# Algorithms for Ranking

- Support Vector Machines (Vapnik, 1995)
  - Very good classifier
  - Can be adapted to ranking and multiclass problems
- Neural Nets
  - RankNet (Burges et al., 2006)
- Tree Ensembles
  - Random Forests (Breiman and Schapire, 2001)
  - Boosted Decision Trees
    - Multiple Additive Regression Trees (Friedman, 1999)
    - LambdaMART (Burges, 2010)
    - Used by AltaVista, Yahoo!, Bing, Yandex, ...

All top teams of the *Yahoo! Learning to Rank Challenge (2010)* used combinations of Tree Ensembles!

# Yahoo! Learning to Rank Challenge

- Yahoo! Webscope dataset (Chapelle and Chang, 2011): 36,251 queries, 883 k documents, 700 features, 5 ranking levels
  - set-1:
    - 473,134 feature vectors
    - 519 features
    - 19,944 queries
  - set-2:
    - 34,815 feature vectors
    - 596 features
    - 1,266 queries
- Winner used a combination of 12 models:
  - 8 Tree Ensembles (LambdaMART)
  - 2 Tree Ensembles (Additive Regression Trees)
  - 2 Neural Nets

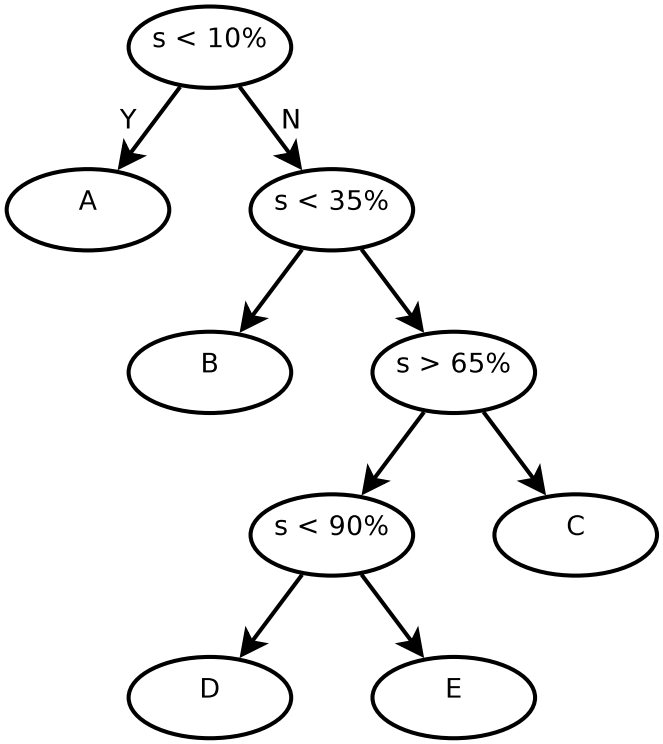
# Decision Trees

Characteristics of a tree:

- Graph based model
- Consists of a root, nodes, and leaves

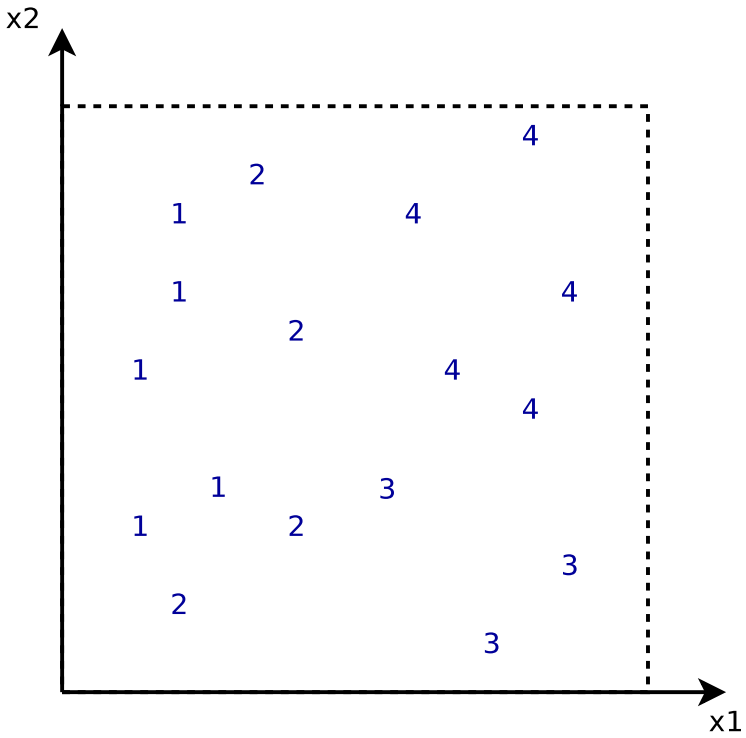
Advantages:

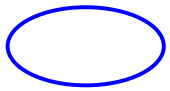
- Simple to understand and interpret
- *White box* model
- Can be combined with other techniques




Decision trees are basic learners for machine learning, e.g. *classification* or *regression trees*.

# Learning a Regression Tree (I)

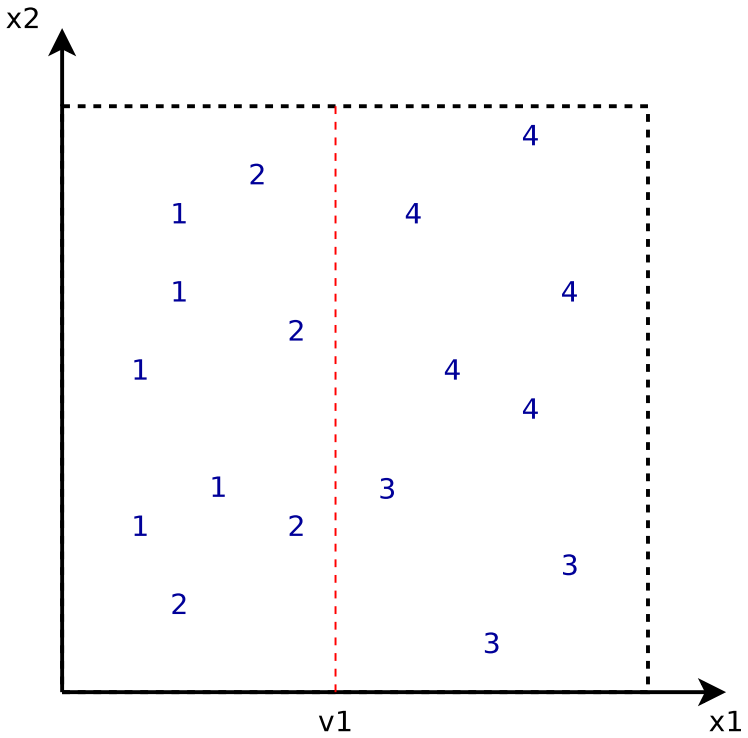


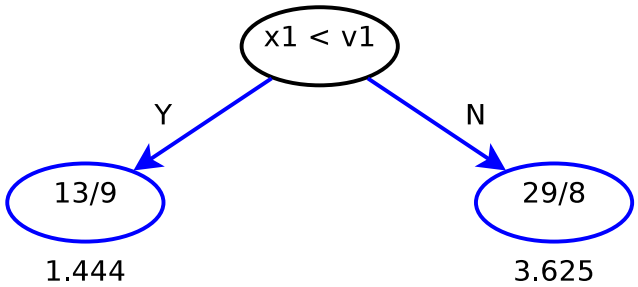
What is the prediction of this empty tree?
Mean value of all training instances

Consider a 2-dimensional space consisting of data points of the indicated values. We start with an empty root node (blue).

# Learning a Regression Tree (II)




How to determine split threshold v1?
How to find the prediction for each leave?

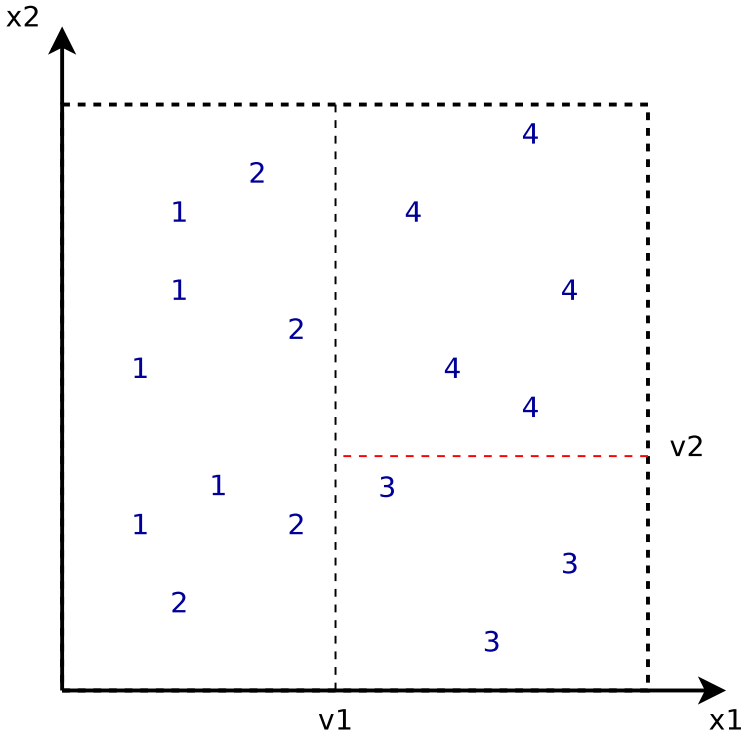


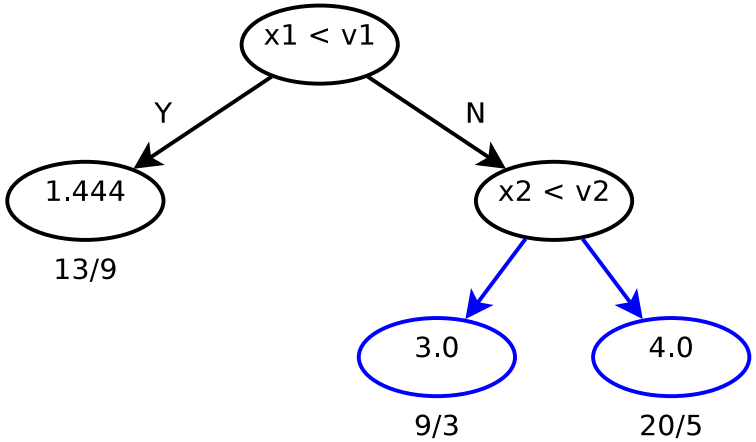

Use mean value of training instances on left leave as prediction:
(5*1+4*2)/9=1.444
For right leave, (4*5+3*3)/8=3.625.

Try all possible v1 thresholds.

The algorithm searches for split variables and split points, $x_1$ and $v_1$, that predict values minimizing the predicted error, e.g. $\sum(y_i - f(x_i))^2$.

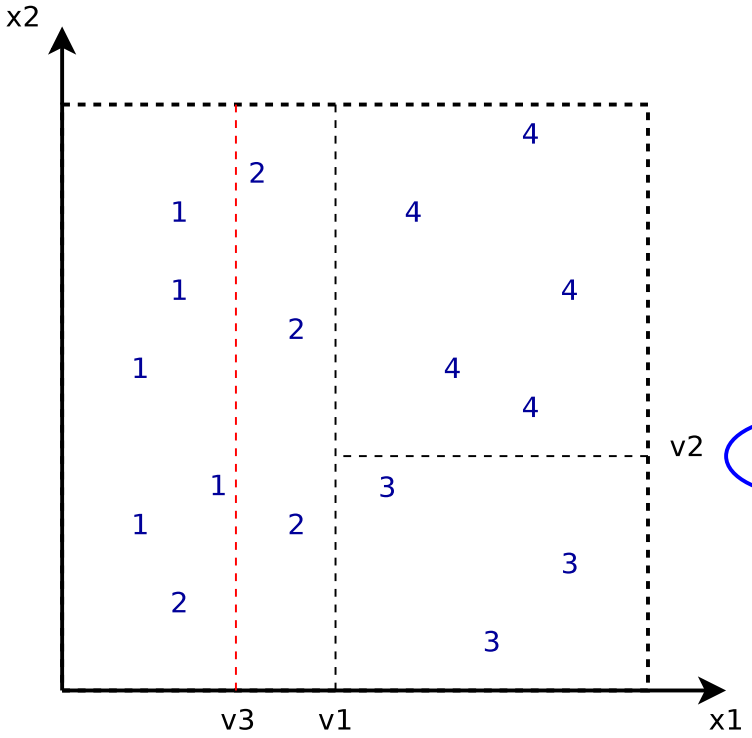
# Learning a Regression Tree (III)

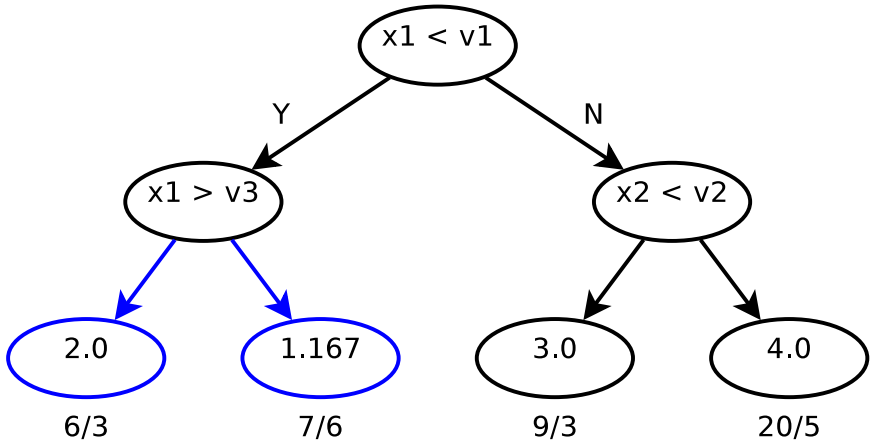



Here we examine the right side first: find a split variable and a split value that minimize the predicted error, i.e. $x_2$ and $v_2$.

# Learning a Regression Tree (IV)

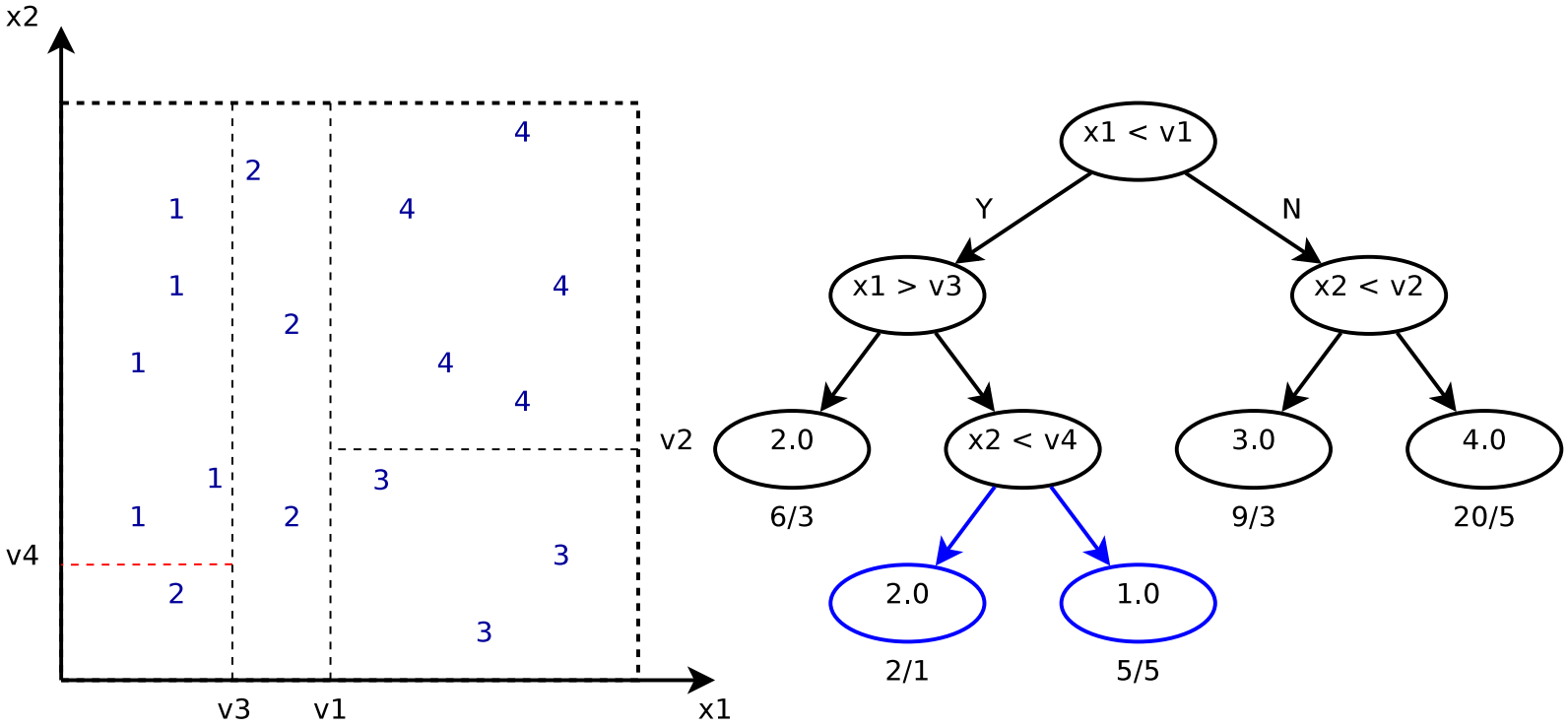



Choose the mean value of training instances within each leave as a prediction

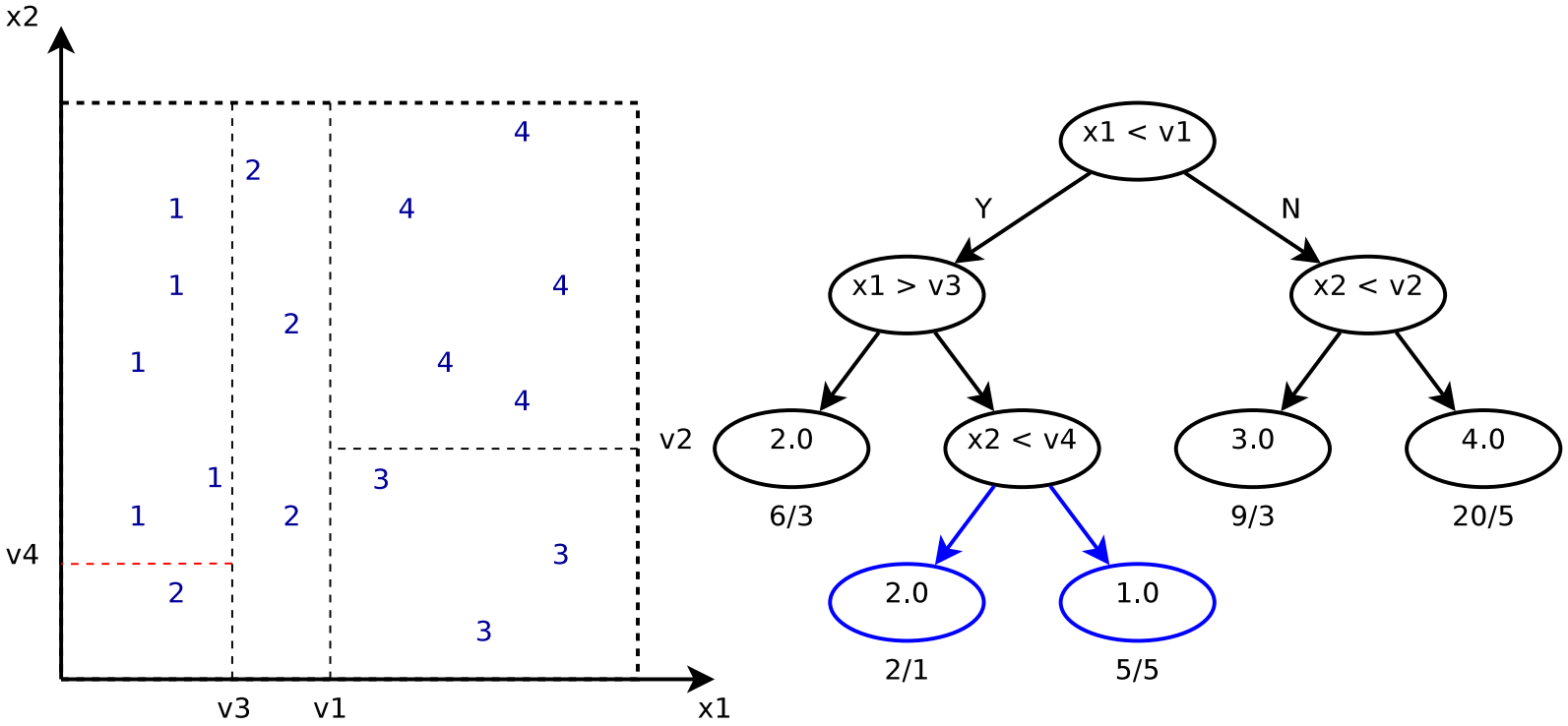
Now to the left side: Again, find a split variable and a split value that minimize the predicted error, i.e. $x_1$ and $v_3$.

# Learning a Regression Tree (V)




Once again, find a split variable and a split value that minimize the predicted error, here $x_2$ and $v_4$.

# Learning a Regression Tree (V)




Once again, find a split variable and a split value that minimize the predicted error, here $x_2$ and $v_4$. The tree perfectly fits the data! Problem?

# Formal Definition of a Decision Tree

A decision tree partitions the parameter space into disjoint regions $R_k$, $k \in \{1, ..., K\}$, $K =$ number of leaves. Formally, the regression model (1) predicts a value using a constant $\gamma_k$ for each region $R_k$:

Prediction:

$$T(\mathbf{x}; \Theta) = \sum_{k=1}^{K} \gamma_k 1(\mathbf{x} \in R_k) \tag{1}$$

$\Theta = \{R_k, \gamma_k\}_1^K$ describes the model parameters, $1(\cdot)$ is the *characteristic function* (1 if argument is true, 0 otherwise), and $\hat{\gamma}_k = \text{mean}(y_i | \mathbf{x}_i \in R_k)$. Optimal parameters $\hat{\Theta}$ are found minimizing the empirical risk:

For all instances

Loss function

$$\hat{\Theta} = \arg\min_{\Theta} \sum_{k=1}^{K} \sum_{\mathbf{x}_i \in R_k} L(y_i, \gamma_k) \tag{2}$$

The combinatorial optimization problem (2) is usually split into two parts: (i) *finding* $R_k$ and (ii) *finding* $\gamma_k$ *given* $R_k$.

# Boosting

**Idea**

Combine multiple weak learners to build a strong learner.
A weak learner is a learner with an error rate slightly better than random guessing. A strong learner is a learner with high accuracy.

Approach:

- Apply a weak learner to iteratively modified data
- Generate a sequence of learners
- For classification tasks: use majority vote
- For regression tasks: build weighted values

Loss function is sum of squared error for all instances

# Function Estimation

Find a function $F^*(\mathbf{x})$ that maps $\mathbf{x}$ to $y$, s.t. the expected value of some loss function $L(y, F(\mathbf{x}))$ is minimized:

$$F^*(\mathbf{x}) = \arg\min_{F(\mathbf{x})} \mathbb{E}_{y,\mathbf{x}} \left[ L(y, F(\mathbf{x})) \right]$$

Boosting approximates $F^*(\mathbf{x})$ by an additive expansion

$$F(\mathbf{x}) = \sum_{m=1}^{M} \beta_m h(\mathbf{x}; \mathbf{a}_m)$$

What are parameters a? In regression trees, they represent how each tree is built and leave prediction is made.

where $h(\mathbf{x}; \mathbf{a})$ are simple functions of $\mathbf{x}$ with parameters $\mathbf{a} = \{a_1, a_2, ..., a_n\}$ defining the function $h$, and $\beta$ are expansion coefficients.

For boosting trees, F(x) is sum of tree classifiers.
M is the number of trees.
Expansion coefficient tyically includes a learning rate factor

# Finding Parameters

Expansion coefficients $\{\beta_m\}_0^M$ and the function parameters $\{\mathbf{a}_m\}_0^M$ are iteratively fit to the training data:

1. Set $F_0(\mathbf{x})$ to initial guess
2. For each $m = 1, 2..., M$

Minimize overall
loss for all N training instances.
M is # of trees

$$(\beta_m, \mathbf{a}_m) = \arg\min_{\beta,\mathbf{a}} \sum_{i=1}^{N} L(y_i, F_{m-1}(\mathbf{x}_i) + \beta h(\mathbf{x}_i, \mathbf{a})) \tag{3}$$

and

$$F_m(\mathbf{x}) = F_{m-1}(\mathbf{x}) + \beta_m h(\mathbf{x}; \mathbf{a}_m) \tag{4}$$

For each iteration, we only determine one tree classifor h().
How to minimize loss for one tree?
Try to find a function close to residual yi-F

# Gradient Boosting

*Gradient boosting* approximately solves (3) for differentiable loss functions:

1. Fit the function $h(\mathbf{x}; \mathbf{a})$ by least squares

$$\mathbf{a}_m = \arg\min_{\mathbf{a}} \sum_{i=1}^{N} [\tilde{y}_{im} - h(\mathbf{x}_i, \mathbf{a})]^2 \tag{5}$$

to the "pseudo"-residuals

$$\tilde{y}_{im} = -\left[\frac{\partial L(y_i, F(\mathbf{x}_i))}{\partial F(\mathbf{x}_i)}\right]_{F(\mathbf{x})=F_{m-1}(\mathbf{x})} \tag{6}$$

2. Given $h(\mathbf{x}; \mathbf{a}_m)$, the $\beta_m$ are

Residual is yi - F

$$\beta_m = \arg\min \sum_{i=1}^{N} L(y_i, F_{m-1}(\mathbf{x}_i) + \beta h(\mathbf{x}_i; \mathbf{a}_m)) \tag{7}$$

$\Rightarrow$ Gradient boosting simplifies the problem to least squares (5).

# Gradient Tree Boosting

*Gradient tree boosting* applies this approach on functions $h(\mathbf{x}; \mathbf{a})$ representing $K$-terminal node regression trees.

$$h(\mathbf{x}; \{R_{km}\}_1^K) = \sum_{k=1}^{K} \bar{y}_{km} 1(\mathbf{x} \in R_{km}) \tag{8}$$

With $\bar{y}_{km} = \text{mean}_{\mathbf{x}_i \in R_{km}}(\tilde{y}_{im})$ the tree (8) predicts a constant value $\bar{y}_{km}$ in region $R_{km}$. Equation (7) becomes a prediction of a $\gamma_{km}$ for each $R_{km}$:

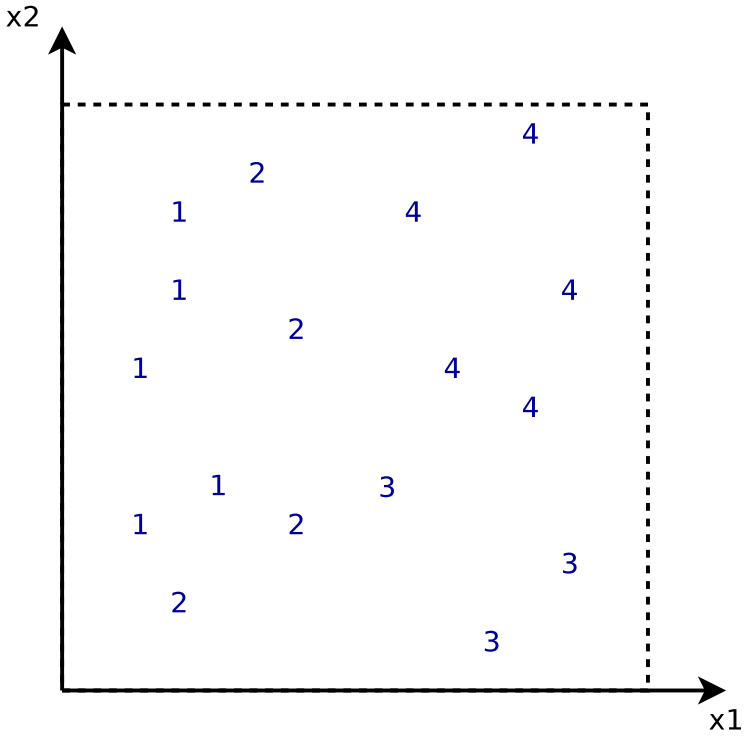
$$\gamma_{km} = \arg\min_{\gamma} \sum_{\mathbf{x}_i \in R_{km}} L(y_i, F_{m-1}(\mathbf{x}_i) + \gamma) \tag{9}$$

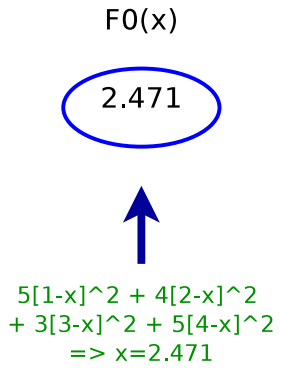
The approximation for $F$ in stage $m$ is then:

$$F_m(\mathbf{x}) = F_{m-1}(\mathbf{x}) + \eta \cdot \gamma_{km} 1(\mathbf{x}_i \in R_{km}) \tag{10}$$

The parameter $\eta$ controls the *learning rate* of the procedure.

# Learning Boosted Regression Trees (I)




Mean value minimizes the above expression

First, learn the most simple predictor that predicts a constant value minimizing the error for all training data.

# Calculating Optimal Leaf Value for $F_0$

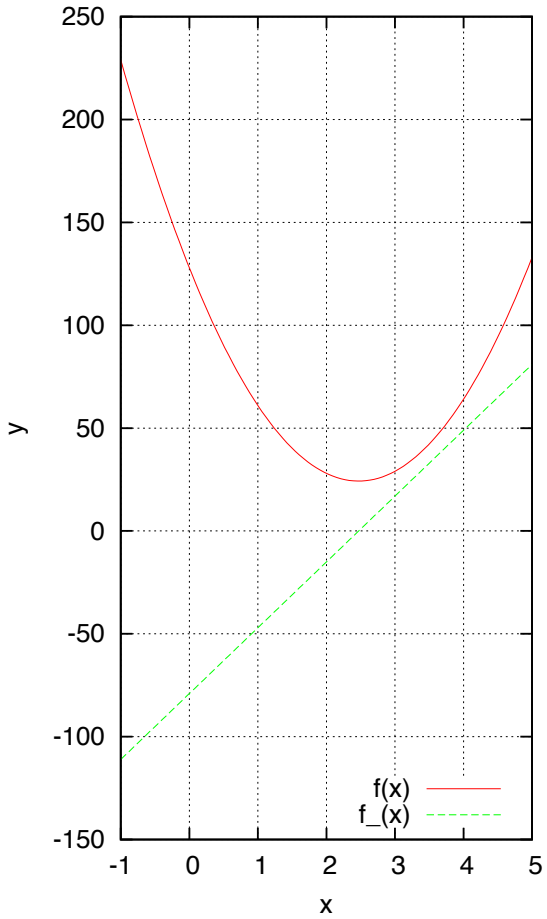
Recall the exp. coefficient: $\gamma_{km} = \arg\min_\gamma \sum_{\mathbf{x}_i \in R_{km}} L(y_i, F_{m-1}(\mathbf{x}_i) + \gamma)$

- Quadratic loss for the leaf (red):

$$\begin{aligned} f(x) =& 5 \cdot (1-x)^2 + 4 \cdot (2-x)^2 \\ &+ 3 \cdot (3-x)^2 + 5 \cdot (4-x)^2 \end{aligned}$$

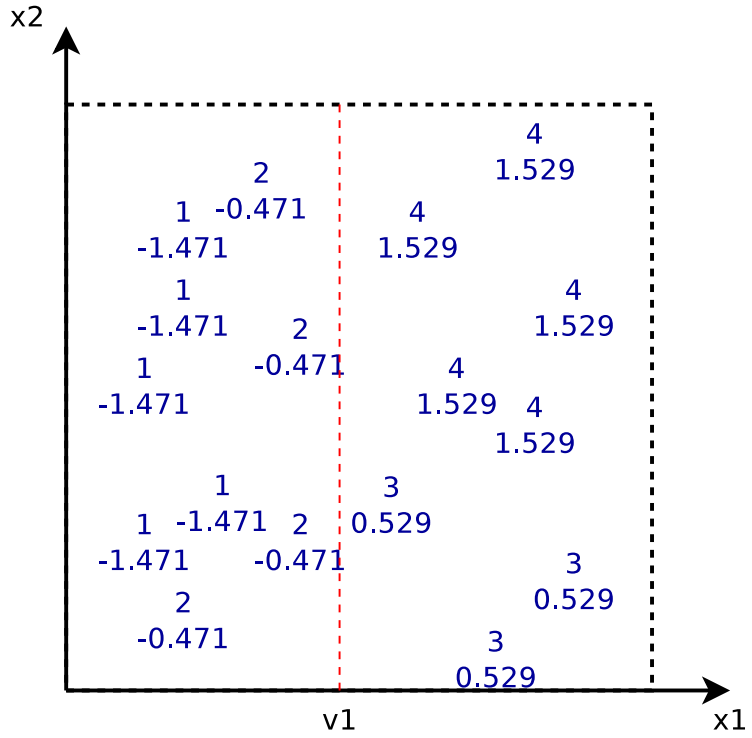
- $f(x)$ is quadratic, *convex*
  $\Rightarrow$ Optimum at $f'(x) = 0$ (green)

$$\begin{aligned} \frac{\partial f(x)}{\partial x} =& 5 \cdot (-2+2x) + 4 \cdot (-4+2x)^2 \\ &+ 3 \cdot (-6+2x)^2 + 5 \cdot (-8+2x)^2 \\ =& -84 + 34x = 32(x - 2.471) \end{aligned}$$

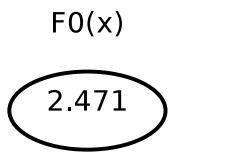



Mean value minimizes the square error loss

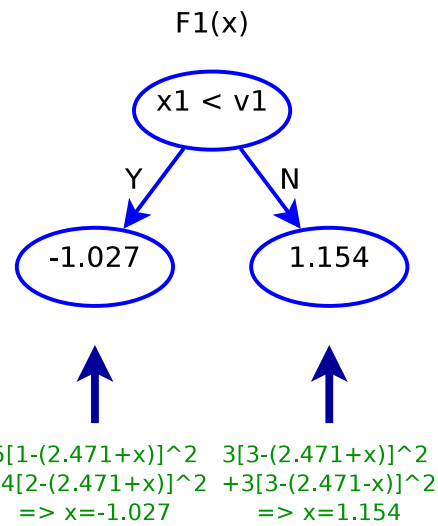
# Learning Boosted Regression Trees (II)




Try all possible v1 as



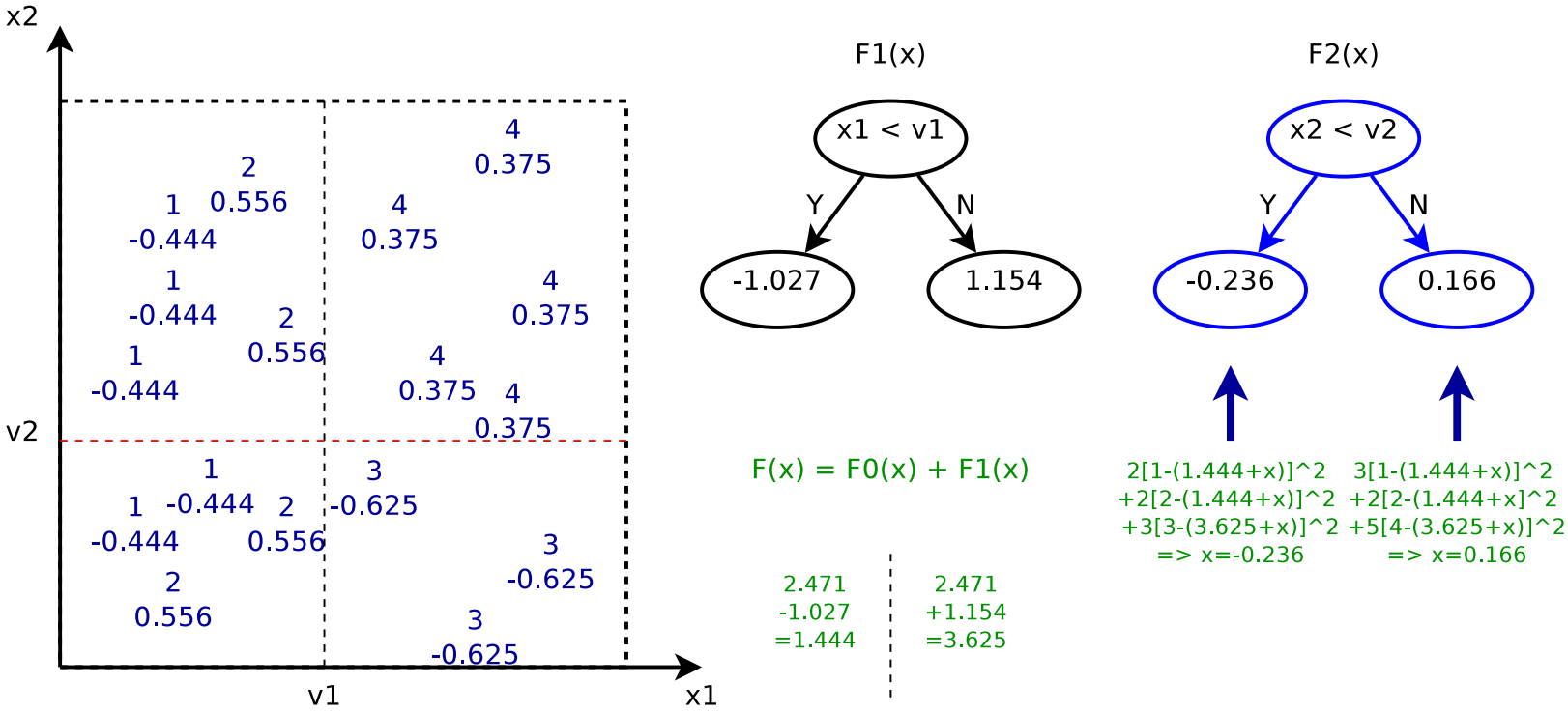

F(x) = F0(x) = 2.471




Split root node based on least squares criterion to build a tree predicting the "pseudo"-residuals.

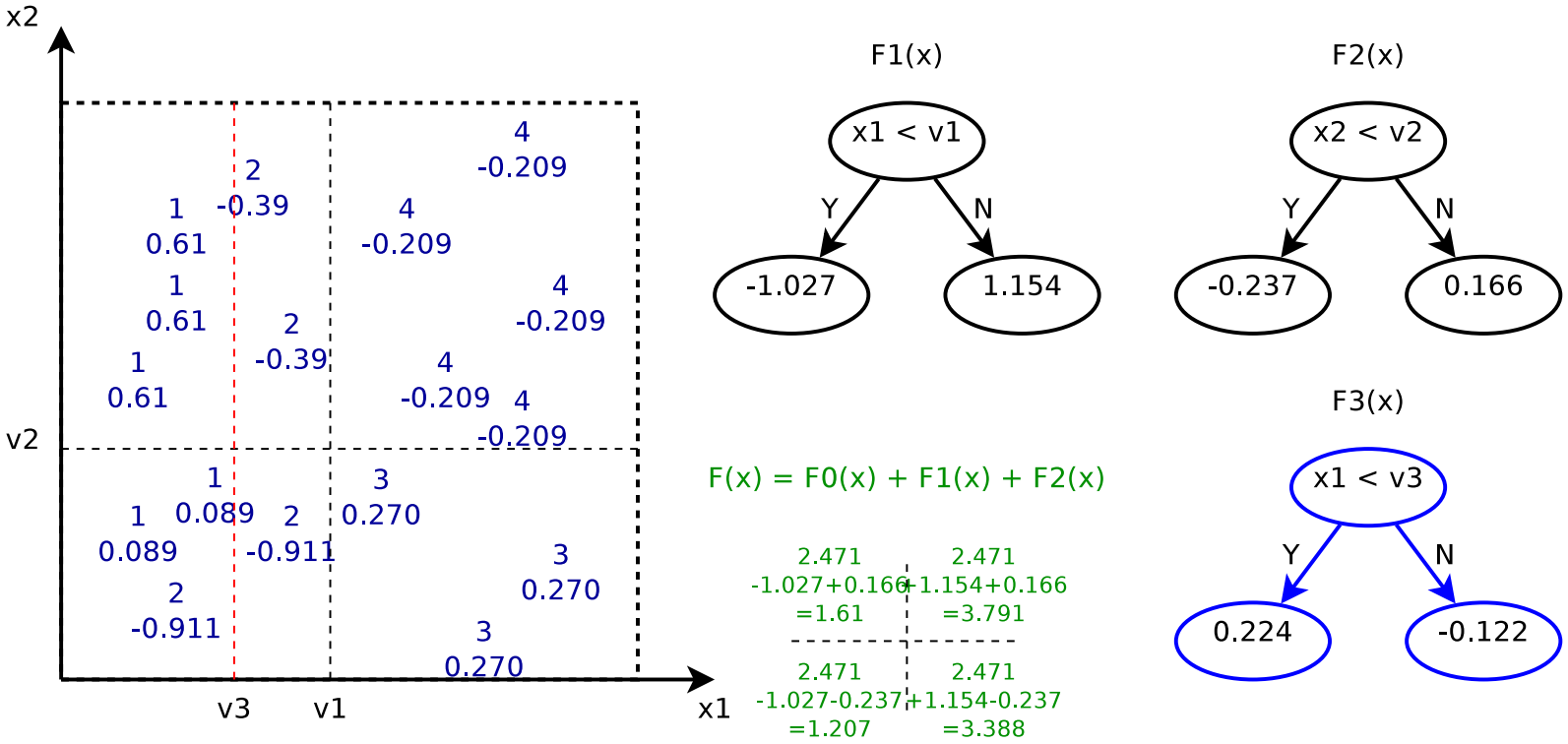
Psudo-residuals are: target value - previous prediction
= yi - F0(x).
For example, target 1, residual = 1-2.471

# Learning Boosted Regression Trees (III)




In the next stage, another tree is created to fit the actual "pseudo"-residuals predicted by the first tree.

# Learning Boosted Regression Trees (IV)




This is iteratively continued: in each stage, the algorithm builds a new tree based on the "pseudo"-residuals predicted by the previous tree ensemble.

# Multiple Additive Regression Trees (MART)

**Algorithm 1** Multiple Additive Regression Trees.

1: Initialize $F_0(\mathbf{x}) = \arg\min_\gamma \sum_{i=1}^{N} L(y_i, \gamma)$
2: **for** $m = 1, ..., M$ **do**
3: **for** $i = 1, ..., N$ **do**
4: $\tilde{y}_{im} = -\left[\frac{\partial L(y_i, F(\mathbf{x}_i))}{\partial F(\mathbf{x}_i)}\right]_{F(\mathbf{x})=F_{m-1}(\mathbf{x})}$
5: **end for**
6: $\{R_{km}\}_{k=1}^{K}$ // Fit a regression tree to targets $\tilde{y}_{im}$
7: **for** $k = 1, ..., K_m$ **do**
8: $\gamma_{km} = \arg\min_\gamma \sum_{x_i \in R_{jm}} L(y_i, F_{m-1}(\mathbf{x}_i) + \gamma)$
9: **end for**
10: $F_m(\mathbf{x}) = F_{m-1}(\mathbf{x}) + \eta \sum_{k=1}^{K_m} \gamma_{km} 1(\mathbf{x}_i \in R_{km})$
11: **end for**
12: Return $F_M(\mathbf{x})$

Revised targets= Pseudo-residual for each instance

Build a regression tree for revised targets

Compute the best prediction for each of K leaves

Add this new tree to the MART multiplied by a learning rate

# References I


Breiman, Leo and E. Schapire (2001). *Random forests*. In *Machine Learning*, pp. 5–32.

Burges, Christopher J. C. (2010). *From RankNet to LambdaRank to LambdaMART: An Overview*.

Burges, Christopher J. C., R. Ragno and Q. V. Le (2006). *Learning to Rank with Nonsmooth Cost Functions.*. In Schölkopf, Bernhard, J. Platt and T. Hoffman, eds.: *NIPS*, pp. 193–200. MIT Press.

Chapelle, Olivier and Y. Chang (2011). *Yahoo! Learning to Rank Challenge Overview.*. Journal of Machine Learning Research - Proceedings Track, 14:1–24.

Croft, W.B., D. Metzler and T. Strohmann (2010). *Search Engines: Information Retrieval in Practice*. Pearson, London, England.

# References II


Friedman, Jerome H. (1999). *Greedy Function Approximation: A Gradient Boosting Machine*. Annals of Statistics, 29(5):1189–1232.

Ganjisaffar, Yasser (2011). *Tree Ensembles for Learning to Rank*. PhD thesis, University of California, Irvine.

Hastie, Trevor, R. Tibshirani and J. Friedman (2002). *The Elements of Statistical Learning*. Springer, New York.

Liu, Tie-Yan (2010). *Learning to Rank for Information Retrieval.*. Springer-Verlag New York Inc.

Manning, Christopher D., P. Raghavan and H. Schütze (2008). *Introduction to Information Retrieval*. Cambridge University Press.

Vapnik, Vladimir N. (1995). *The Nature of Statistical Learning Theory*. Springer New York Inc., New York, NY, USA.

Wu, Qiang, C. J. C. Burges, K. M. Svore and J. Gao (2008). *Ranking, Boosting, and Model Adaptation*.