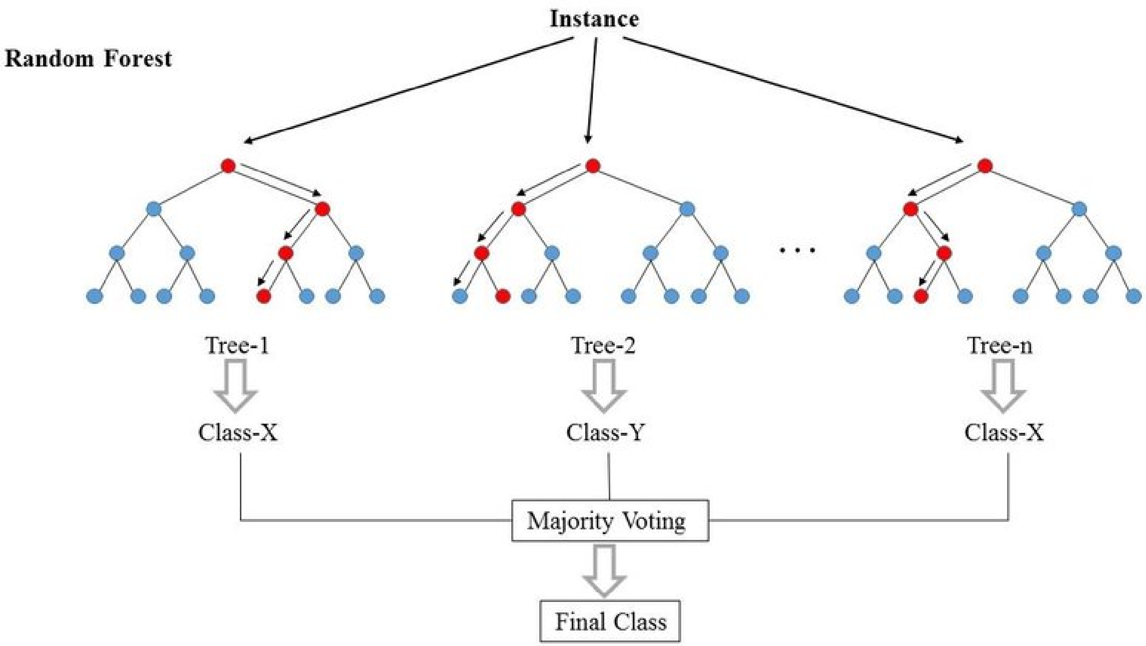
Right: $gini = 1 - (0^2 + 1^2) = 0$

Perfect split!



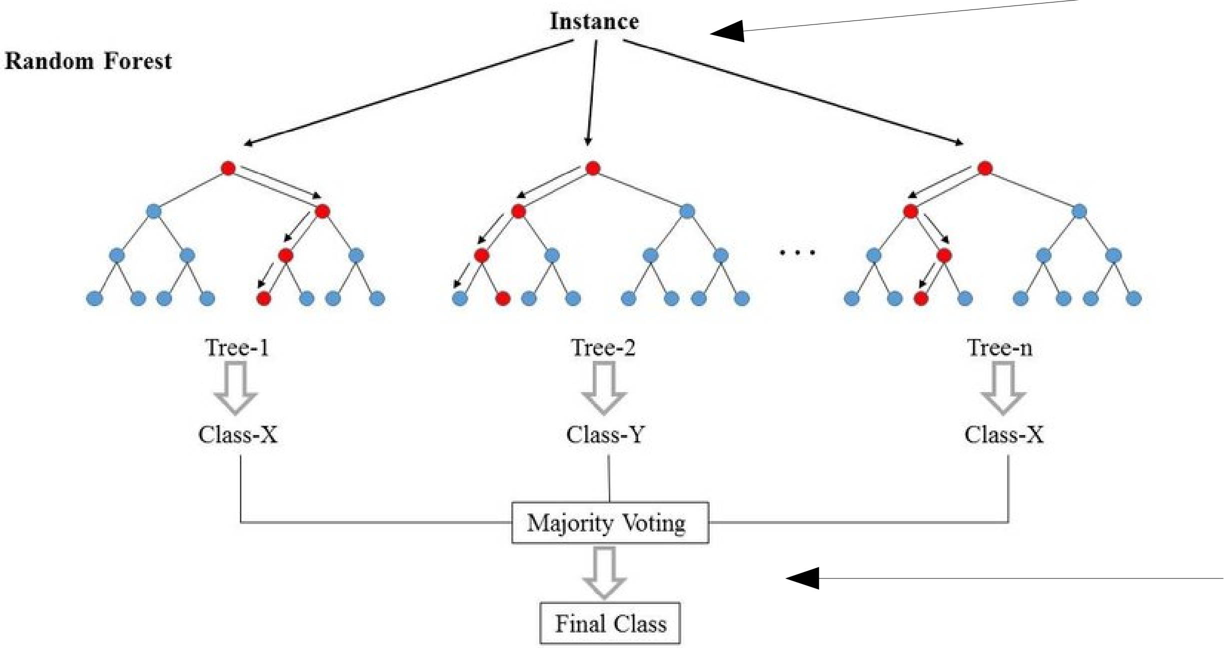
Random Forests

Ensemble Learning: A Forest of Trees



Random Forests

Ensemble Learning: A Forest of Trees

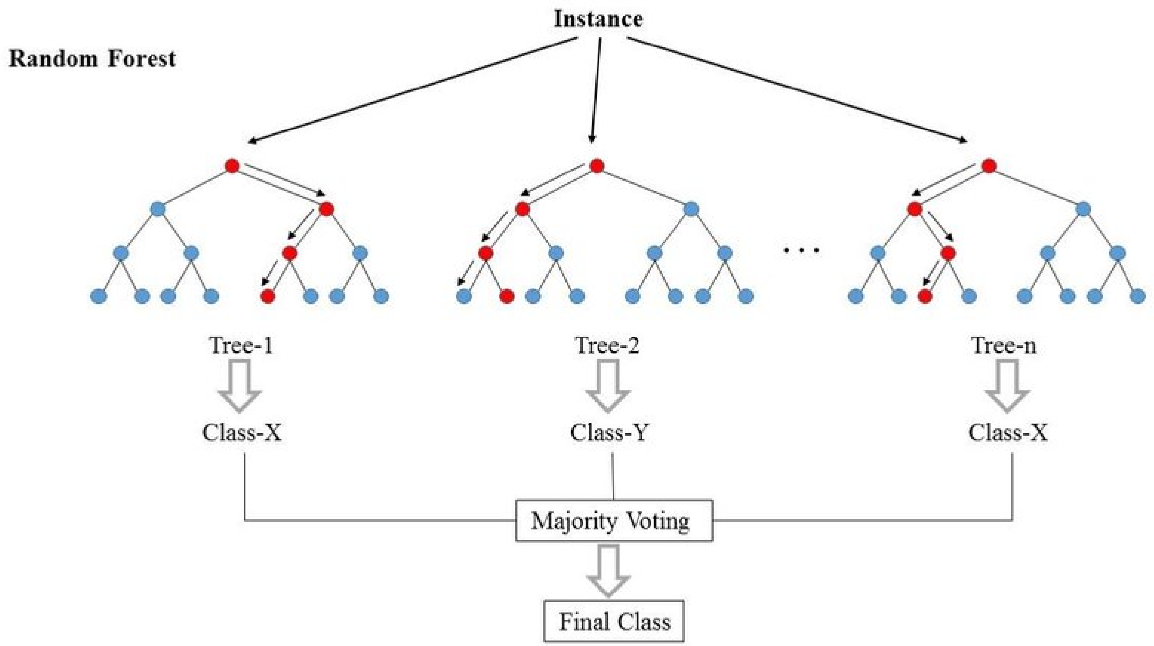


Split Training data into random subsets
→ Bootstrap

Combine Models
→ Bagging

Random Forests

Ensemble Learning: A Forest of Trees

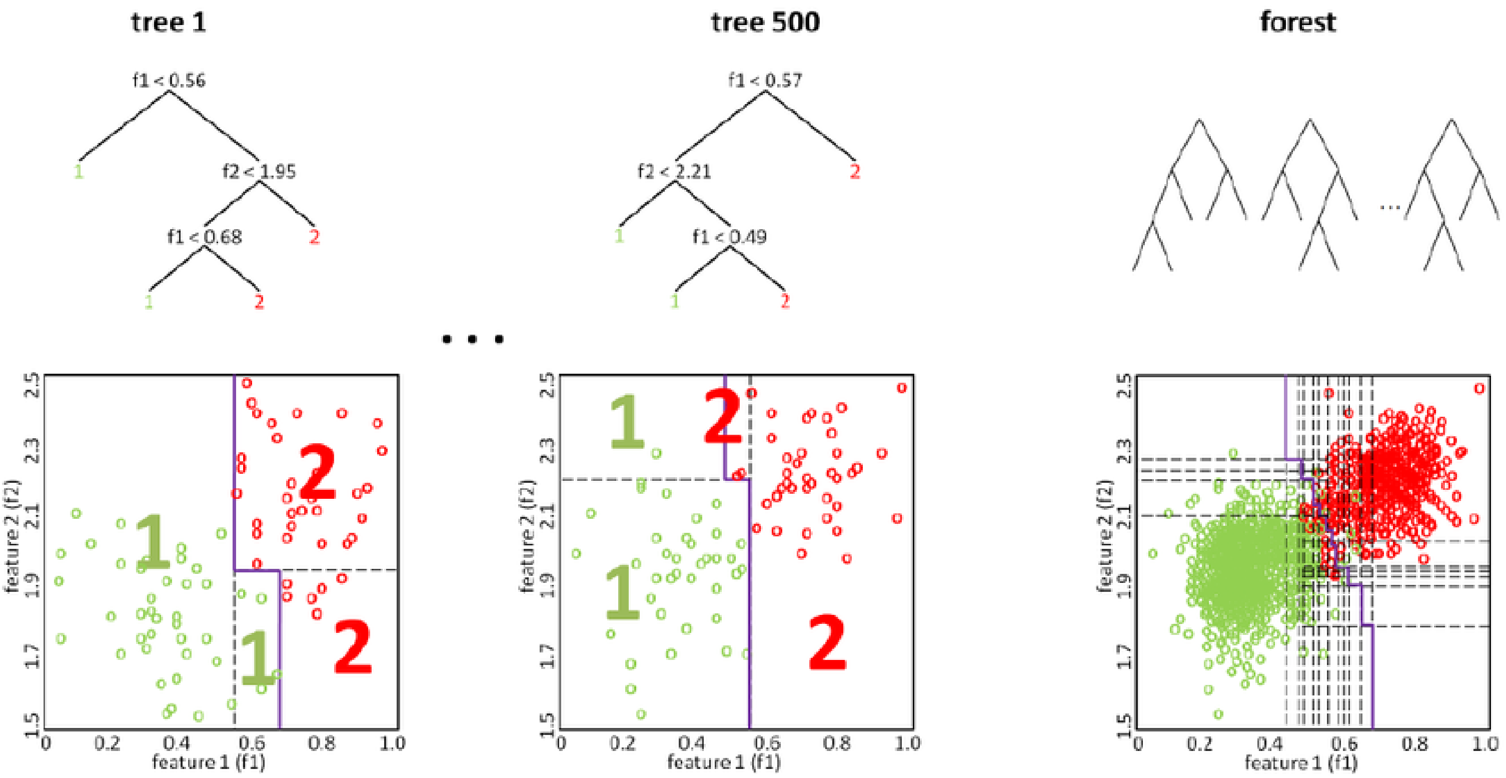


Parameters:

- #trees
- Portion of data per tree
- #vars per split
- Stopping
  - max depth
  - min samples per node

Random Forests

Classification example

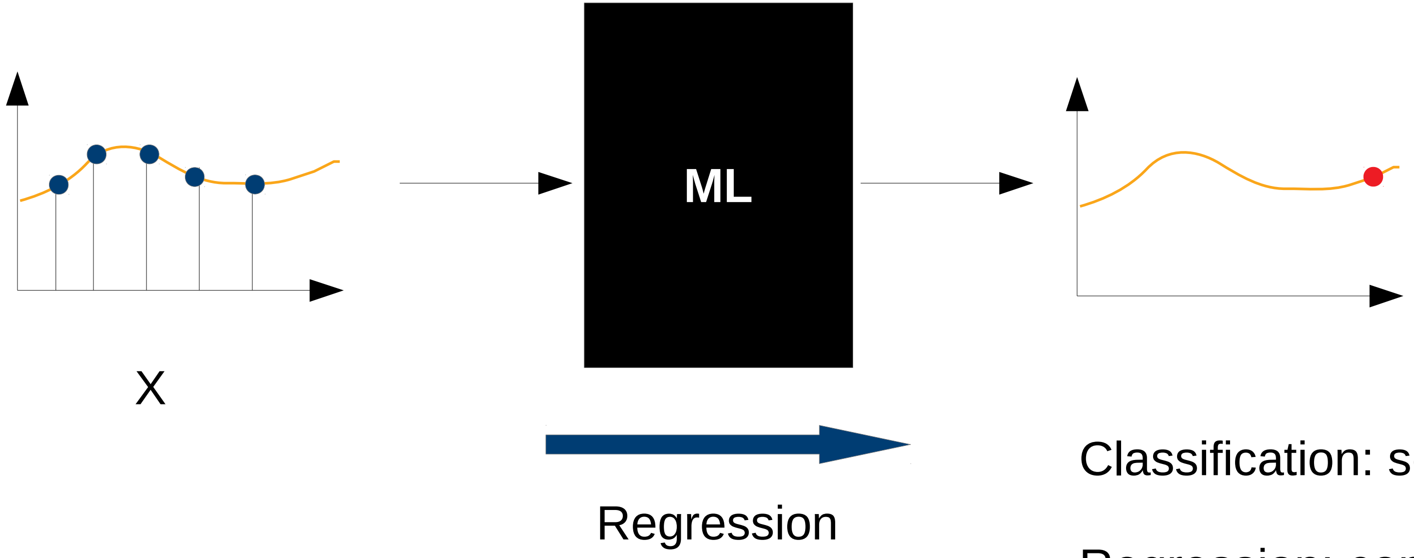


Random Forests

Discussion

Regression

Recall:



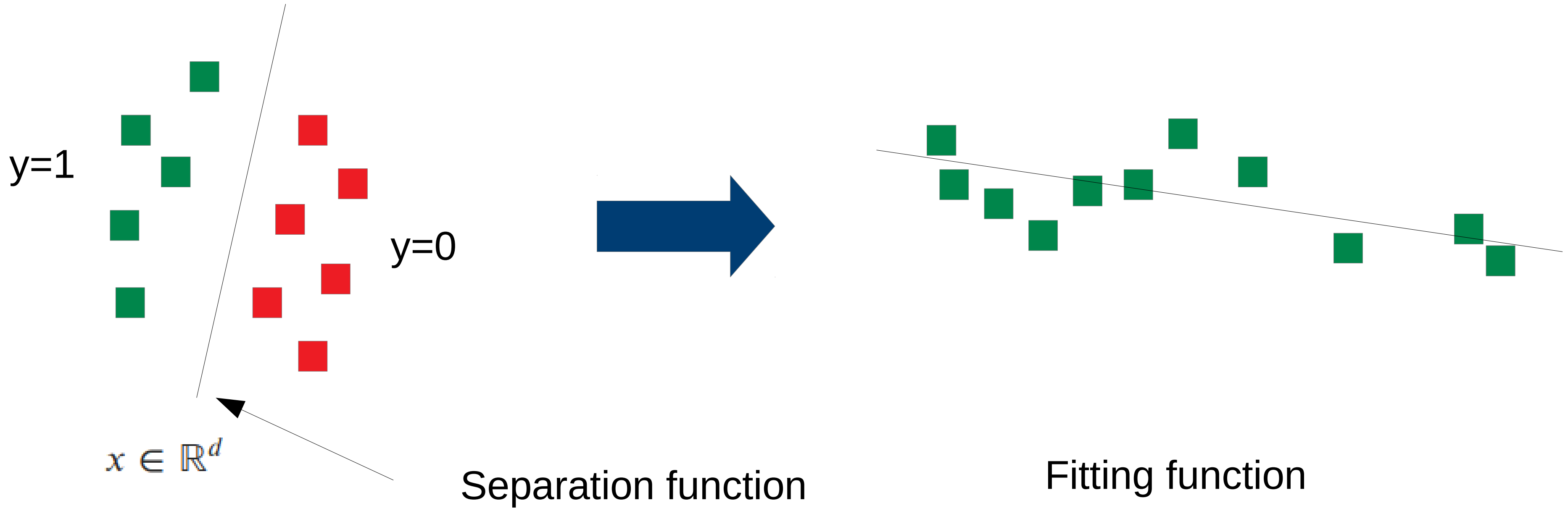
Y is **continuous**

Classification: separation function

Regression: continuous mapping
→ harder problem

Linear Regression

How do we have to change out linear classifier to predict continuous values?

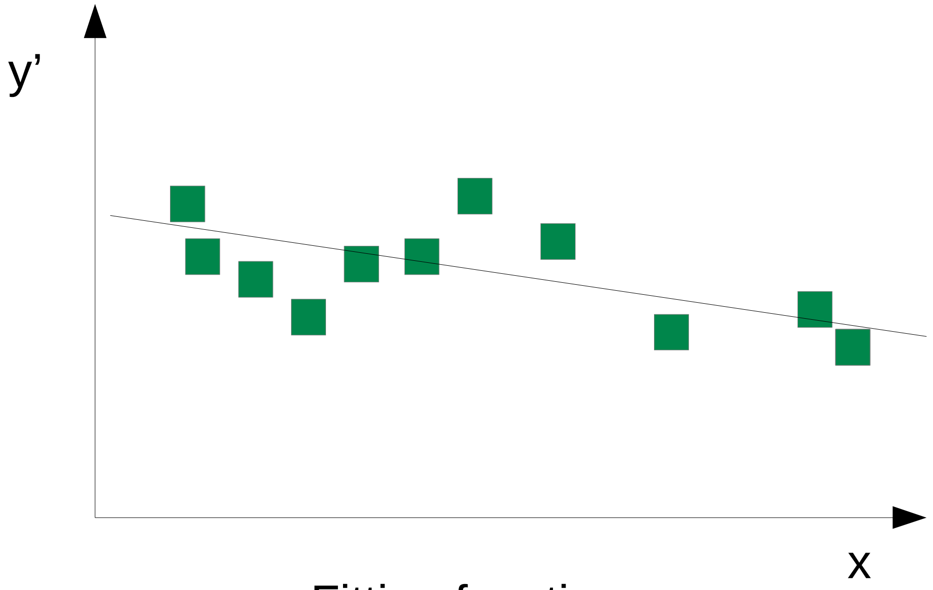


Linear Regression

How do we have to change out linear classifier to predict continuous values?

Still can use the same framework

$$f(x) = y' = w^T x = \sum_{j=0}^{d} w_j x_j$$



Fitting function

Linear Regression

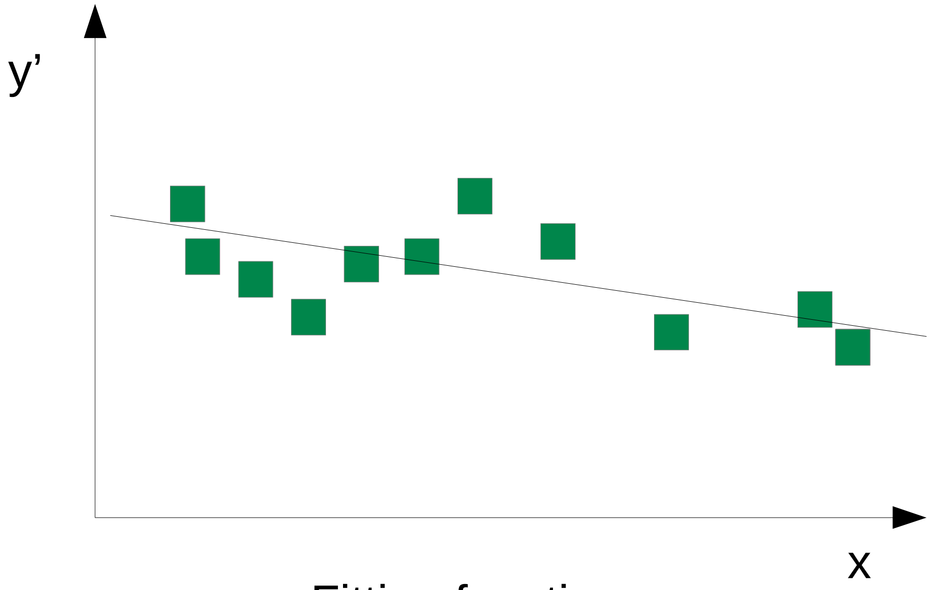
How do we have to change out linear classifier to predict continuous values?

Still can use the same framework

$$f(x) = y' = w^T x = \sum_{j=0}^{d} w_j x_j$$

Simply need new loss function in the optimization

$$\arg\min_{w} \sum_{i=0}^{N} L(y_i, w^T x_i)$$



Fitting function

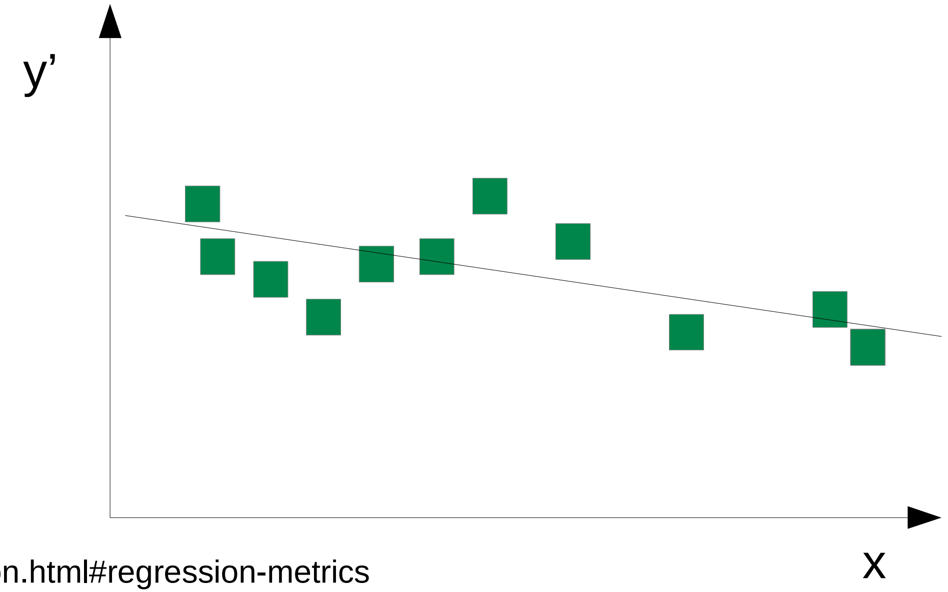
Linear Regression

Loss functions for regression:

$$\arg\min_{w} \sum_{i=0}^{N} L(y_i, w^T x_i)$$

As simple as least squares error

$$L_{LSE}(y, y') := \|y - y'\|^2$$



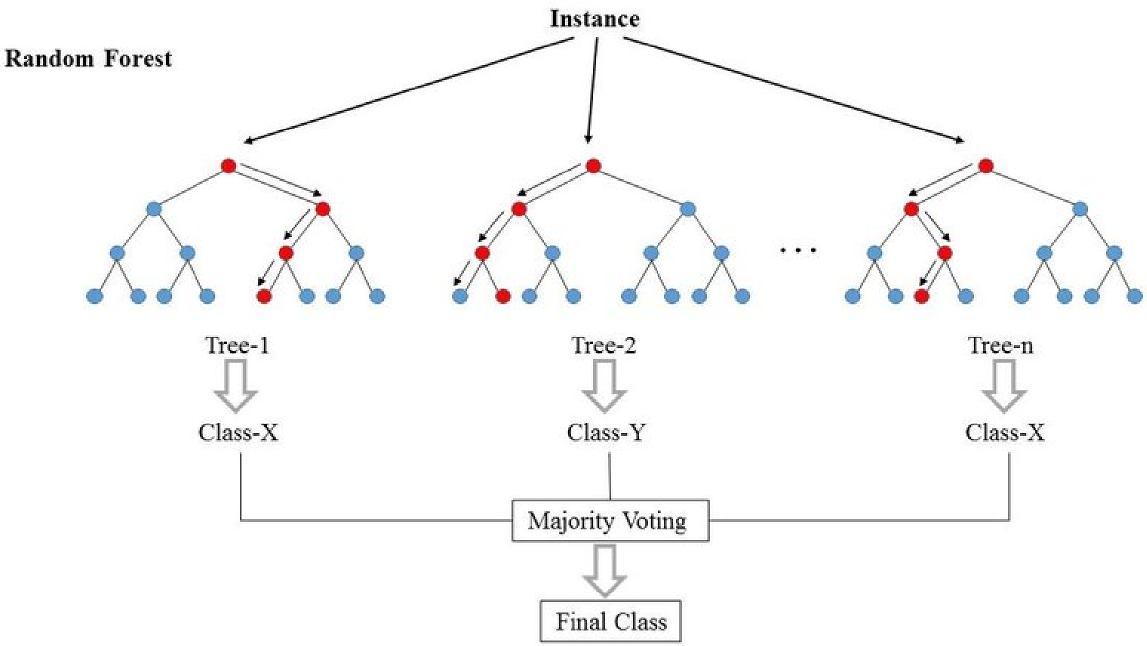
Many other error measures possible

- L1 (Histogram intersection)
- …

→ see https://scikit-learn.org/stable/modules/model_evaluation.html#regression-metrics

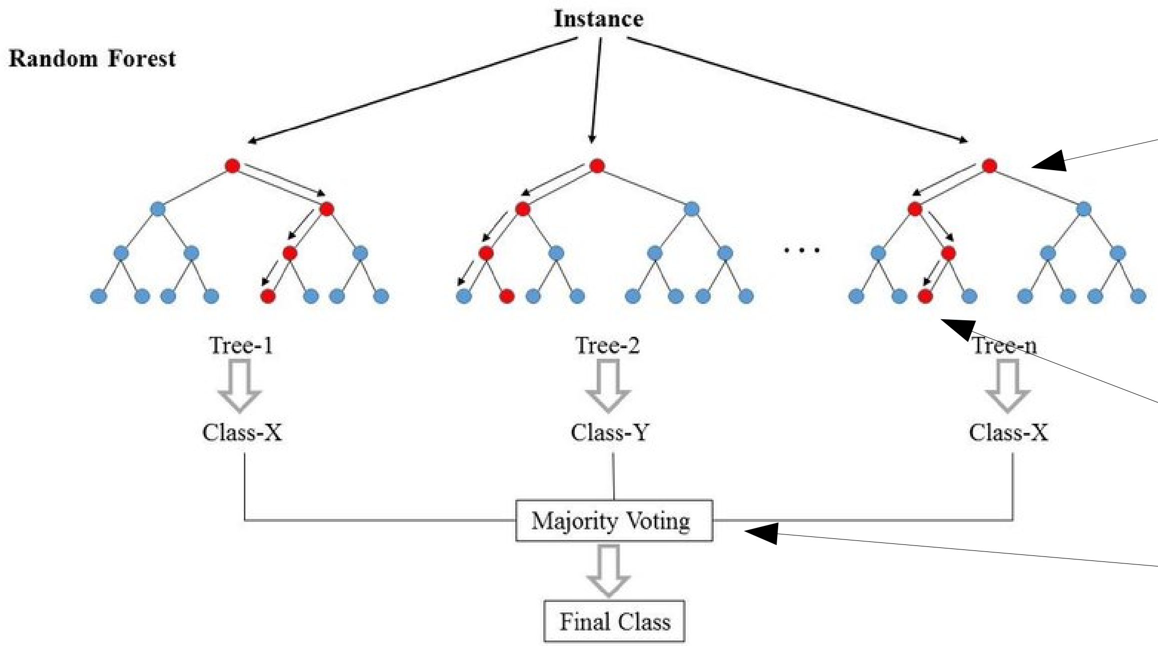
Regression with Random Forests

Recall:



Regression with Random Forests

Recall:



New splitting function

Replace majority vote by average

Regression with Random Forests

Splitting functions for regression:

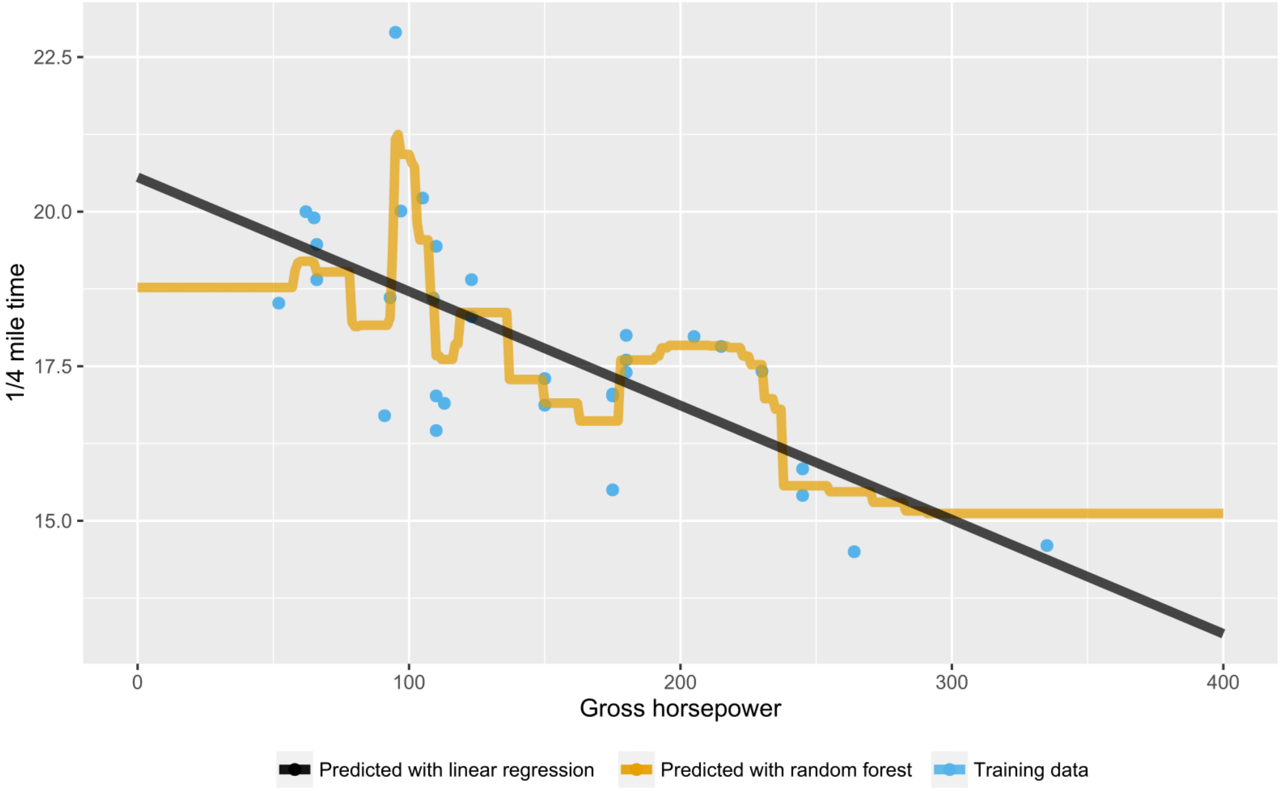
Goal: reduce “data spread” in node

→ use simple statistical measure like “mean square error”

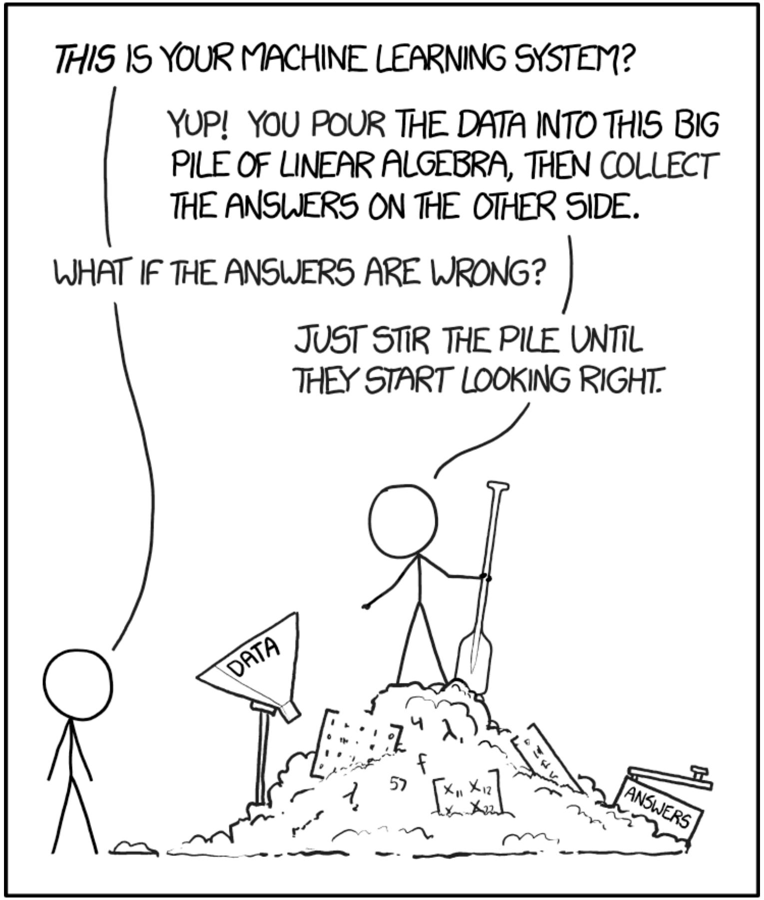
$$MSE : \sum_{(x_i, y_i) \in X_n} \| \mu_y - y_i \|^2$$

Regression with Random Forests

Example



Discussion



https://xkcd.com/1838/