COMPSCI 589
Lecture 11: Neural Network, KNN and Tree-Based Regression

Benjamin M. Marlin

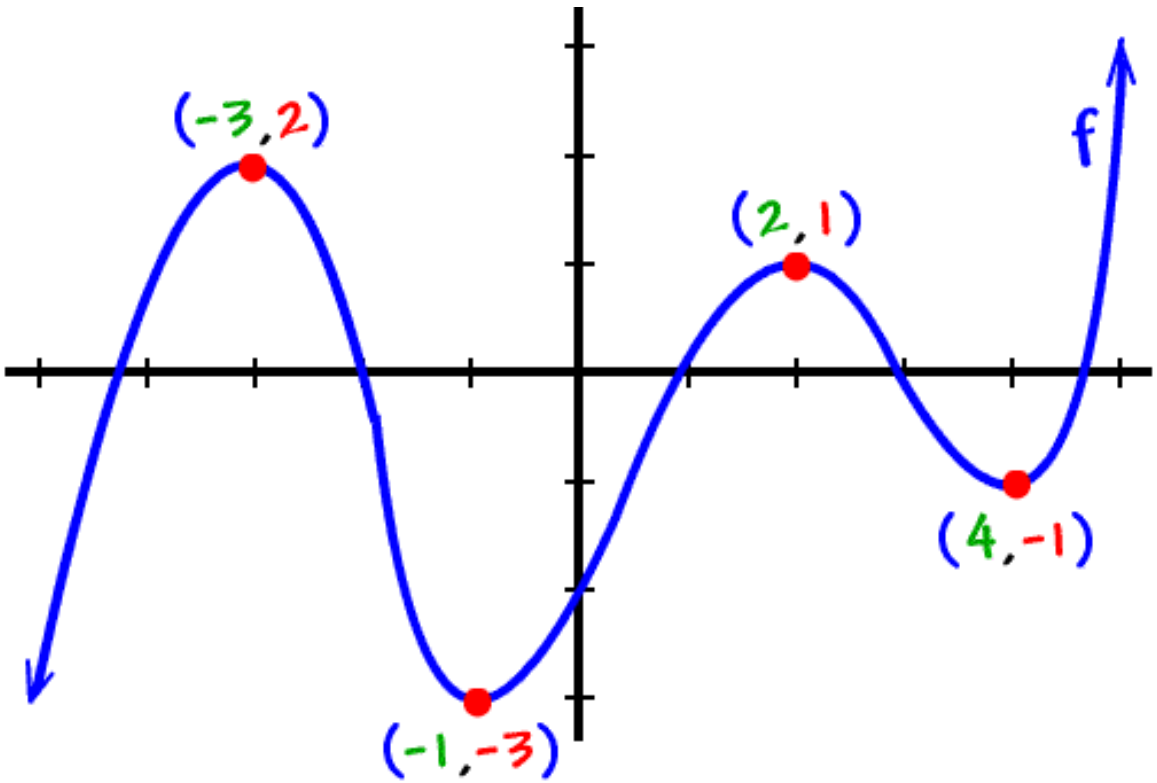
College of Information and Computer Sciences
University of Massachusetts Amherst

Slides by Benjamin M. Marlin (marlin@cs.umass.edu).
Created with support from National Science Foundation Award# IIS-1350522.

The Regression Task

Definition: The Regression Task

Given a feature vector $\mathbf{x} \in \mathbb{R}^D$, predict its corresponding output value y.



The Regression Learning Problem

Definition: Regression Learning Problem

Given a data set of example pairs $\mathcal{D} = \{(\mathbf{x}_i, y_i), i = 1 : N\}$ where $\mathbf{x}_i \in \mathbb{R}^D$ is a feature vector and $y_i \in \mathbb{R}$ is the output, learn a function $f : \mathbb{R}^D \to \mathbb{R}$ that accurately predicts y for any feature vector $\mathbf{x}$.

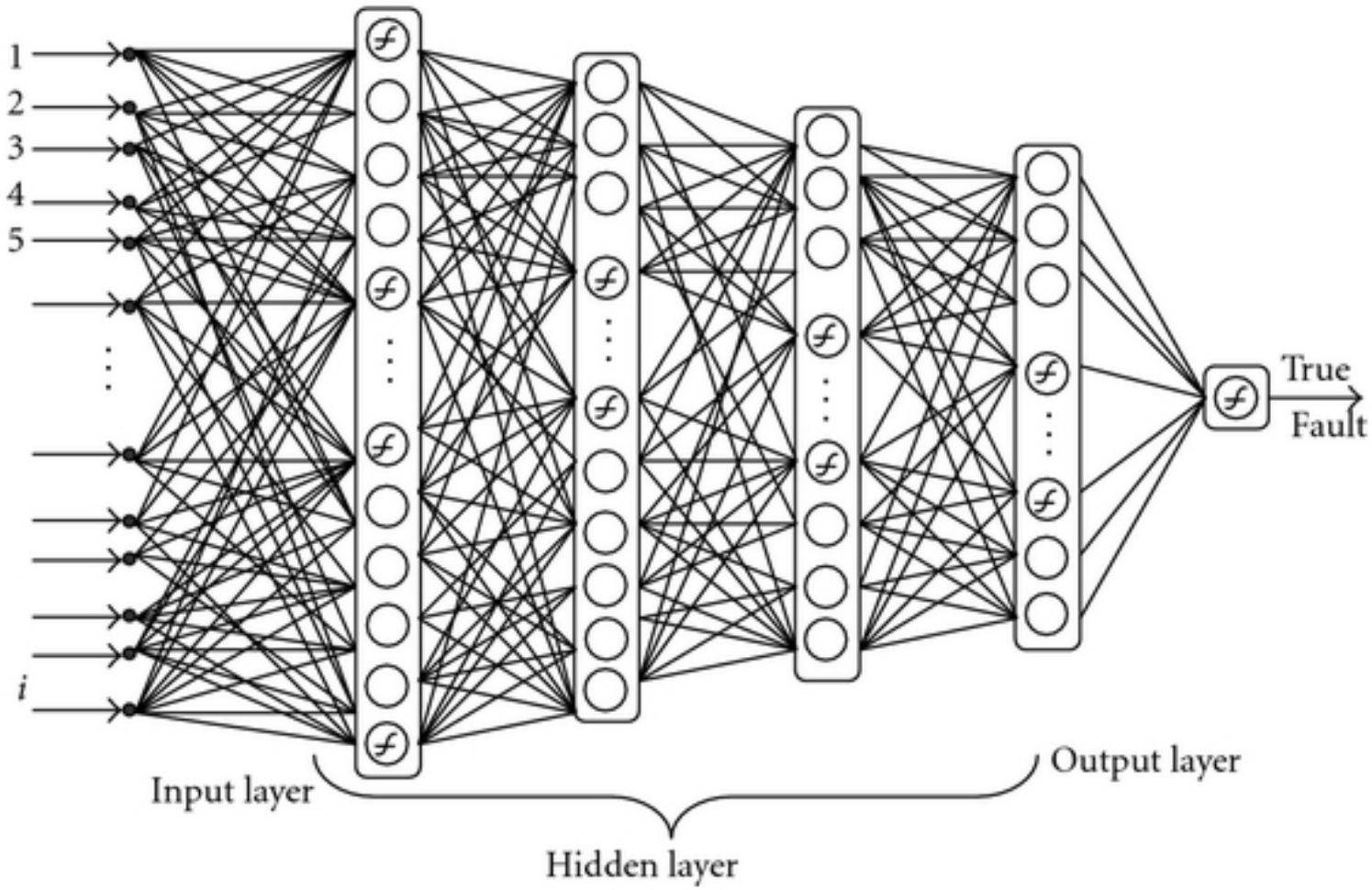
Error Measures: MSE

Definition: Mean Squared Error

Given a data set of example pairs $\mathcal{D} = \{(\mathbf{x}_i, y_i), i = 1 : N\}$ and a function $f : \mathbb{R}^D \to \mathcal{Y}$, the mean squared error of f on $\mathcal{D}$ is:

$$MSE(f, \mathcal{D}) = \frac{1}{N} \sum_{i=1}^{N} (y_i - f(\mathbf{x}_i))^2$$

Multi-Layer Perceptron



Neural Network Regression

To convert an MLP from classification to regression, we only need to change the output activation function from logistic to linear.

- The hidden layer non-linearities are smooth functions:

$$h_k^1 = \frac{1}{1+\exp(-(\sum_d w_{dk}^1 x_d + b_{dk}^1))}$$
$$h_k^i = \frac{1}{1+\exp(-(\sum_l w_{lk}^i h_l^{(i-1)} + b_{lk}^i))} \text{ for } i = 2, ..., L$$

- The output layer activation function is a linear function:

$$\hat{y} = \sum_l w_l^o h_l^L + b^o$$

Learning

- Let θ be the complete collection of parameters defining a neural network model. If MSE is the notion of performance that we care about, we can find the value of θ that minimizes the MSE on the training data set $\mathcal{D} = \{y_i, \mathbf{x}_i\}_{i=1:N}$

$$\mathcal{L}_{MSE}(\mathcal{D}|\theta) = \frac{1}{N}\sum_{n=1}^{N}(y_n - \hat{y}_n)^2$$

- To learn the model, we need the gradient of the objective function with respect to the weights.
- As with neural networks for classification, we can derive gradients by hand and implement the Backpropagation algorithm.
- Alternatively, we can implement the model using a library like PyTorch and use automatic differentiation to obtain gradients from the objective function.

Trade-Offs

- Neural network regression has low bias, but high variance.
- The objective function has local optima and requires iterative numerical optimization using backpropagation to compute the gradients, which can be slow. SGD and other learning enhancements apply.
- Making predictions with trained models can be very fast.
- Capacity control in these models can be crucial. The capacity parameters are the depth of the network and the size of each layer for MLPs.
- These models can also be trained using ℓ_2 or ℓ_1 regularization or the more recent dropout scheme as an alternative to controlling network structure.
- All of the architectures used for classification can also be used for regression including MLPs, CNNs, RNNs, and transformers.

K Nearest Neighbors Regression

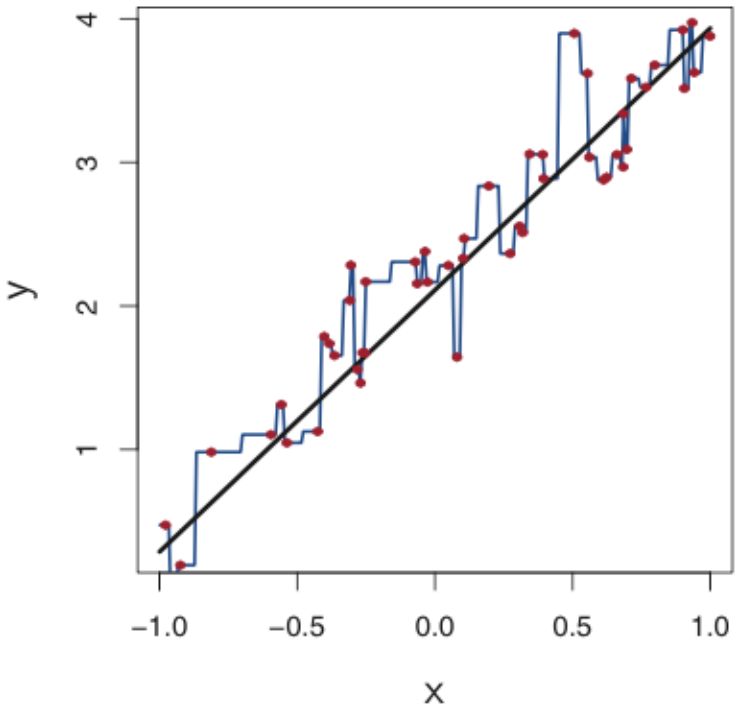
The KNN regression is a non-parametric regression method that simply stores the training data $\mathcal{D}$ and makes a prediction for each new instance $\mathbf{x}$ using an average over its set of K nearest neighbors $\mathcal{N}_K(\mathbf{x})$ computed using any distance function $d : \mathbb{R}^D \times \mathbb{R}^D \to \mathbb{R}$.

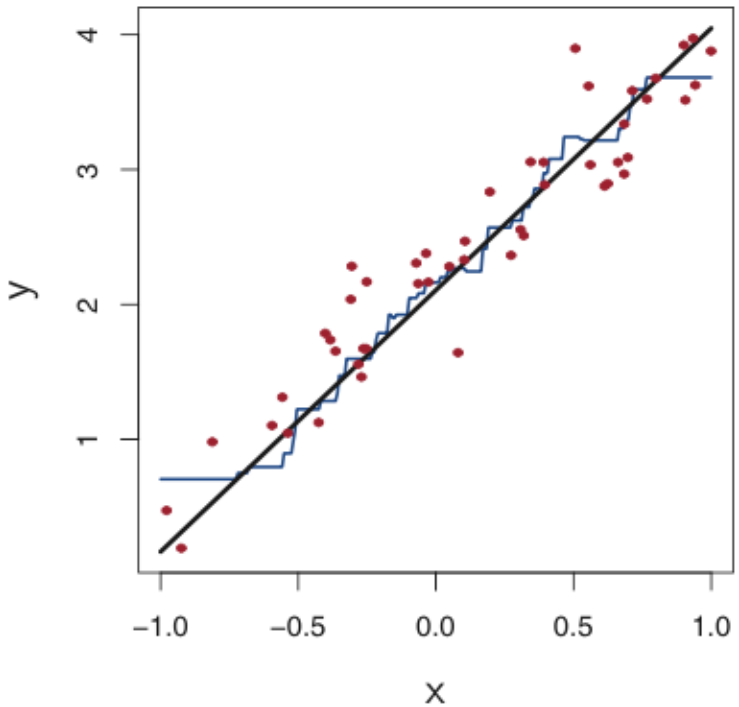
KNN Regression Function

$$f_{KNN}(\mathbf{x}) = \frac{1}{K} \sum_{i \in \mathcal{N}_K(\mathbf{x})} y_i$$

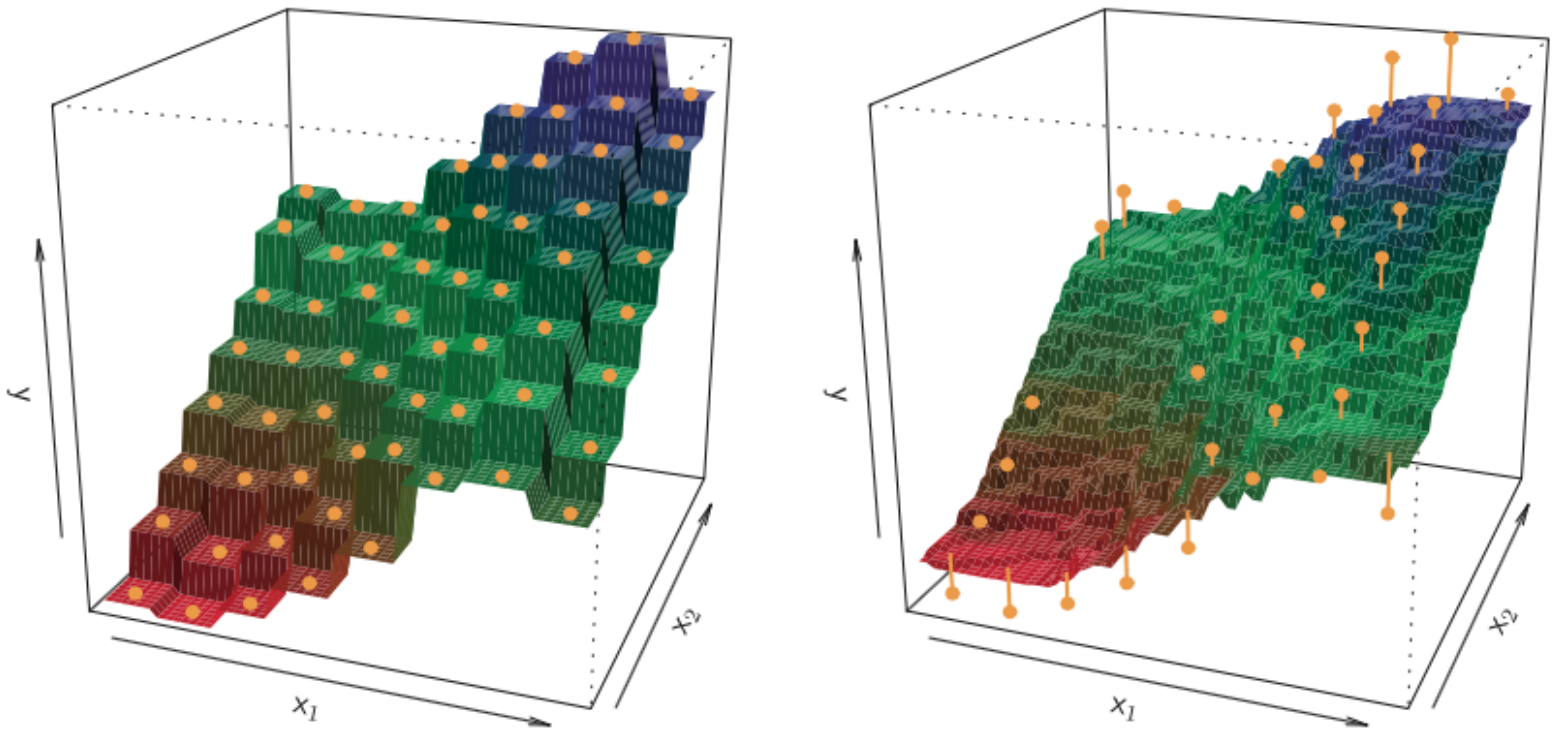
As with classification, use of KNN requires choosing the distance function d and the number of neighbors K.

Example: 1D KNN (K=1 vs K=9)





Example: 2D KNN (K=1 vs K=9)



Weighted KNN Regression

- Instead of giving all of the K neighbors equal weight in the average, a distance-weighted average can be used:

$$f_{KNN}(\mathbf{x}) = \frac{\sum_{i \in \mathcal{N}_K(\mathbf{x})} w_i y_i}{\sum_{i \in \mathcal{N}_K(\mathbf{x})} w_i}$$

$$w_i = \exp(-\alpha d_i)$$

Probabilistic KNN Regression

How could we make KNN regression probabilistic?

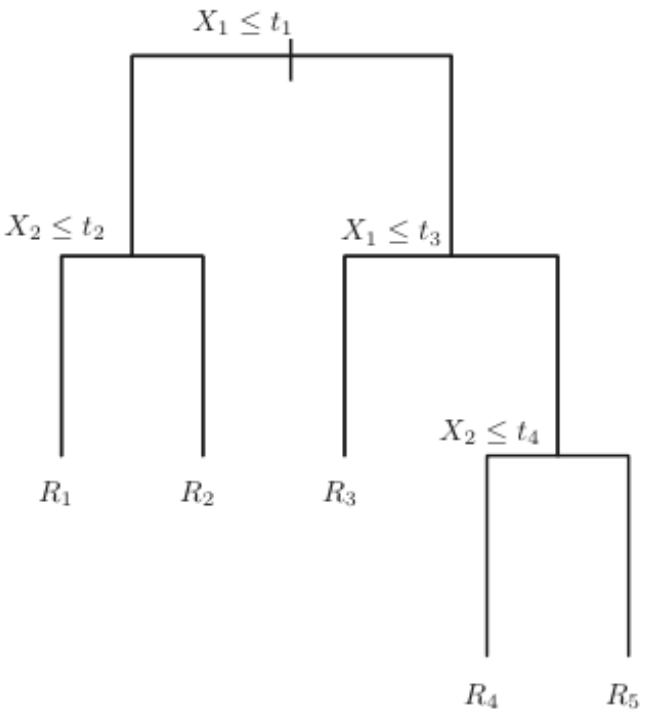
Trade-Offs

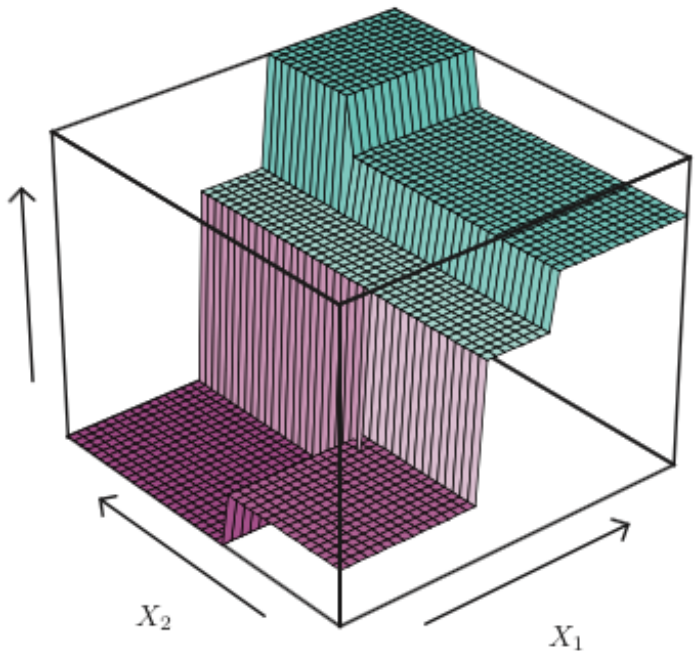
- KNN regression has low bias, but high variance.
- We must know what distance or similarity function to use for the method to perform well.
- However, due to the curse of dimensionality, KNN typically only works well when we have many training cases relative to the feature dimensionality.
- Even then, using KNN requires access to all of the training data when making predictions. Both the storage and computation required scales with the training data set size.

Regression Trees

- A regression tree makes predictions using a conjunction of rules organized into a binary tree structure.
- Each internal node in a regression tree contains a rule of the form $(x_d < t)$ or $(x_d = t)$ that tests a single data dimension d against a single threshold value t and assigns the data case to its left or right sub-tree according to the result.
- A data case is routed through the tree from the root to a leaf. Each leaf node is associated with a predicted output, and a data case is assigned the output of the leaf node it is routed to.

Example: 2D Regression Trees





Capacity Control

- The two primary hyper-parameters for a regression tree are maximum depth of the tree, and the minimum number of data cases in a leaf node.
- To set these hyper-parameters and evaluate the resulting model, we need to apply a train-validation-test experiment.

Building Regression Trees

Algorithm $BuildTree(Root, \mathcal{D}, h, minS, maxD)$

$d, t = BestSplit(\mathcal{D})$
$\mathcal{D}_1 \leftarrow \{(y_i, \mathbf{x}_i) | x_{id} \leq t\}, \mathcal{D}_2 = \{(y_i, \mathbf{x}_i) | x_{id} > t\}$
if $|\mathcal{D}_1| \leq minS$ or $h + 1 \geq maxD$ **then**
  $Root.RightChild.Prediction \leftarrow \frac{1}{|\mathcal{D}_1|} \sum_{y \in \mathcal{D}_1} y$
else
  $BuildTree(Root.RightChild, \mathcal{D}_1, h + 1, minS, maxD)$
if $|\mathcal{D}_2| \leq minS$ or $h + 1 \geq maxD$ **then**
  $Root.LeftChild.Prediction \leftarrow \frac{1}{|\mathcal{D}_2|} \sum_{y \in \mathcal{D}_2} y$
else
  $BuildTree(Root.LeftChild, \mathcal{D}_2, h + 1, minS, maxD)$
$Root.d \leftarrow d, Root.t \leftarrow t$
return $Root$

Finding the Best Split

Algorithm *BestSplit*$(\mathcal{D})$

for d from 1 to D **do**

$\mathbf{s} \leftarrow sort(\{x_{d1}, ..., x_{dN}\})$

for t in $\{(s_i + s_{i+1})/2 | i = 1...N-1\}$ **do**

$\mathcal{D}_1 \leftarrow \{(y_i, \mathbf{x}_i) | x_{id} \leq t\}$

$\mathcal{D}_2 \leftarrow \{(y_i, \mathbf{x}_i) | x_{id} > t\}$

$\bar{y}_1 \leftarrow \frac{1}{|\mathcal{D}_1|} \sum_{y \in \mathcal{D}_1} y$

$\bar{y}_2 \leftarrow \frac{1}{|\mathcal{D}_2|} \sum_{y \in \mathcal{D}_2} y$

$Score(d,t) \leftarrow \sum_{y \in \mathcal{D}_1} (y - \bar{y}_1)^2 + \sum_{y \in \mathcal{D}_2} (y - \bar{y}_2)^2$

$d, t \leftarrow \arg\min_{d', t'} Score(d,t)$

return (d,t)

Example: Building Regression Trees