Chapter 7:
Numerical Prediction

Lecture: Prof. Dr. Thomas Seidl

Tutorials: Julian Busch, Evgeniy Faerman, Florian Richter, Klaus Schmid

© for the original version: Jörg Sander and Martin Ester, Jiawei Han and Micheline Kamber
Chapter 7: Numerical Prediction

1) Introduction
   - Numerical Prediction problem, linear and nonlinear regression, evaluation measures

2) Piecewise Linear Numerical Prediction Models
   - Regression Trees, axis parallel splits, oblique splits
   - Hinging Hyperplane Models

3) Bias-Variance Problem
   - Regularization, Ensemble methods
Numerical Prediction

• Related problem to classification: **numerical prediction**
  – Determine the numerical value of an object
  – Method: e.g., regression analysis
  – Example: prediction of flight delays

• Numerical prediction is **different** from classification
  – Classification refers to predict categorical class label
  – Numerical prediction models continuous-valued functions

• Numerical prediction is **similar** to classification
  – First, construct a model
  – Second, use model to predict unknown value
    • Major method for numerical prediction is regression
      – Linear and multiple regression
      – Non-linear regression
Examples

- Housing values in suburbs of Boston
  - Inputs
    - number of rooms
    - Median value of houses in the neighborhood
    - Weighted distance to five Boston employment centers
    - Nitric oxides concentration
    - Crime rate per capita
    - ...
  - Goal: compute a model of the housing values, which can be used to predict the price for a house in that area
Examples

• Control engineering:
  – Control the inputs of a system in order to lead the outputs to a given reference value
  – Required: a model of the process

![Diagram of control system]
- Controller
- Process
- Inputs (manipulated variables)
- Optimization
- Process model
- Measured outputs

Wind turbine

Diesel engine

Numerical Prediction → Introduction
Examples

• Fuel injection process:
  – database of spray images
  – Inputs: settings in the pressure chamber
  – Outputs: spray features, e.g., penetration depth, spray width, spray area

compute a model which predicts the spray features, for input settings which have not been measured
Numerical Prediction

• Given: a set of observations
• Compute: a generalized model of the data which enables the prediction of the output as a continuous value

Quality measures:
– Accuracy of the model
– Compactness of the model
– Interpretability of the model
– Runtime efficiency (training, prediction)
Linear Regression

• Given a set of $N$ observations with inputs of the form $\mathbf{x} = [x_1, \ldots, x_d]$ and outputs $y \in \mathbb{R}$

• Approach: minimize the **Sum of Squared Errors (SSE)**

• Numerical Prediction: describe the outputs $y$ as a linear equation of the inputs

$$\hat{y} = f(\mathbf{x}) = \beta_0 + \beta_1 \cdot x_1 + \ldots + \beta_d \cdot x_d = [1 \; x_1 \; \ldots \; x_d] \cdot \begin{bmatrix} \beta_0 \\ \beta_1 \\ \vdots \\ \beta_d \end{bmatrix} = [1 \; x_1 \; \ldots \; x_d] \cdot \mathbf{\beta}$$

• Train the parameters $\mathbf{\beta} = [\beta_0 \; \beta_1 \; \ldots \; \beta_d]$:

$$\sum_{i=1}^{N} (y_i - f(\mathbf{x}_i))^2 \rightarrow min$$

$$d = 1: \quad y = 0,5645 \cdot x + 1,2274$$

Numerical Prediction → Introduction
Linear Regression

• Matrix notation: let $X \in \mathbb{R}^{N \times (d+1)}$ be the matrix containing the inputs, $Y \in \mathbb{R}^N$ the outputs, and $\beta$ the resulting coefficients:

$$X = \begin{bmatrix} 1 & x_{11} & \cdots & x_{1d} \\ \vdots & \vdots & \ddots & \vdots \\ 1 & x_{N1} & \cdots & x_{Nd} \end{bmatrix}, \quad Y = \begin{bmatrix} y_1 \\ y_2 \\ \vdots \\ y_N \end{bmatrix} \Rightarrow \beta = \begin{bmatrix} \beta_0 \\ \beta_1 \\ \vdots \\ \beta_d \end{bmatrix}$$

• Goal: find the coefficients $\beta$, which minimize the SSE:

$$\min_{\beta} g(\beta) = \min_{\beta} \|X\beta - Y\|_2^2 = \min_{\beta} (X\beta - Y)^T (X\beta - Y)$$

$$= \min_{\beta} (\beta^T X^T X\beta - 2Y^T X\beta + Y^T Y)$$
Linear Regression

- Set the first derivative of \( g(\beta) = \beta^T X^T X\beta - 2Y^T X\beta + Y^T Y \) to zero:
  \[
  X^T X\beta = X^T Y
  \]

- If \( X^T X \) is non-singular then:
  \[
  \beta = (X^T X)^{-1} \cdot X^T Y
  \]

- For \( d = 1 \), the regression coefficients \( \beta_0 \) and \( \beta_1 \) can be computed as:
  \[
  \beta_1 = \frac{\text{Cov}(x,y)}{\text{Var}(x)} = \frac{\bar{x}^T \bar{y}}{\bar{x}^T \bar{x}}
  \]
  and
  \[
  \beta_0 = \bar{y} - \beta_1 \cdot \bar{x}
  \]

Note that if \( \bar{x} = 0 \) \( \Rightarrow \) \( \beta_1 = \frac{x^T y}{x^T x} \) and \( \beta_0 = 0 \)
Polynomial Regression

• Second order polynomial for \( d = 1 \):
  \[
  \hat{y} = \beta_0 + \beta_1 \cdot x_1 + \beta_2 \cdot x_1^2 = x_d = [1 \ x_1 \ x_1^2] \cdot \begin{bmatrix} \beta_0 \\ \beta_1 \\ \beta_2 \end{bmatrix}
  \]

  with \( X = \begin{bmatrix} 1 & x_{11} & x_{11}^2 \\ \vdots & \vdots & \vdots \\ 1 & x_{N1} & x_{N1}^2 \end{bmatrix} \) and \( \beta = (X^T \cdot X)^{-1} \cdot X^T \cdot Y \)

• Second order polynomial for \( d = 2 \):
  \[
  \hat{y} = \beta_0 + \beta_1 \cdot x_1 + \beta_2 \cdot x_2 + \beta_3 \cdot x_1^2 + \beta_4 \cdot x_2^2 + \beta_5 \cdot x_1 \cdot x_2
  \]
Polynomial Regression

- The number of coefficients increases exponentially with $k$ and $d$
- Model building strategies: forward selection, backward elimination
- The order of the polynomial should be as low as possible, high order polynomials tend to overfit the data

**Linear model**

**Polynomial model 2$^{nd}$ order**

**Polynomial model 6$^{th}$ order**
Nonlinear Regression

• Different nonlinear functions can be approximated

• Transform the data to a linear domain

\[ \hat{y} = \alpha \cdot e^{\gamma x} \Rightarrow \ln(\hat{y}) = \ln(\alpha) + \gamma x \]

\[ \Rightarrow \hat{y}' = \beta_0 + \beta x \]

( for \( \hat{y}' = \ln(\hat{y}) \), \( \beta_0 = \ln(\alpha) \), and \( \beta_1 = \gamma \))

• The parameters \( \beta_0 \) and \( \beta \) are estimated with LS

• The parameters \( \alpha \) and \( \gamma \) are obtained, describing an exponential curve which passes through the original observations

• Problem: LS determines normally distributed errors in the transformed space \( \Rightarrow \) skewed error distribution in the original space
Nonlinear Regression

• Different nonlinear functions can be approximated
• Outputs are estimated by a function with nonlinear parameters, e.g., exponential, trigonometric
• Example type of function:
  \[ \hat{y} = \beta_0 + \beta_1 e^{\beta_2 x} + \sin(\beta_3 x) \]
• Approach: the type of nonlinear function is chosen and the corresponding parameters are computed
• No closed form solution exists \(\Rightarrow\) numerical approximations:
  – Gauss Newton, Gradient descent, Levenberg-Marquardt
Linear and Nonlinear Regression

• Problems:
  – Linear regression – most of the real world data has a nonlinear behavior
  – Polynomial regression – limited, cannot describe arbitrary nonlinear behavior
  – General nonlinear regression – the type of nonlinear function must be specified in advance
Piecewise Linear Regression

• Piecewise linear functions:

\[ f(x) = \begin{cases} 
\beta_0 + \beta_1 \cdot x_1 + \ldots + \beta_d \cdot x_d, & x \in \mathcal{P}_1 \\
\vdots \\
\beta_k \cdot x_1 + \ldots + \beta_d \cdot x_d, & x \in \mathcal{P}_k 
\end{cases} \]

• Simple approach
• Able to describe arbitrary functions
• The **accuracy** is increasing with an increasing number of partitions/linear models
• The **compactness & interpretability** is increasing with a decreasing number of partitions/linear models
• **Challenge:** find an appropriate partitioning in the input space (number and shapes)
1. Introduction of different learning techniques for piecewise linear models

2. Discussion of the bias-variance problem, regression and ensemble techniques
Chapter 7: Numerical Prediction

1) Introduction
   - Numerical Prediction problem, linear and nonlinear regression, evaluation measures

2) Piecewise Linear Numerical Prediction Models
   - Regression Trees, axis parallel splits, oblique splits
   - Hinging Hyperplane Models

3) Bias-Variance Problem
   - Regularization, Ensemble methods
Regression Trees

- Greedy divide and conquer: recursive partitioning of the input space
- Example with input $x$ and output $y$:

![Graph showing regression tree with input $x$ and output $y$.]
Regression Trees

• **Example:**

```
\[ x \leq x_4 \]
\[ x > x_4 \]
\[ x < x_2 \]
\[ x \geq x_2 \]
\[ x < x_3 \]
\[ x \geq x_3 \]
\[ x < x_5 \]
\[ x \geq x_5 \]
\[ x < x_6 \]
\[ x \geq x_6 \]
\[ x < x_7 \]
\[ x \geq x_7 \]
```
Regression Trees

• General approach of learning a regression tree:
  – Given: set of observations $T$
  – Find a split of $T$ in $T_1$ and $T_2$ with minimal summed impurity
    \[ \text{imp}(T_1) + \text{imp}(T_2) \]
  – If the stopping criterion is not reached: repeat for $T_1$ and $T_2$
  – If the stopping criterion is reached: undo the split

• Internal node denotes a test in the input space
• Branch represents an outcome of the test
• Leaf nodes contain a linear function, used to predict the output
**Impurity Measure**

- Variance of the output: \( \text{imp}(T) = \frac{1}{|T|} \sum_{(x,y) \in T} (y - \bar{y})^2 \)

- Better: variance of the residuals:

\[ \text{imp}(T) = \frac{1}{|T|} \sum_{(x,y) \in T} (y - f(x))^2 \]

if constant models are learned

if linear models are learned
Stopping Criterion: Impurity Ratio

- The recursive splitting is stopped if:
  a) The sample size of a node is below a specified threshold
  b) The split is not significant:

  - If the relative impurity ratio induced by a split is higher than a given threshold, then the split is not significant

    \[ \tau = \frac{imp(T_1) + imp(T_2)}{imp(T)} \] \[ > \tau_0 \]

  - As the tree grows the resulting piecewise linear model gets more accurate. \( \tau \) increases, becoming higher than \( \tau_0 \)

- Choosing the parameter \( \tau_0 \) ⇔ trading accuracy with overfitting
- stopping too soon ⇒ model is not accurate enough
- stopping too late ⇒ model overfits the observations
Split Strategy

- The split strategy determines how the training samples are partitioned, whether the split is actually performed is decided by the stopping criterion.

- The most common splits are axis parallel:
  - Split = a value in one input dimension
  - Compute the impurity of all possible splits in all input dimensions and choose at the end the split with the lowest impurity
  - For each possible split compute the two corresponding models and their impurity ⇒ expensive to compute

4 axis parallel splits in the 2D input space, in order to separate the red from the blue samples.
Strategy for Oblique Splits

- More intuitive to use oblique splits
- An oblique split is a linear separator in the input space instead of a split value in an input dimension
- The optimal split (with minimal impurity measure) cannot be efficiently computed
- Heuristic approach required
Strategy for Oblique Splits

- Heuristic approach:
  a) Compute a clustering in the full (input + output) space, such that the samples are as well as possible described by linear equations
  b) Project the clusters onto the input space
  c) Use the clusters to train a linear classifier in the input space. Split = separating hyperplane in input space
Strategy for Oblique Splits

• Heuristic approach:
  a) Compute a clustering in the full (input + output) space, such that the samples are as well as possible described by linear equations
  b) Project the clusters onto the input space
  c) Use the clusters to train a linear classifier in the input space. Split = separating hyperplane in input space
  d) Compute linear models for the two linearly separated clusters
Example Models

- Example piecewise linear models (with oblique splits in the input space):
Chapter 7: Numerical Prediction

1) Introduction
   - Numerical Prediction problem, linear and nonlinear regression, evaluation measures

2) Piecewise Linear Numerical Prediction Models
   - Regression Trees, axis parallel splits, oblique splits
   - Hinging Hyperplane Models

3) Bias-Variance Problem
   - Regularization, Ensemble methods
Hinging Hyperplane Models

- **Hinging Hyperplane Model (HH-model)** for continuous models

Model: \( f(x) = \sum_{i=1}^{K} h_i(x) \)

Hinge: \( \Delta = \beta^+ - \beta^- \)

Hinge function:

\[
 h(x) = \begin{cases} 
 \tilde{x}^T \beta^+, & \tilde{x}^T \cdot \Delta > 0 \\
 \tilde{x}^T \beta^-, & \tilde{x}^T \cdot \Delta \leq 0
\end{cases}
\]

with \( \tilde{x}^T = (1, x_1, \ldots, x_n) \).

[ L. Breiman (1993) ]

Numerical Prediction → Hinging Hyperplane Models
Hinging Hyperplane Models

- **Hinge Finding Algorithm (HFA)**

1) Start with a random partitioning of the input space: $\Delta_j$ ($j = 0$)

2) Determine the two corresponding partitions:

$$S_j^- = \{ \mathbf{x} \mid \mathbf{x}^T \Delta_j \leq 0 \} \text{ and } S_j^+ = \{ \mathbf{x} \mid \mathbf{x}^T \Delta_j > 0 \}$$

3) Compute the regression hyperplanes for $S_j^+$ and $S_j^-$
**Hinge Finding Algorithm (HFA)**

3) Compute the regression hyperplanes for $S_j^+$ and $S_j^-$

4) Compute the hinge $\Delta_{j+1}$ from the regression coefficients $\beta_j^-$ and $\beta_j^+$

5) Determine the new partitions $S_{j+1}^+$ and $S_{j+1}^-$ determined by $\Delta_{j+1}$

6) If $S_{j+1}^+ = S_j^+$ or $S_{j+1}^+ = S_j^-$, then stop, else return to step 3).
The Hinge Finding Algorithm (HFA) might not converge – a hinge might induce a partitioning outside the defined input space.

**Line search**: binary search to guarantee convergence (to a local minimum).

Instead of updating the hinge directly $\Delta_j \rightarrow \Delta_{j+1}$, first check the accuracy improvement brought by $\Delta_{j+1}$.

If $\Delta_{j+1}$ does not improve the model impurity, then perform a binary search after the linear combination of $\Delta_j$ and $\Delta_{j+1}$ yielding the lowest impurity.

$$
\Delta'_{j+1} = \Delta_i + \lambda (\Delta_{j+1} - \Delta_j), \quad \lambda \in \left\{ \frac{1}{2}, \frac{1}{4}, \frac{1}{8}, \frac{1}{16} \right\}
$$
Fit Multiple Hinges

• Goal: Describe the target function $y$ as a sum of $N$ hinge functions

$$y = \sum_{i=1}^{N} h_i(x)$$

• Each hinge function $h_i$ can be seen as fitted to the residual function

$$y[i] = y - \sum_{j \neq i} h_j(x),$$

since then $h_i(x) = y - \sum_{j \neq i} h_j(x)$.

• Fit multiple hinges iteratively in $N$ steps:
  
  − Start with $h_1(x) = \cdots = h_N(x) = 0$

  − Step $n$: Fit the $n$-th hinge $h_n$ to $y[n] = y - (h_1(x) + \cdots + h_{n-1}(x))$. Then repeatedly refit $h_1$ to $y[1]$, $\ldots$, and $h_{n-1}$ to $y[n-1]$ until the hinges do not change anymore.
Fit Multiple Hinges

• Example:
  – Step 1: Fit the first hinge function \( h_1 \) to \( y_{[1]} = y - 0 \).
  – Step 2: Compute the residual outputs
    \[
    y_{[2]} = y - h_1(x).
    \]
    Fit the second hinge function \( h_2 \) to \( y_{[2]} \).
    Then refit the first hinge to \( y_{[1]} = y - h_2(x) \).
  – Step 3: Compute the residual outputs
    \[
    y_{[3]} = y - h_1(x) - h_2(x).
    \]
    Fit the third hinge function \( h_3 \) to \( y_{[3]} \).
    Then repeatedly refit
    \[
    h_1 \text{ to } y_{[1]} = y - h_2(x) - h_3(x) \text{ and }
    h_2 \text{ on } y_{[2]} = y - h_1(x) - h_3(x).
    \]
    until no more changes occur.
Example Models

- Example piecewise linear models (with oblique splits in the input space):
Chapter 7: Numerical Prediction

1) Introduction
   - Numerical Prediction problem, linear and nonlinear regression, evaluation measures

2) Piecewise Linear Numerical Prediction Models
   - Regression Trees, axis parallel splits, oblique splits
   - Hinging Hyperplane Models

3) Bias-Variance Problem
   - Regularization, Ensemble methods
Underfitting vs. Overfitting

- **Underfitting** = a learned model is not flexible enough to capture the underlying trend
- **Overfitting** = a learned model is too flexible, allowing to capture illusory trends in the data, which appear due to noise

**Polynomial models**

- Underfitting
- Good fitting
- Overfitting

**Piecewise linear models**

- Underfitting
- Good fitting
- Overfitting
• Assuming that the data generation process can be repeated (with a certain amount of randomness) ⇒ obtain several datasets $D_i$ & for each $D_i$ a model $f_i$ is learned

• **Bias** = the difference between the average prediction of these models and the correct value

• **Variance** = the variability of the predictions of the different models

Image after [http://scott.fortmann-roe.com/docs/BiasVariance.html](http://scott.fortmann-roe.com/docs/BiasVariance.html)
Bias-Variance Tradeoff

- Underfitting = low variance, high bias (e.g. use mean output as estimator)
- High bias = a model does not approximate the underlying function well
- Overfitting = high variance, low bias
- When a model is too complex, small changes in the data cause the predicted value to change a lot ⇒ high variance

- **Search for the best tradeoff between bias and variance**
- Regression: control the bias-variance tradeoff by means of the polynomial order/number of coefficients
- Regression trees: control the bias-variance tradeoff by means of the tree size/number of submodels
Bias and Variance

Consider the expected prediction error of a learned model:

\[ \text{Err}(\mathbf{x}) = E[(f(\mathbf{x}) - y)^2] \]

\[ \text{Err}(\mathbf{x}) = E[f(\mathbf{x})^2 - 2f(\mathbf{x})y + y^2] \]

\[ \text{Err}(\mathbf{x}) = E[f(\mathbf{x})^2] - 2E[f(\mathbf{x})]E[y] + E[y^2] \]

\[ \text{Err}(\mathbf{x}) = E[(f(\mathbf{x}) - \overline{f(\mathbf{x})})^2] + \overline{f(\mathbf{x})}^2 - 2\overline{f(\mathbf{x})}E[y] + E[(y - E[y])^2] + E[y^2] \]

\[ \text{Err}(\mathbf{x}) = E[(f(\mathbf{x}) - \overline{f(\mathbf{x})})^2] + (\overline{f(\mathbf{x})} - E[y])^2 + E[(y - E[y])^2] \]

error due to the variance of \( f \)

error due to the bias of \( f \)

(systematic error)

irreducible error

\((\star): E[z^2] = E[(z - E[z])^2] + E[z]^2\)
Consider the expected prediction error of a learned model:

\[ Err(x) = E[(f(x) - \bar{f}(x))^2] + \left(\bar{f}(x) - E[y]\right)^2 + E[(y - E[y])^2] \]

- Error due to the variance of \( f \)
- Error due to the bias of \( f \) (systematic error)
- Irreducible error

Carefully balance between these two types of error, in order to minimize the total expected prediction error.
Regularization

- Minimizing the sum of squared errors

\[ \sum_{(x,y) \in T} (f(x) - y)^2 \rightarrow \min \]

computes an unbiased linear model with very high variance

- Idea: give up the unbiasedness and obtain a variance decrease by penalizing the model complexity

- Regularization: simultaneously minimize the sum of squared errors and the norm of the coefficient vector

- Linear regularization (ridge regression):

\[ \sum_{(x,y) \in T} (f(x) - y)^2 + \lambda \| \beta \|_2^2 \rightarrow \min \]
Regularization

- Lasso regularization:
  \[
  \arg\min_\beta \sum_{(x,y) \in T} (f(x) - y)^2, \quad s.t. \|\beta\|_1 \leq s
  \]
  - solvable with a quadratic programming algorithm
  - With an increasing penalty more and more coefficients are shrunk towards zero, generating more sparse models

- Linear regularization (ridge regression):
  \[
  \arg\min_\beta \sum_{(x,y) \in T} (f(x) - y)^2, \quad s.t. \|\beta\|_2^2 \leq s
  \]
  - solvable similar to SSE: \( \beta = (X^TX + \lambda I)^{-1} \cdot X^TY \)
  - Reduces all coefficients simultaneously
Bagging

- When discussing the bias-variance tradeoff, we assumed infinitely many replications of our data set, but in practice we have only one training set $T$
- Simulate multiple training sets $T_1, T_2, \ldots, T_k$ by constructing bootstrap replicates of the original training set $T$, by randomly drawing samples from $T$ (with replacement) such that $|T_j| = |T|, j \in \{1, \ldots, k\}$
- Learn a model $f_j$ for each replicate $T_j$ (use as test set $T_{Sj} = T \setminus T_j$)
- For each input $x$, we have several predictions $y_1, \ldots, y_k \Rightarrow$ compute the average prediction
- $f(x) \approx \overline{f(x)} \Rightarrow (f(x) - \overline{f(x)})^2 \approx 0 \Rightarrow$ the variance is removed/reduced
- Bias: $(\overline{f(x)} - E[y])^2$ is the same as before
Ensemble Methods

• Bagging:
  – use it for models with a low bias
  – If the bias is low, bagging reduces the variance, while bias remains the same
  – use it for complex models, which tend to overfit the training data
  – in practice it might happen that the bagging approach slightly increases the bias

• Boosting:
  – Can be adapted for regression models
  – Reduces the bias in the first iterations
  – Reduces the variance in later iterations