Machine Learning for Predicting Numeric Values

Nysret Musliu

Database and Artificial Intelligence Group (DBAI)

Example 1

- Predicting CPU perfomance

	Cycle time (ns)	Main memory (Kb)		Cache (Kb)	Channels		Performance
	MYCT	MMIN	MMAX	CACH	CHMIN	CHMAX	PRP
1	125	256	6000	256	16	128	198
2	29	8000	32000	32	8	32	269
…							
208	480	512	8000	32	0	0	67
209	480	1000	4000	0	0	0	45

Example 2

- Predicting house prices in Vienna

Nr.Rooms	Size (m^2)	Age of house	…	Price (K)
1	35	1		110
4	120	20		250
3	85	10		235
3	78	2		210
2	54	5		170
…				

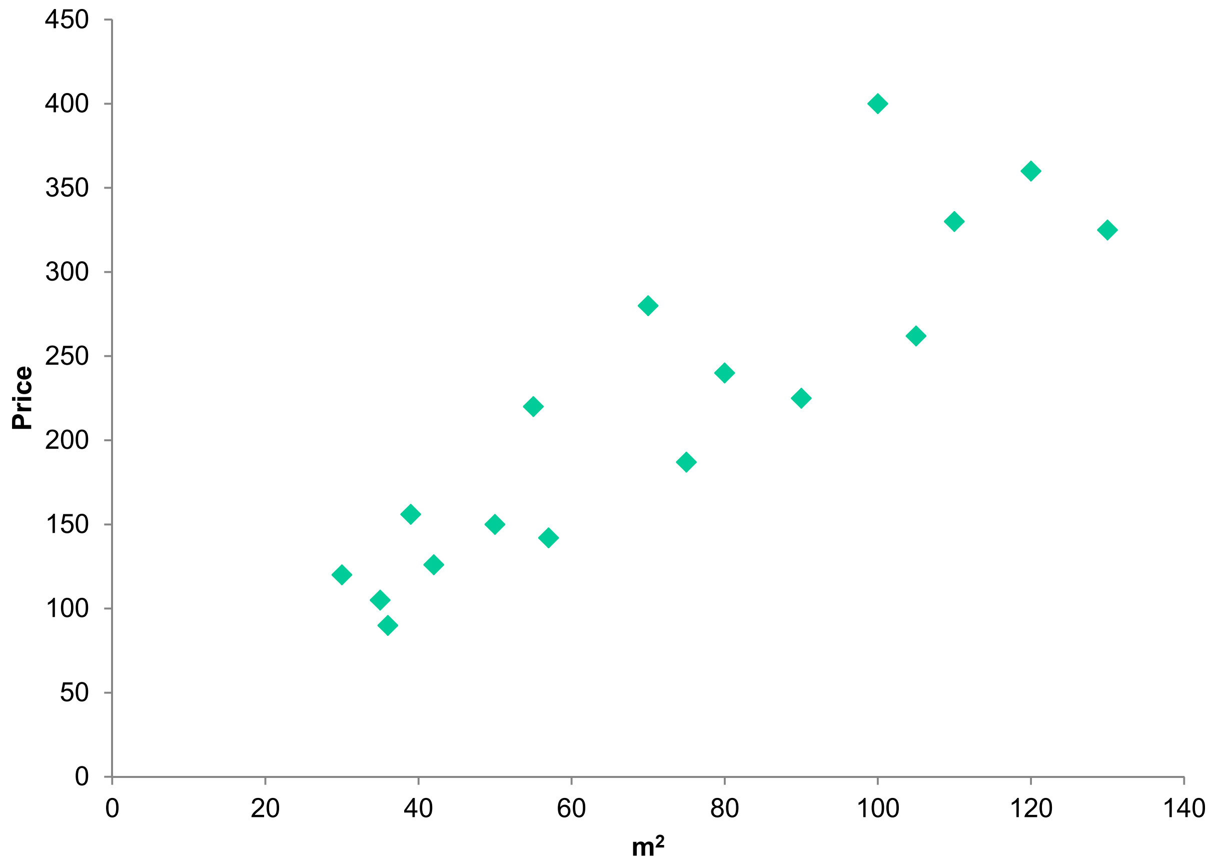
Example 3

- Predicting the running time of an algorithm (e.g. for solving of a SAT problem … See SATzilla)

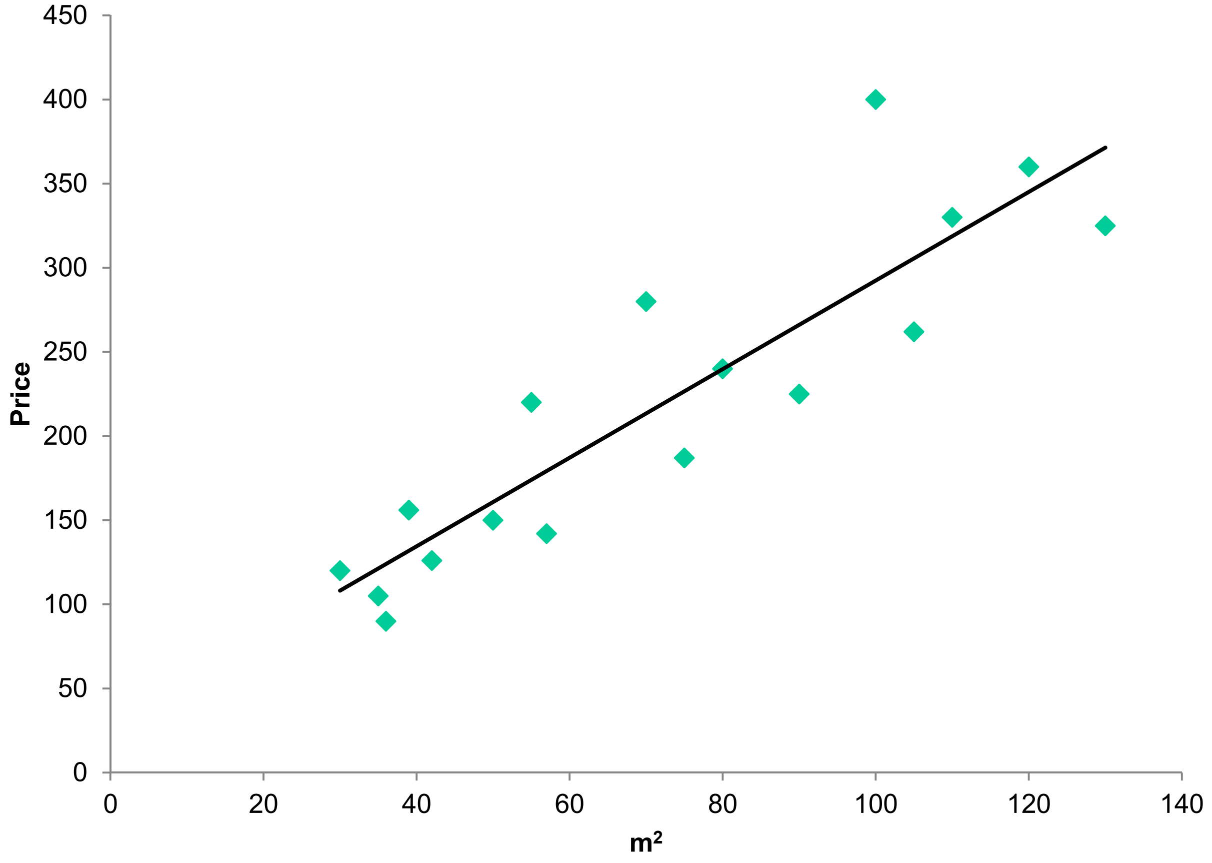
NrClauses	NrVariables	Ratio (C/V)	…	Time (sec)
100	80	1		10
4000	400	2		450
30000	8500	1		2350
300	78	2		25
2000	540	1		170
…				

- Many other examples (see UCI repository, Kaggle…)

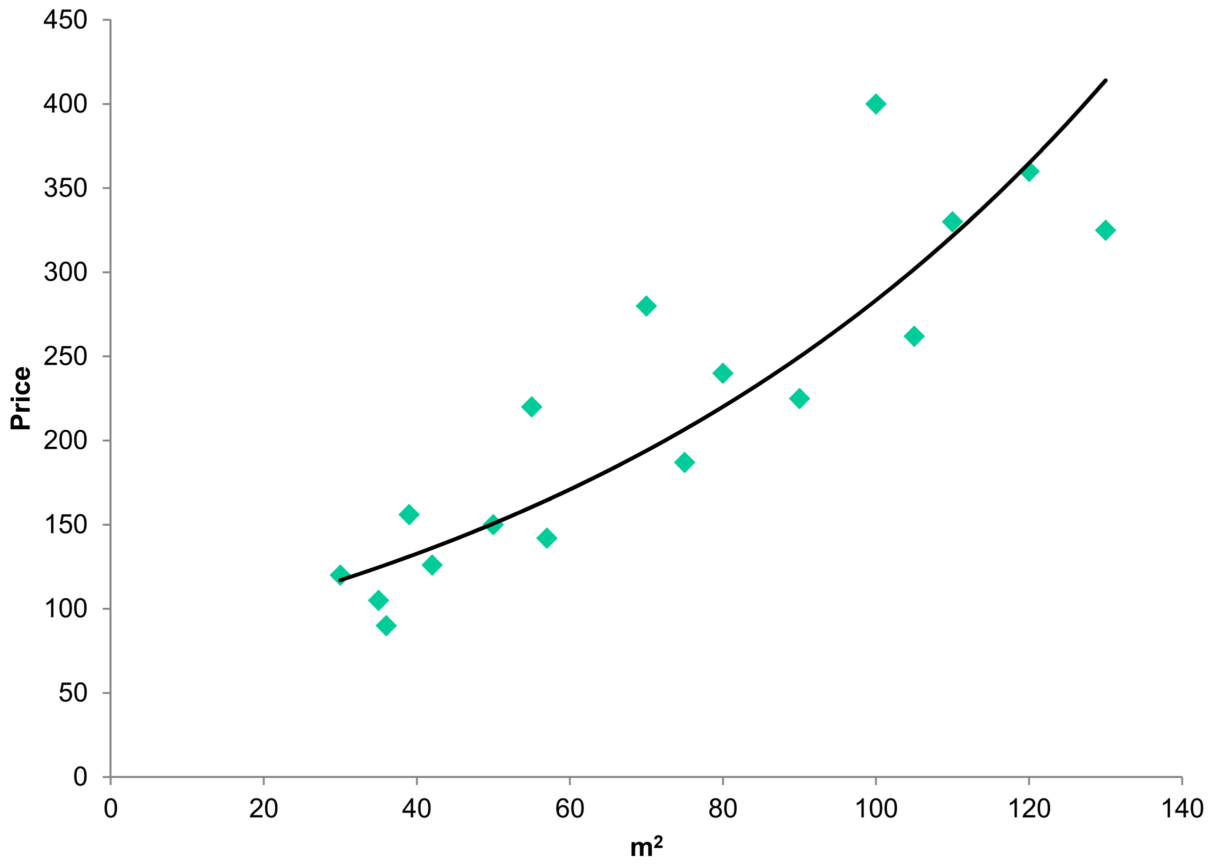
Prices vs. Square meter



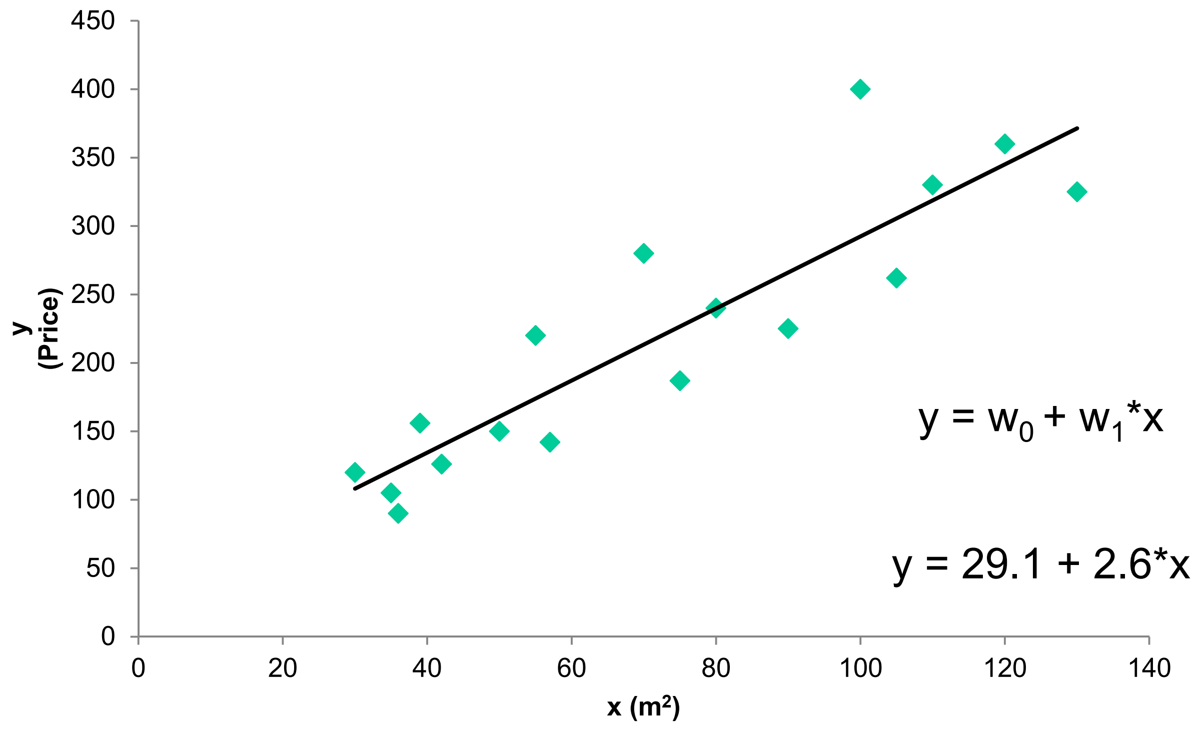
Model: Linear regression



Model: Other functions



Model: Linear regression



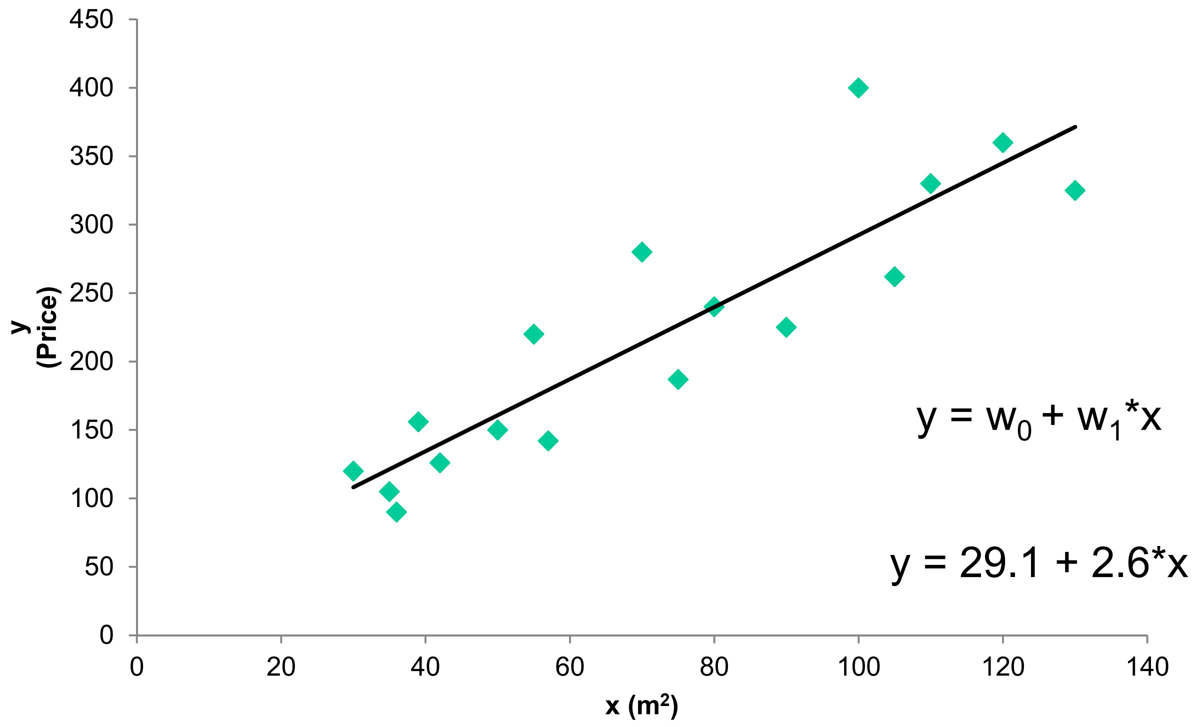
w_0, w_1 -> parameters

Problem: Find best values for w_0, w_1

Minimize some cost metric (sum of squared errors of prediction, …)

Residual sum of squares (RSS)

$$\text{RSS}(w_0, w_1) = \sum_{i=1}^{N} \left(y_i - (w_0 + w_1 * x_i)\right)^2$$



Finding best parameters

Minimize cost over all possible w_0, w_1

$$\min \mathrm{RSS}(w_0, w_1) = \sum_{i=1}^{N} (y_i - (w_0 + w_1 x_i))^2$$

Gradient descent algorithm

- Initial values for w_0, w_1
- Iterative change of these values
- Until convergence

Gradient descent

Repeat until convergence

$$w_0 = w_0 - \alpha \frac{\partial}{\partial w_0} RSS(w_0, w_1)$$

$$w_1 = w_1 - \alpha \frac{\partial}{\partial w1} RSS(w_0, w_1)$$

Update w_1 and w_0 simultaneously

α -> learning rate

Linear regression with multiple variables

- Gradient descent

Repeat until convergence

$$w_i = w_i - \alpha \frac{\partial}{\partial w_i} RSS(w_0, w_1, \ldots, w_n)$$

Update $w_0, w_1, \ldots, w_n$ simultaneously

Analytical solution for linear regression

Nr.Rooms	Size (m²)	Age	…	Price (K)
1	35	1		110
4	120	20		250
3	85	10		235
3	78	2		210
2	54	5		170
…				

X

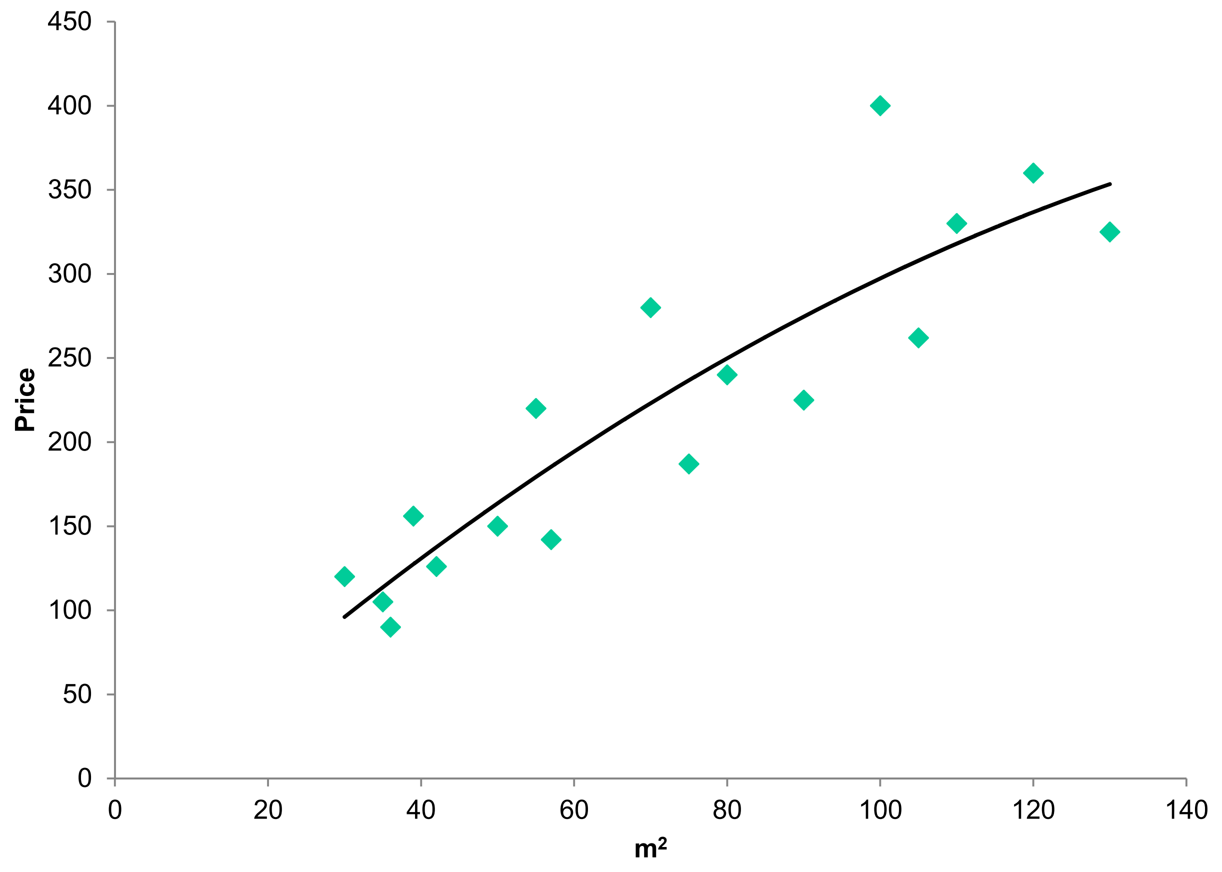
y

$$w = (X^T X)^{-1} X^T y$$

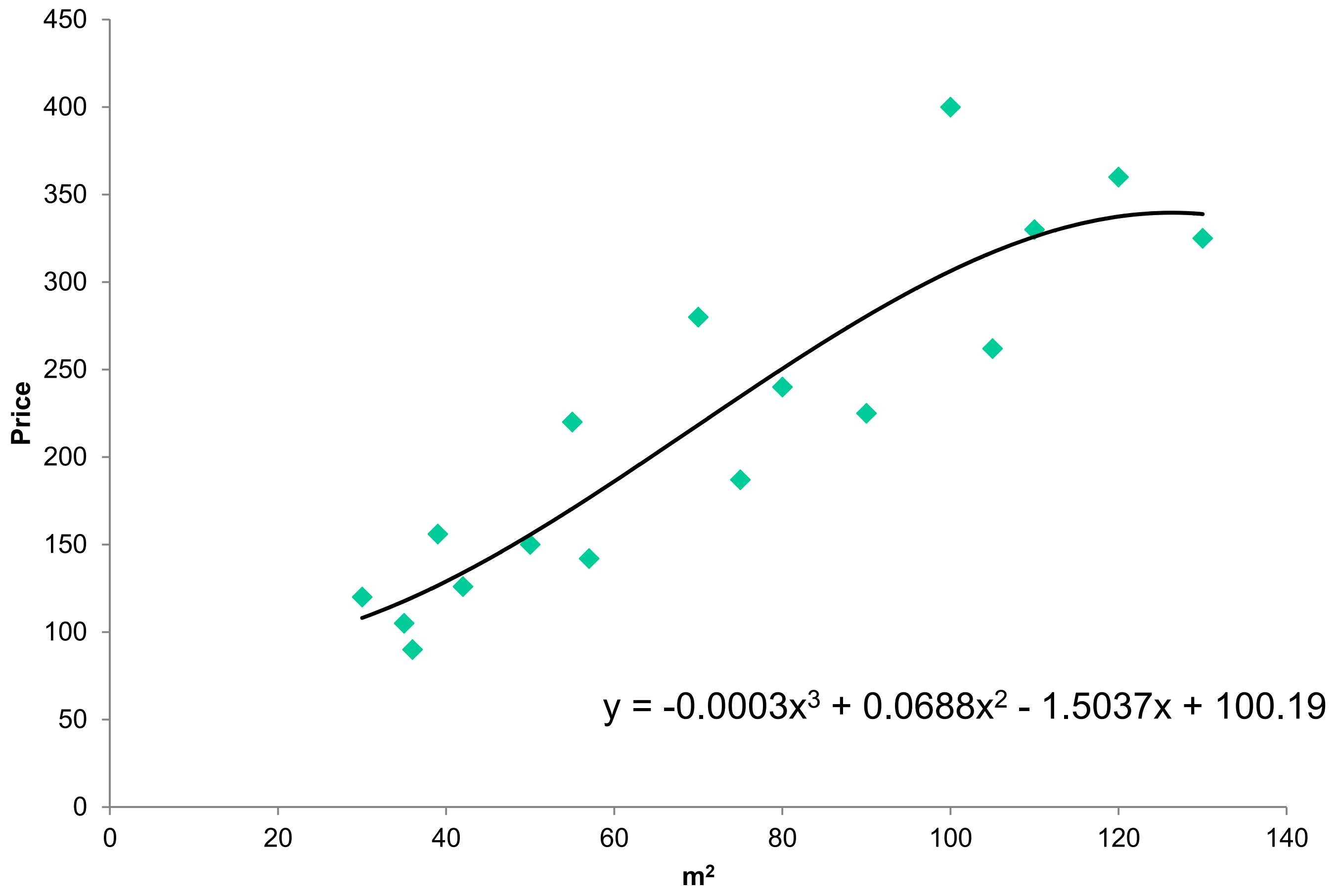
Gradient descent versus analytical solution

- Gradient Descent
  - Needs many iteration
  - Learning rate should be determined
  - Useful when the number of features is large
- Normal equation
  - No iterations
  - No need to choose learning rate
  - Works slow if the number of features is too large

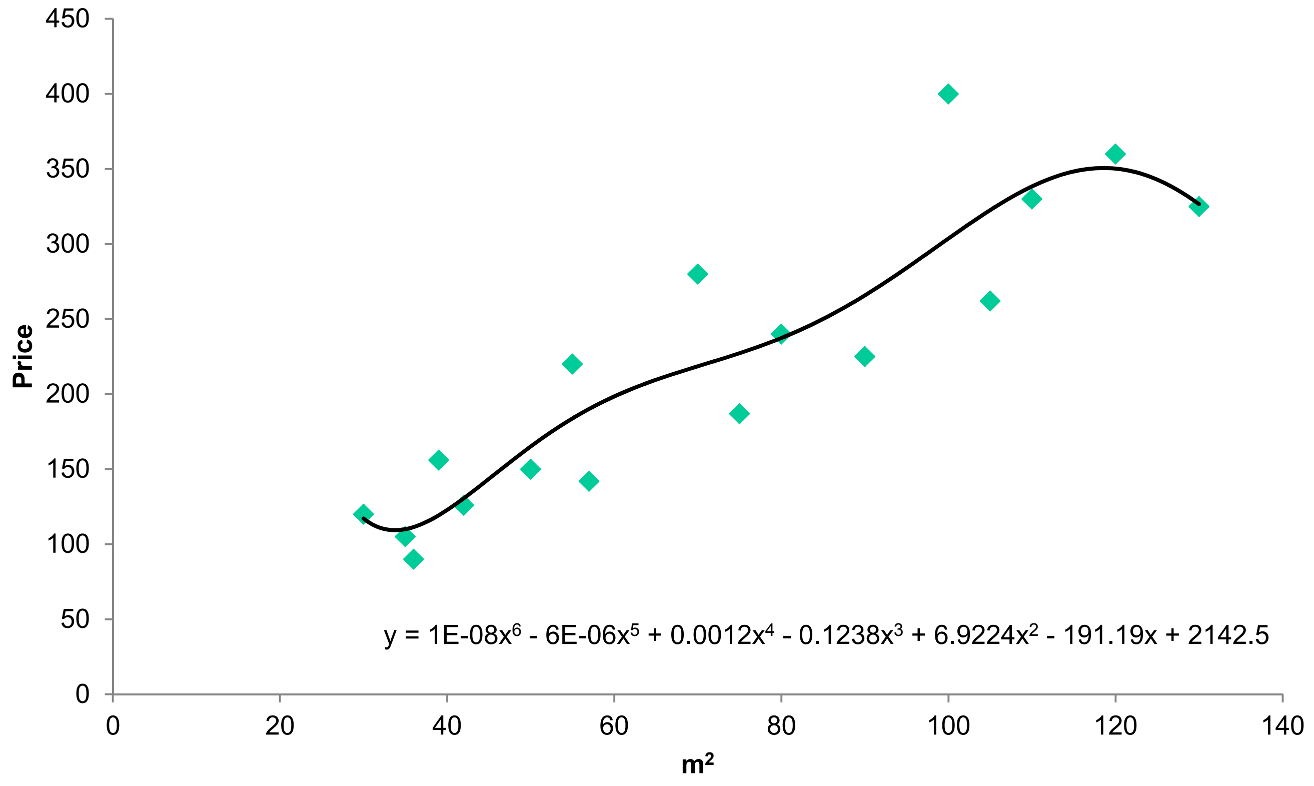
Polynomial regression



Polynomial regression



Polynomial regression



- High-order polynomials
  - Flexible
  - Problem of overfitting
  - Usually coefficients get very large

Ridge regression

- Try to avoid overfitting caused by high-order polynomials
- Objective:
  - Balance between
    - How well function fits the data
    - Magnitude of coefficients

Total cost=measure of fit + λ * measure of magnitude of coefficients

Selection of λ

- Large λ
  - High bias, low variance

- Small λ
  - Low bias, high variance

- λ controls the model complexity

Lasso Regression

- Feature selection task

Feature Selection

- Feature selection task
  – Feature selection: select features that minimize redundancy and maximize the relevance to the target

- Advantages of feature selection
  – Improved learning performance
  – Lower computational complexity
  – Decreased storage

Feature Selection: all subsets

- Enumerate all possible subsets
  - Subsets for 8 features

1	0	0	0	0	0	0	0

1	1	0	0	0	0	0	0
1	0	1	0	0	0	0	0
1	0	0	1	0	0	0	0
1	0	0	0	1	0	0	0

…

1	1	1	1	1	1	1	1

Too many possibilities:

$2^{\text{Number of features}}$

- The size of search space for *m* features is $O(2^m)$
- NP-hard problem
- Wrapper Model: Search strategies
  - Best-first search
  - Hill Climbing
  - Genetic Algorithms
  - Branch and Bound
  - Tabu Search
  - …

Greedy strategies

- Efficient and robust against overfitting
- Forward selection
  - Empty set of features
  - Features are added in the subset iteratively
- Backward elimination
  - Full set of features
  - Iterative elimination of less promising features

Greedy strategies

- Forward Search
  - F={ }
  - While the desired number of features arrived
  - For each feature f not in F
    - Estimate by cross-validation the error of model on F U f
  - Add f with lowest error to F
- Backward elimination
  - F={All features}
  - While not reduced to desired number of features
  - For each feature f in F:
    - Estimate by cross-validation the error of model on F/f
  - Remove from F the feature f with the highest error

Other techniques

- Metahuristic based techniques
- Wrapper models give better results than filter models
- Computationally expensive
- Better performance for the predefined classifier

Regularization for feature selection

- Use regularization to eliminate some features
- Some coefficients should get *exactly 0*
- Some features are not used
- Features are selected if the coefficients are non-zero

Using regularization for feature selection

- Ridge Regression (L_2 regularized regression)

Total cost=measure of fit + λ * measure of magnitude of coefficients

Total cost=RSS(w_0, …,w_n) + λ * $||w||^2$

$||w||^2 = w_0^2 + \ldots + w_n^2$

Encourages small weights but not exactly 0

Ridge Regression (L_2 regularized regression)

Total cost=measure of fit + λ * measure of magnitude of coefficients

Total cost=RSS(w_0, …,w_n) + λ * $||w||_2^2$

$||w||_2^2 = w_0^2 + \ldots + w_n^2$

Encourages small weights but not exactly 0

Set small ridge coefficients to 0

Lasso Regression (L_1 regularized regression)

Total cost=measure of fit + λ * measure of magnitude of coefficients

Total cost=$RSS(w_0, \ldots, w_n) + \lambda * ||w||_1$

$||w||_1 = |w_0| + \ldots + |w_n|$

k-NN Regression

- Data set of pairs $(x_1, y_1), (x_2, y_2), \dots, (x_n, y_n)$
- $y_1, y_2, \dots, y_n$ -> numeric values
- Prediction for the new instance x_m with k-NN:
  - Similar to k-NN for classification
  - Find k closest x_i in the data set: $x_{n1}, x_{n2}, \dots, x_{ni}$
  - Predict

  $y_m = 1/k(y_{n1}, y_{n2}, \dots, y_{ni})$

Evaluating numeric prediction

- Same strategies: independent test set, cross-validation, significance tests, etc.
- Difference: error measures
- Actual target values: $a_1\ a_2 \ldots a_n$
- Predicted target values: $p_1\ p_2 \ldots\ p_n$
- Most popular measure: *mean-squared error*

$$\frac{(p_1 - a_1)^2 + \cdots + (p_n - a_n)^2}{n}$$

  - Easy to manipulate mathematically

Slides 32-49 are taken from this book: Data Mining Practical Machine Learning Tools and Techniques by I. H. Witten, E. Frank, M. A. Hall and C. J. Pal

Other measures

- The *root mean-squared error* :

$$\sqrt{\frac{(p_1-a_1)^2+\cdots+(p_n-a_n)^2}{n}}$$

- The *mean absolute error* is less sensitive to outliers than the mean-squared error:

$$\frac{|p_1-a_1|+\cdots+|p_n-a_n|}{n}$$

- Sometimes *relative* error values are more appropriate (e.g. 10% for an error of 50 when predicting 500)

Improvement on the mean

- How much does the scheme improve on simply predicting the average?
- The *relative squared error* is:

$$\frac{(p_1 - a_1)^2 + \dots + (p_n - a_n)^2}{(a_1 - \bar{a})^2 + \dots + (a_n - \bar{a})^2}$$

(in this formula and the following two, $\bar{a}$ is the mean value over the training data)

- The *root relative squared error* and the *relative absolute error* are:

$$\sqrt{\frac{(p_1 - a_1)^2 + \dots + (p_n - a_n)^2}{(a_1 - \bar{a})^2 + \dots + (a_n - \bar{a})^2}} \qquad \frac{|p_1 - a_1| + \dots + |p_n - a_n|}{|a_1 - \bar{a}| + \dots + |a_n - \bar{a}|}$$

Correlation coefficient

- Measures the *statistical correlation* between the predicted values and the actual values

$$\frac{S_{PA}}{\sqrt{S_P S_A}}, \text{ where } S_{PA} = \frac{\sum_i (p_i - \bar{p})(a_i - \bar{a})}{n-1}, \; S_P = \frac{\sum_i (p_i - \bar{p})^2}{n-1},$$

$$S_A = \frac{\sum_i (a_i - \bar{a})^2}{n-1} \text{ (here, } \bar{a} \text{ is the mean value over the test data)}$$

- Scale independent, between –1 and +1
- Good performance leads to large values!

Which measure?

- Best to look at all of them
- Often it doesn't matter
- Example:

	A	B	C	D
Root mean-squared error	67.8	91.7	63.3	57.4
Mean absolute error	41.3	38.5	33.4	29.2
Root rel squared error	42.2%	57.2%	39.4%	35.8%
Relative absolute error	43.1%	40.1%	34.8%	30.4%
Correlation coefficient	0.88	0.88	0.89	0.91

- D best, C second-best
- A, B arguable

Regression trees

- Like decision trees, but:
  - Splitting criterion: minimize intra-subset variation
  - Termination criterion: std. dev. becomes small
  - Pruning criterion: based on numeric error measure
  - Prediction: Leaf predicts average class value of instances
- Yields piecewise constant functions
- Easy to interpret
- More sophisticated version: *model trees*

Example Predicting CPU performance

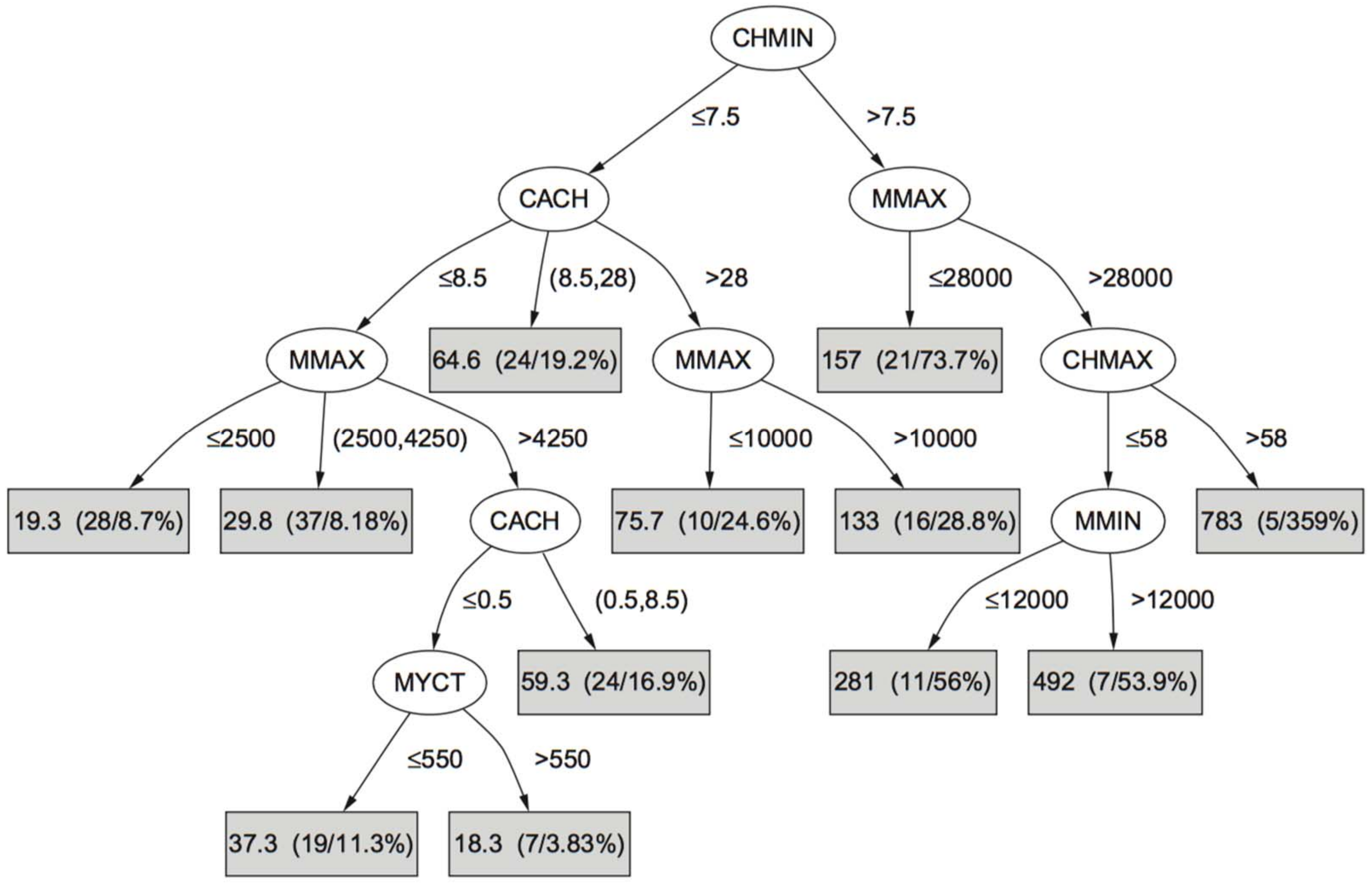
- Example: 209 different computer configurations

	Cycle time (ns)	Main memory (Kb)		Cache (Kb)	Channels		Performance
	MYCT	MMIN	MMAX	CACH	CHMIN	CHMAX	PRP
1	125	256	6000	256	16	128	198
2	29	8000	32000	32	8	32	269
…							
208	480	512	8000	32	0	0	67
209	480	1000	4000	0	0	0	45

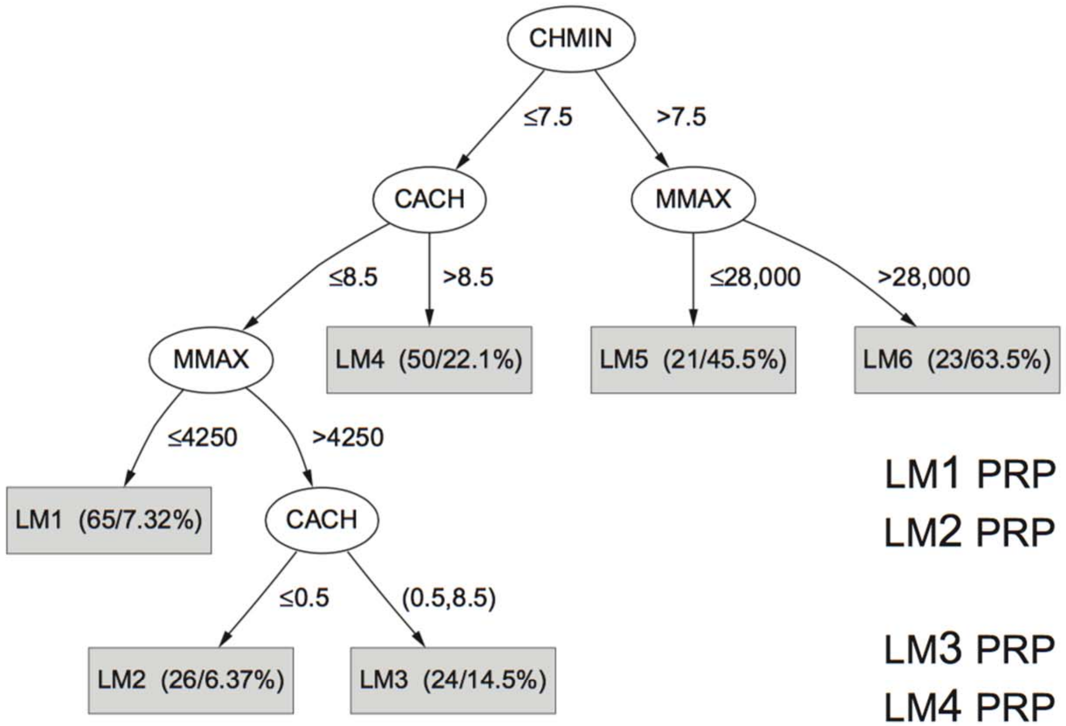
- Linear regression function

```
PRP = -55.9 + 0.0489 MYCT + 0.0153 MMIN + 0.0056 MMAX
      + 0.6410 CACH - 0.2700 CHMIN + 1.480 CHMAX
```

Regression tree for the CPU data



Model tree for the CPU data



LM1 PRP = 8.29 + 0.004 MMAX + 2.77 CHMIN
LM2 PRP = 20.3 + 0.004 MMIN – 3.99 CHMIN
+ 0.946 CHMAX
LM3 PRP = 38.1 + 0.012 MMIN
LM4 PRP = 19.5 + 0.002 MMAX + 0.698 CACH
+ 0.969 CHMAX
LM5 PRP = 285 1.46 MYCT + 1.02 CACH
– 9.39 CHMIN
LM6 PRP = – 65.8 + 0.03 MMIN – 2.94 CHMIN
+ 4.98 CHMAX

Model trees

- Build a regression tree
- Each leaf ⇒ linear regression function
- Smoothing: factor in ancestor's predictions
  - Smoothing formula: $p' = \frac{np + kq}{n + k}$
  - Same effect can be achieved by incorporating ancestor models into the leaves
- p' -> the prediction passed up to the next higher node
- p -> prediction passed to this node from below
- q -> the value predicted by the model at this node
- n -> number of training instances that reach the node below
- k -> smoothing constant

Model trees

- Build a regression tree
- Each leaf $\Rightarrow$ linear regression function
- Smoothing: factor in ancestor's predictions
  - Smoothing formula: $p' = \dfrac{np + kq}{n + k}$
  - Same effect can be achieved by incorporating ancestor models into the leaves
- Need linear regression function at each *node*
- At each node, use only a subset of attributes to build linear regression model
  - Those occurring in subtree
  - (+maybe those occurring in path to the root)
- Fast: tree usually uses only a small subset of the attributes

Building the tree

- Splitting: standard deviation reduction $$SDR = sd(T) - \sum_i \left|\frac{T_i}{T}\right| \times sd(T_i)$$

- Termination of splitting process:
  - Standard deviation < 5% of its value on full training set
  - Too few instances remain (e.g., < 4)

Pruning:

- Heuristic estimate of absolute error of linear regression models:

$$\frac{n+v}{n-v} \times \text{average\_absolute\_error}$$

- Greedily remove terms from LR models to minimize estimated error
- Proceed bottom up: compare error of LR model at internal node to error of subtree (this happens before smoothing is applied)
- Heavy pruning: single model may replace whole subtree

Nominal attributes

- Convert nominal attributes to binary ones
  - Sort attribute values by their average class values
  - If attribute has *k* values,
    generate *k* – 1 binary attributes
    - *i* th attribute is 0 if original nominal value is part of the first *i* nominal values in the sorted list, and 1 otherwise
- Treat binary attributes as numeric in linear regression models and when selecting splits
- Can prove: best SDR split on one of the new binary attributes is the best (binary) SDR split on original nominal attribute
- In practice this process is not applied at every node of the tree but globally at the root node of the tree
  - Splits are no longer optimal but runtime and potential for overfitting are reduced this way

Pseudo-code for M5'

- Let us consider the pseudo code for the model tree inducer M5’
- Four methods:
  - Main method: *MakeModelTree*
  - Method for splitting: *split*
  - Method for pruning: *prune*
  - Method that computes error: *subtreeError*
- We will briefly look at each method in turn
- We will assume that the linear regression method performs attribute subset selection based on error (discussed previously)
- Nominal attributes are replaced globally at the root node

MakeModelTree

```
MakeModelTree (instances)
{
  SD = sd(instances)
  for each k-valued nominal attribute
    convert into k-1 synthetic binary attributes
  root = newNode
  root.instances = instances
  split(root)
  prune(root)
  printTree(root)
}
```

split

```
split(node)
{
  if sizeof(node.instances) < 4 or
     sd(node.instances) < 0.05*SD
    node.type = LEAF
  else
    node.type = INTERIOR
    for each attribute
      for all possible split positions of attribute
        calculate the attribute's SDR
    node.attribute = attribute with maximum SDR
    split(node.left)
    split(node.right)
}
```

prune

```
prune(node)
{
  if node = INTERIOR then
    prune(node.leftChild)
    prune(node.rightChild)
    node.model = linearRegression(node)
    if subtreeError(node) > error(node) then
      node.type = LEAF
}
```

subtreeError

```
subtreeError(node)
{
  l = node.left; r = node.right
  if node = INTERIOR then
    return (sizeof(l.instances)*subtreeError(l)
                + sizeof(r.instances)*subtreeError(r))
                 /sizeof(node.instances)
  else return error(node)
}
```