IIT KHARAGPUR

NPTEL ONLINE CERTIFICATION COURSES

Data Mining

Week 8: Regression, Dimensionality Reduction

Pabitra Mitra

Computer Science and Engineering, IIT Kharagpur

Regression

- Predict real-valued output for given input – given a training set
- Examples:
- Predict rainfall in cm for month
- Predict stock prices in next day
- Predict number of users who will click on an internet advertisement

Linear Regression

- Task: predict real-valued Y, given real-valued vector X using a regression model f
- Error function, e.g., least squares is often used
-

$$S(\underline{\theta}) = \sum_i \left[y(i) - f\left(X(i) ; \underline{\theta}\right) \right]^2$$

target value

predicted value

- Model structure: e.g., linear $f(X ; \underline{\theta}) = \alpha_0 + \sum \alpha_j x_j$
- Model parameters = $\underline{\theta} = \{\alpha_0, \alpha_1, \ldots\ldots \alpha_p\}$

Estimating θ (having least error): we can write-

$$S(\theta) = \sum_i [y(i) - \sum \alpha_j x_j]^2$$

$$= \sum_i e_i^2$$

$$= e' e$$

$$= (y - X\theta)' (y - X\theta)$$

where $e = y - X\theta$

y = N x 1 vector of target values

N x (p+1) vector of input values

(p+1) x 1 vector of parameter values

$$S(\theta) = \Sigma e^2 = e' e = (y - X\theta)' (y - X\theta)$$

$$= y' y - \theta' X' y - y' X \theta + \theta' X' X \theta$$

$$= y' y - 2\theta' X' y + \theta' X' X \theta$$

Taking derivative of $S(\theta)$ with respect to the components of θ gives -

$$dS/d\theta = -2 X' y + 2 X' X \theta$$

Set this to 0 to find the minimum of S as a function of θ.

Set to 0 to find the minimum of S as a function of θ …

$\Rightarrow$ $-2 X' y + 2 X' X \theta = 0$

$\Rightarrow$ $X' X \theta = X' y$ (known in statistics as the Normal Equations)

Letting $X' X = C$, and $X' y = b$,

we have $C \theta = b$, i.e., a set of linear equations

We could solve this directly, e.g., by matrix inversion

$$\theta = C^{-1} b = (X' X)^{-1} X' y$$

Solving for the θ's

- Problem is equivalent to inverting X' X matrix
  - Inverse does not exist if matrix is not of full rank
    - E.g., if 1 column is a linear combination of another (collinearity)
    - Note that X'X is closely related to the covariance of the X data
      - So we are in trouble if 2 or more variables are perfectly correlated
    - Numerical problems can also occur if variables are almost collinear

- Equivalent to solving a system of p linear equations
  - Many good numerical methods for doing this, e.g.,
    - Gaussian elimination, LU decomposition, etc
  - These are numerically more stable than direct inversion

- Alternative: gradient descent
  - Compute gradient and move downhill

Multivariate Linear Regression

- Prediction model is a linear function of the parameters
- Score function: quadratic in predictions and parameters
  - $\Rightarrow$ Derivative of score is linear in the parameters
  - $\Rightarrow$ Leads to a linear algebra optimization problem, i.e., $C\,\theta = b$
- Model structure is simple….
  - p-1 dimensional hyperplane in p-dimensions
  - Linear weights => interpretability
- Often useful as a baseline model
  - e.g., to compare more complex models to
- Note: even if it's the wrong model for the data (e.g., a poor fit) it can still be useful for prediction

Limitations of Linear Regression

- True relationship of X and Y might be non-linear
  - Suggests generalizations to non-linear models
- Complexity:
  - $O(N p^2 + p^3)$ - problematic for large p
- Correlation/Collinearity among the X variables
  - Can cause numerical instability (C may be ill-conditioned)
  - Problems in interpretability (identifiability)
- Includes all variables in the model…
  - But what if p=1000 and only 3 variables are actually related to Y?

Non-linear Regression

- We can generalize further to models that are nonlinear:

$$f(\underline{x}\,;\,\underline{\theta}) = \alpha_0 + \Sigma\, \alpha_k\, g_k(\beta_{k0} + \Sigma\, \beta_{kj}\, x_j)$$

where the g's are *non-linear functions.*

- In statistics this is referred to as a generalized linear regression

- Closed form (analytical) solutions are rare.

- We have a multivariate non-linear optimization problem (which may be quite difficult!)

Optimization in the Non-Linear Case

- We seek the minimum of a function in d dimensions, where d is the number of parameters (d could be large!)

- There are a multitude of heuristic search techniques
  - Steepest descent (follow the gradient)
  - Newton methods (use 2nd derivative information)
  - Conjugate gradient
  - Line search
  - Stochastic search
  - Genetic algorithms
- Two cases:
  - Convex (nice -> means a single global optimum)
  - Non-convex (multiple local optima => need multiple starts)

Other non-linear models

- Splines
  - "patch" together different low-order polynomials over different parts of the x-space
  - Works well in 1 dimension, less well in higher dimensions

- Memory-based models

  $y' = \sum w_{(x',x)}\, y,$ where y's are from the training data

  $w_{(x',x)}$ = function of distance of x from x'

- Local linear regression

  $y' = \alpha_0 + \sum \alpha_j x_j$, where the alpha's are fit at prediction time just to the (y,x) pairs that are close to x'

Overfitting

- Squared Error score (as an example: we could use other scores)

$$S(\underline{\theta}) = \sum_i \left[y(i) - f\left(\underline{x}(i) ; \underline{\theta}\right) \right]^2$$

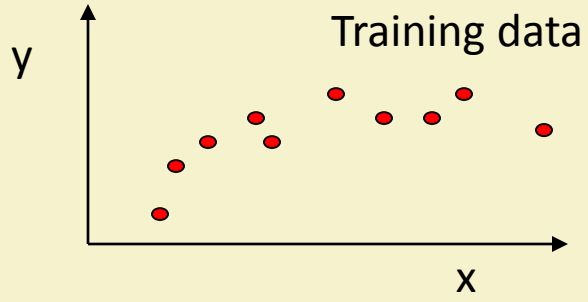
where $S(\underline{\theta})$ is defined on the training data D

- We are really interested in finding the $f(x; \underline{\theta})$ that best predicts y on **future** data, i.e., minimizing

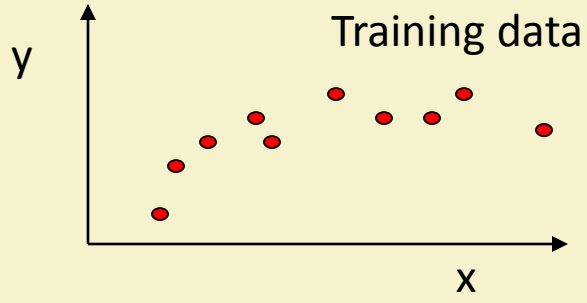
$$E[S] = E\left[y - f\left(\underline{x} ; \underline{\theta}\right) \right]^2$$ (where the expectation is over future data)

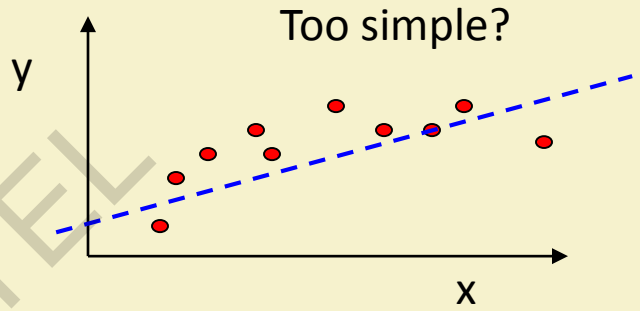
- Empirical learning
  - Minimize $S(\underline{\theta})$ on the training data D_{train}
  - If D_{train} is large and model is simple we are assuming that the best f on training data is also the best predictor f on future test data D_{test}

Complexity versus Goodness of Fit

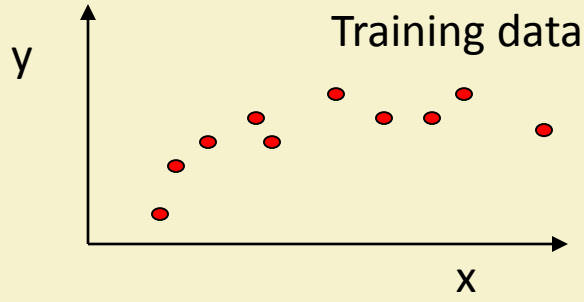


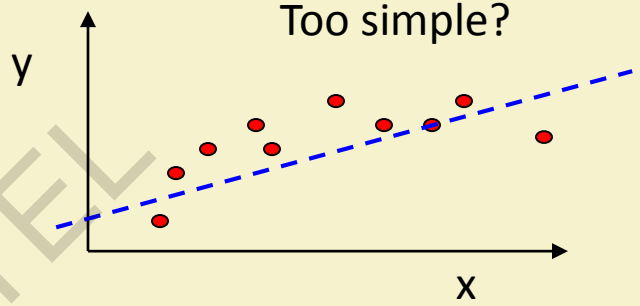
Complexity versus Goodness of Fit

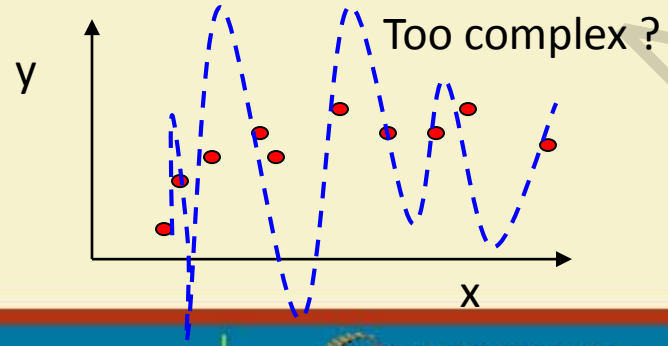




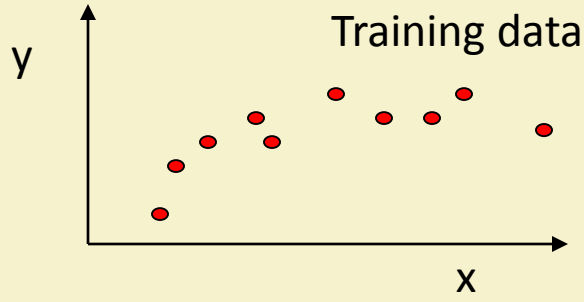
Complexity versus Goodness of Fit

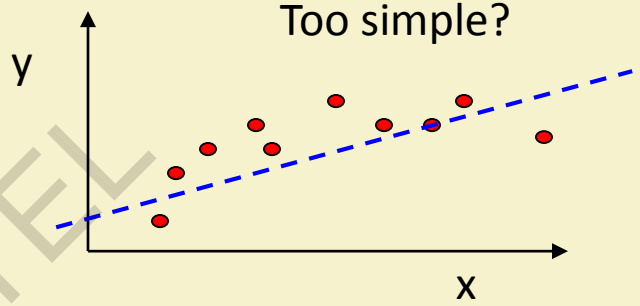


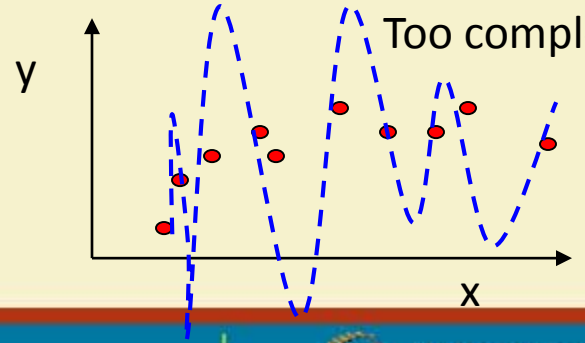


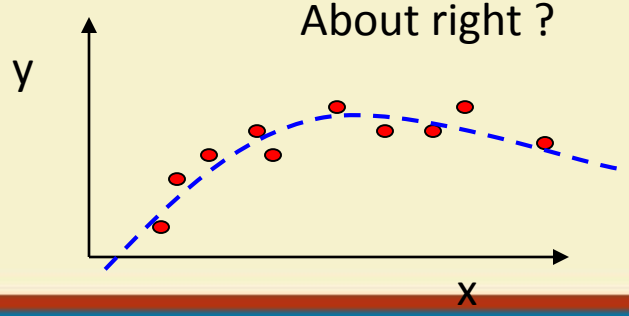


Complexity versus Goodness of Fit

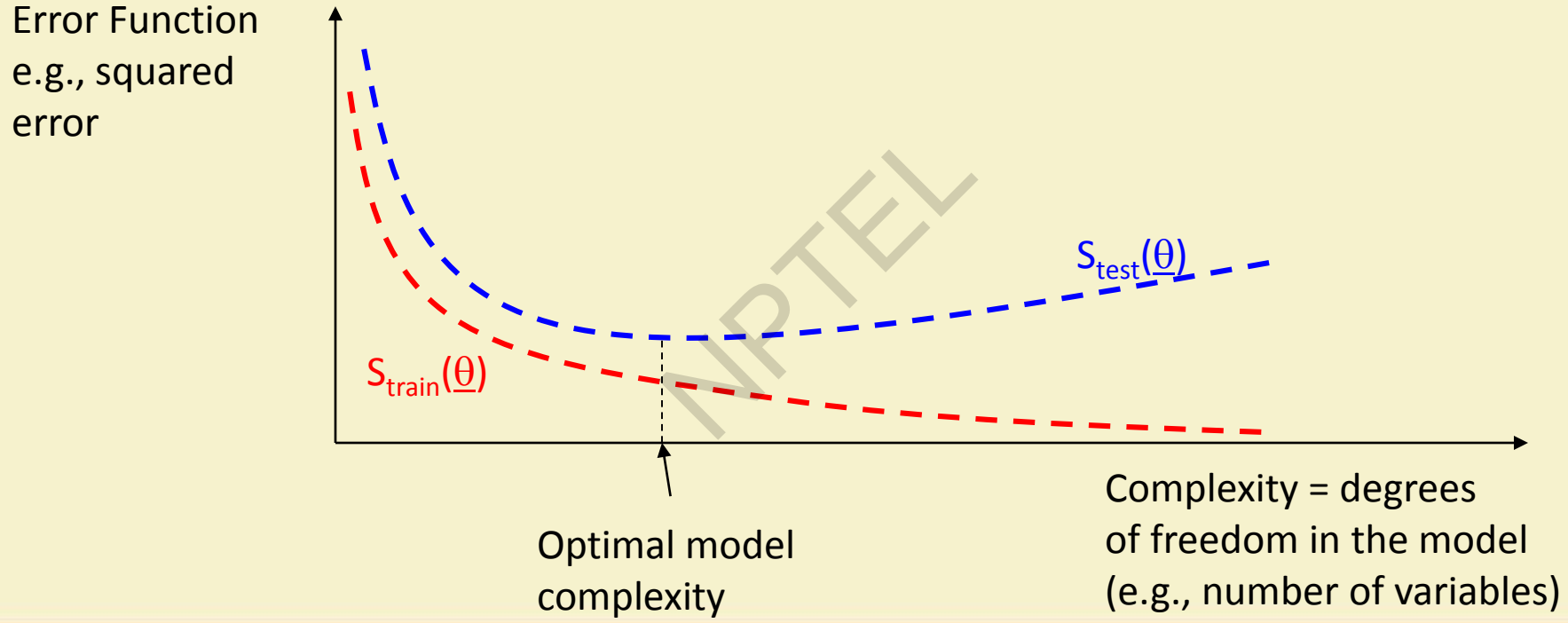






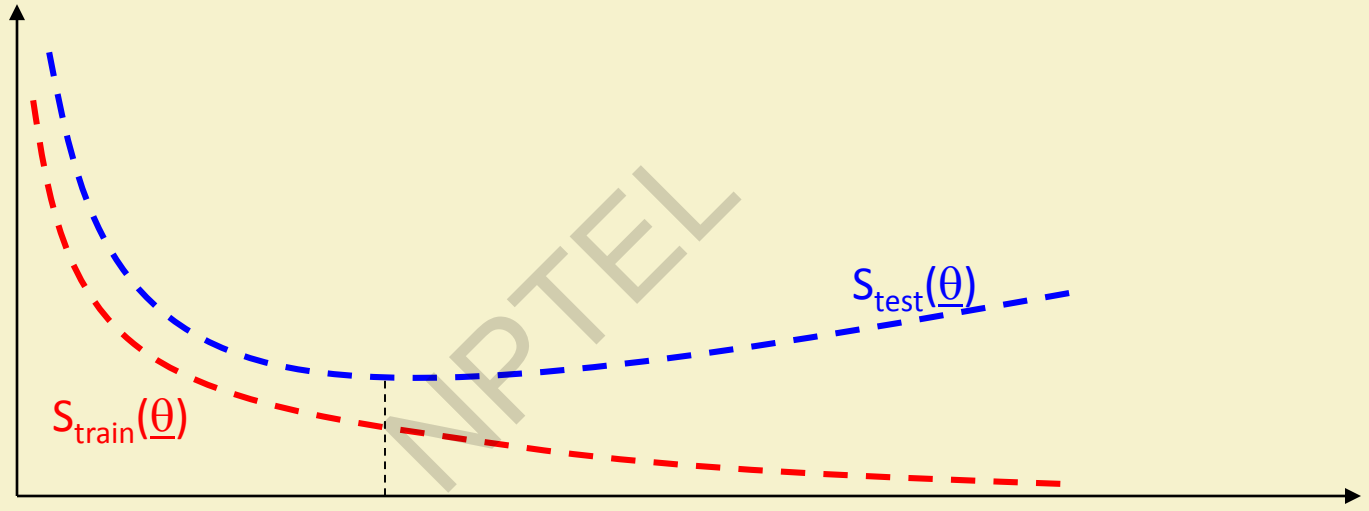


Model Complexity and Generalization



Complexity and Generalization

Score Function
e.g., squared
error



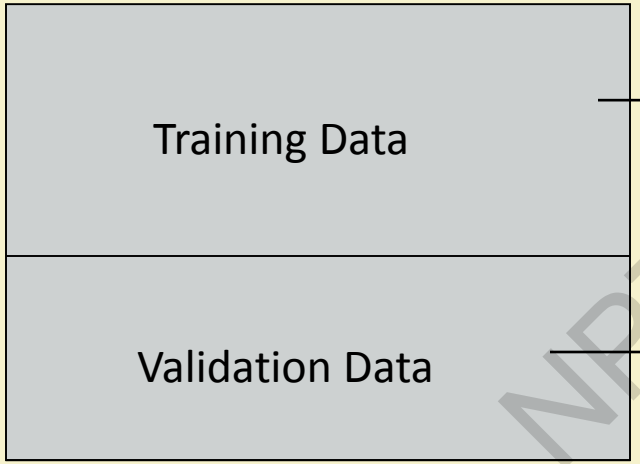
High bias
Low variance

Low bias
High variance

Training Data

Use this data to find the best $\underline{\theta}$ for each model $f_k(\underline{x} ; \underline{\theta})$

Validation Data



Use this data to find the best $\underline{\theta}$ for each model $f_k(\underline{x} ; \underline{\theta})$

Use this data to

(1) calculate an estimate of $S_k(\underline{\theta})$ for each $f_k(\underline{x} ; \underline{\theta})$ and

(2) select $k^* = \arg\min_k S_k(\underline{\theta})$

Validation Data

can generalize to **cross-validation….**

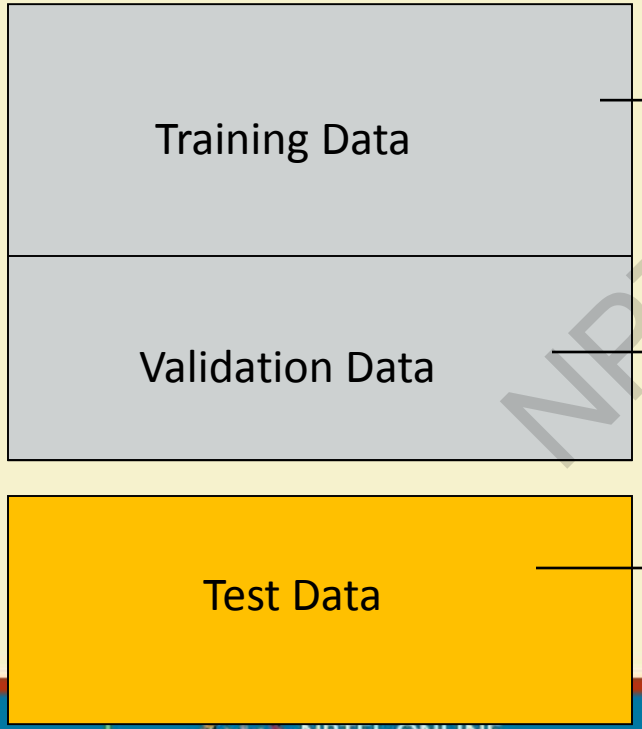
Use this data to find the best $\underline{\theta}$ for each model $f_k(\underline{x} ; \underline{\theta})$

Use this data to

(1) calculate an estimate of $S_k(\underline{\theta})$ for each $f_k(\underline{x} ; \underline{\theta})$ and

(2) select $k^* = \arg\min_k S_k(\underline{\theta})$

Test Data



Use this data to find the best $\underline{\theta}$ for each model $f_k(\underline{x} ; \underline{\theta})$

Use this data to
(1) calculate an estimate of $S_k(\underline{\theta})$ for each $f_k(\underline{x} ; \underline{\theta})$ and
(2) select $k^* = \arg\min_k S_k(\underline{\theta})$

Use this data to calculate an unbiased estimate of $S_{k^*}(\underline{\theta})$ for the selected model

Time-series prediction as regression

- Measurements over time $x_1, \ldots\ldots x_t$
- We want to predict x_{t+1} given $x_1, \ldots\ldots x_t$
- Autoregressive model

$$x_{t+1} = f(x_1, \ldots\ldots x_t ; \underline{\theta}) = \sum \alpha_k x_{t-k}$$

  - Number of coefficients K = memory of the model
  - Can take advantage of regression techniques in general to solve this problem (e.g., linear in parameters, score function = squared error, etc)

- Generalizations
  - Vector x
  - Non-linear function instead of linear
  - Add in terms for time-trend (linear, seasonal), for “jumps”, etc

Generalized Linear Models (GLMs)

- $g(y) = u(x) = \alpha_0 + \sum \alpha_j x_j$
  - Where g [ ] is a "link" function
  - u(x) is a linear function of the vector x

- Examples:
  - g = identity function -> linear regression
  - Logistic regression: $g(y) = \log(y / 1\text{-}y) = \alpha_0 + \sum \alpha_j x_j$
  - Logarithmic link: $g(y) = \log(y) = \alpha_0 + \sum \alpha_j x_j$
  - GLMs are widely used in statistics
  - Details of learning/fitting algorithm depend on the specifics of the link function

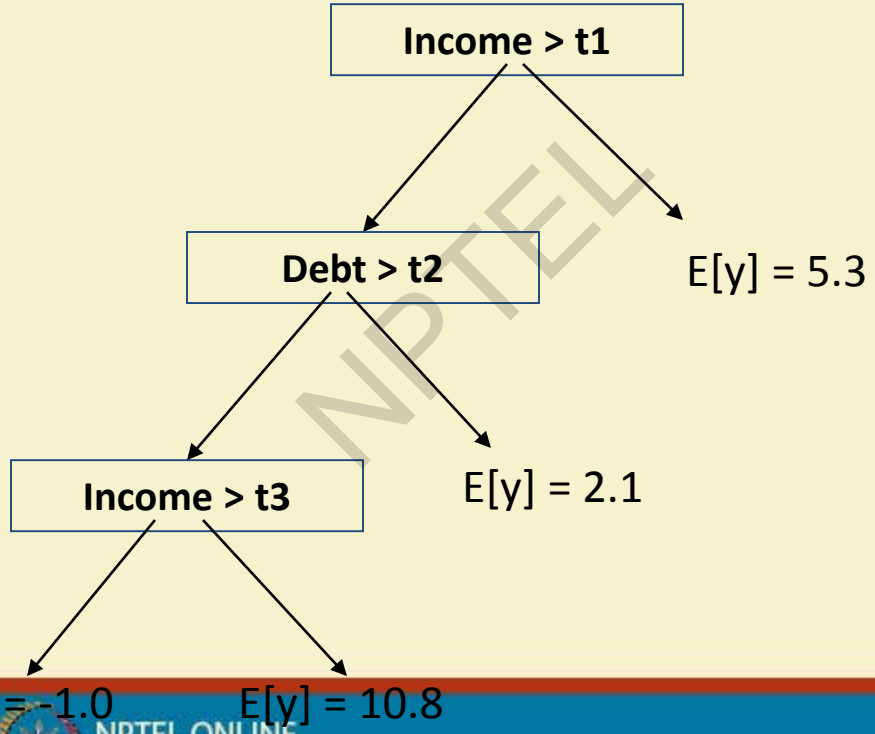
Tree-Structured Regression

- Functional form of model is a “regression tree”
  - Univariate thresholds at internal nodes
  - Constant or linear surfaces at the leaf nodes
  - Yields piecewise constant (or linear) surface
  - (like classification tree, but for regression)

- Very crude functional form…. but
  - Can be very useful in high-dimensional problems
  - Can useful for interpretation
  - Can handle combinations of real and categorical variables

- Search problem
  - Finding the optimal tree is intractable
  - Practice: greedy algorithms

Simple example of Tree Model



Greedy Search for Learning Regression Trees

- Binary_node_splitting, real-valued variables
  - For each variable x_j
    - For each possible threshold t_{jk} , compute

$$MSE(y; t_{jk}) = P(x \leq t_{jk})MSE(y|x \leq t_{jk}) + P(x > t_{jk})MSE(y|x > t_{jk})$$

MSE in left branch MSE in right branch

    - Select t_{jk} with the lowest MSE for that variable
  - Select variable x_j and t_{jk} with the lowest MSE
  - Split the training data into the 2 branches
  - For each branch
    - If leaf-node: prediction at this leaf node = mean value of y data points
    - If not: call binary_node_splitting recursively
- Time complexity?

Model Averaging/Ensembles

- Can average over parameters and models
  - E.g., weighted linear combination of predictions from multiple models

$$y = \sum w_k \, y_k$$

  - Why? Any predictions from a point estimate of parameters or a single model has only a small chance of the being the best

  - Averaging makes our predictions more stable and less sensitive to random variations in a particular data set (good for less stable models like trees)

Components of Data Mining Algorithms

- Model Representation:
  - Determining the nature and structure of the representation to be used
- Score function
  - Measuring how well different representations fit the data
- Search/Optimization method
  - An algorithm to optimize the score function
- Data Management
  - Deciding what principles of data management are required to implement the algorithms efficiently.

Steps of Data Mining Algorithm

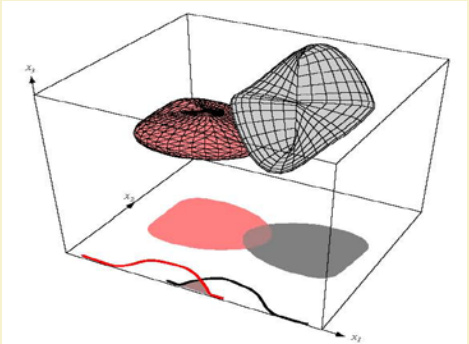
Dimensionality Reduction

- Purpose:
  - Avoid curse of dimensionality
  - Reduce amount of time and memory required by data mining algorithms
  - Allow data to be more easily visualized
  - May help to eliminate irrelevant features or reduce noise
- Techniques
  - Principle Component Analysis
  - Singular Value Decomposition
  - Others: supervised and non-linear techniques

Data Dimensionality

- From a theoretical point of view, increasing the number of features should lead to better performance.

- In practice, the inclusion of more features leads to worse performance (i.e., curse of dimensionality).



- The number of training examples required increases exponentially with dimensionality.

Dimensionality Reduction

- Significant improvements can be achieved by first mapping the data into a *lower-dimensional* space.

$$x = \begin{bmatrix} a_1 \\ a_2 \\ \dots \\ a_N \end{bmatrix} \text{ --> reduce dimensionality --> } y = \begin{bmatrix} b_1 \\ b_2 \\ \dots \\ b_K \end{bmatrix} \quad (K << N)$$

- Dimensionality can be reduced by:
  - Combining features using a linear or non-linear
  - Selecting a subset of features (i.e., feature selection).

Dimensionality Reduction (cont'd)

- Linear combinations are particularly attractive because they are simple to compute and analytically tractable.

- Given $x \in R^N$, the goal is to find an N x K matrix U such that:

$$y = U^T x \in R^K \text{ where } K << N$$

U^T

$$x = \begin{bmatrix} a_1 \\ a_2 \\ \dots \\ a_N \end{bmatrix} \text{ --> reduce dimensionality --> } y = \begin{bmatrix} b_1 \\ b_2 \\ \dots \\ b_K \end{bmatrix} \quad (K << N)$$

Dimensionality Reduction (cont'd)

- Idea: represent data in terms of basis vectors in a lower dimensional space (embedded within the original space)

(1) Higher-dimensional space representation:

$$x = a_1 v_1 + a_2 v_2 + \cdots + a_N v_N$$

$v_1, v_2, \ldots, v_N$ is a basis of the N-dimensional space

$$x = \begin{bmatrix} a_1 \\ a_2 \\ \cdots \\ a_N \end{bmatrix}$$

(2) Lower-dimensional sub-space representation:

$$\hat{x} = b_1 u_1 + b_2 u_2 + \cdots + b_K u_K$$

$u_1, u_2, \ldots, u_K$ is a basis of the K-dimensional space

$$y = \begin{bmatrix} b_1 \\ b_2 \\ \cdots \\ b_K \end{bmatrix}$$

Dimensionality Reduction (cont'd)

- Classical approaches for finding an optimal linear transformation:
  - Principal Components Analysis (PCA): Seeks a projection that preserves as much **information** in the data as possible (in a least-squares sense).
  - Linear Discriminant Analysis (LDA): Seeks a projection that best **separates** the data (in a least-squares sense).

Principal Component Analysis (PCA)

- Dimensionality reduction implies information loss; PCA preserves as much information as possible by minimizing the reconstruction error:

$$\|x - \hat{x}\|$$

$$x = a_1 v_1 + a_2 v_2 + \cdots + a_N v_N$$

$$\hat{x} = b_1 u_1 + b_2 u_2 + \cdots + b_K u_K$$

- How should we determine the "best" lower dimensional space?

The "best" low-dimensional space can be determined by the "best" eigenvectors of the covariance matrix of the data (i.e., the eigenvectors corresponding to the "largest" eigenvalues – also called "principal components").

PCA - Steps

- Suppose $x_1, x_2, ..., x_M$ are N x 1 vectors

Step 1: $\bar{x} = \frac{1}{M}\sum_{i=1}^{M} x_i$

Step 2: subtract the mean: $\Phi_i = x_i - \bar{x}$ (i.e., center at zero)

Step 3: form the matrix $A = [\Phi_1 \; \Phi_2 \; \cdots \; \Phi_M]$ ($N \times M$ matrix), then compute:

$$C = \frac{1}{M}\sum_{n=1}^{M} \Phi_n \Phi_n^T = AA^T$$

(sample **covariance** matrix, $N \times N$, characterizes the *scatter* of the data)

Step 4: compute the eigenvalues of C: $\lambda_1 > \lambda_2 > \cdots > \lambda_N$

Step 5: compute the eigenvectors of C: $u_1, u_2, \ldots, u_N$

PCA – Steps (cont'd)

- Since C is symmetric, $u_1, u_2, \ldots, u_N$ form a basis, (i.e., any vector x or actually $(x - \bar{x})$, can be written as a linear combination of the eigenvectors):

$$x - \bar{x} = b_1 u_1 + b_2 u_2 + \cdots + b_N u_N = \sum_{i=1}^{N} b_i u_i \qquad \text{where } b_i = \frac{(x - \bar{x}).u_i}{(u_i.u_i)}$$

Step 6: (**dimensionality reduction step**) keep only the terms corresponding to the K largest eigenvalues:

$$\hat{x} - \bar{x} = \sum_{i=1}^{K} b_i u_i \text{ where } K << N$$

- The representation of $\hat{x} - \bar{x}$ into the basis $u_1, u_2, \ldots, u_K$ is thus

$$\begin{bmatrix} b_1 \\ b_2 \\ \ldots \\ b_K \end{bmatrix}$$

PCA – Linear Transformation

If u_i has unit length:

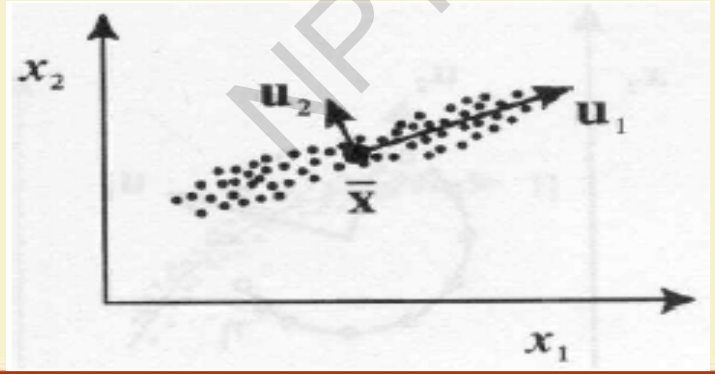
$$b_i = \frac{(x - \bar{x}).u_i}{(u_i.u_i)} = (x - \bar{x}).u_i$$

- The linear transformation $R^N \rightarrow R^K$ that performs the dimensionality reduction is:

$$\begin{bmatrix} b_1 \\ b_2 \\ \dots \\ b_K \end{bmatrix} = \begin{bmatrix} u_1^T \\ u_2^T \\ \dots \\ u_K^T \end{bmatrix} (x - \bar{x}) = U^T (x - \bar{x})$$

Geometric interpretation

- PCA projects the data along the directions where the data varies most.
- These directions are determined by the eigenvectors of the covariance matrix corresponding to the largest eigenvalues.
- The magnitude of the eigenvalues corresponds to the variance of the data along the eigenvector directions.



Error due to dimensionality reduction

- The original vector x can be reconstructed using its principal components:

$$\hat{x} - \bar{x} = \sum_{i=1}^{K} b_i u_i \text{ or } \hat{x} = \sum_{i=1}^{K} b_i u_i + \bar{x}$$

- PCA minimizes the reconstruction error:

$$e = \|x - \hat{x}\|$$

- It can be shown that the reconstruction error is:

$$e = 1/2 \sum_{i=K+1}^{N} \lambda_i$$

Feature Subset Selection

- Another way to reduce dimensionality of data
- Redundant features
  - duplicate much or all of the information contained in one or more other attributes
  - Example: purchase price of a product and the amount of sales tax paid
- Irrelevant features
  - contain no information that is useful for the data mining task at hand
  - Example: students' ID is often irrelevant to the task of predicting students' GPA

Feature Subset Selection

- Evaluate a subset of feature
- Search for the best subset

Feature Subset Selection

- Techniques:
  - Brute-force approch:
    - Try all possible feature subsets as input to data mining algorithm
  - Embedded approaches:
    - Feature selection occurs naturally as part of the data mining algorithm
  - Filter approaches:
    - Features are selected before data mining algorithm is run
  - Wrapper approaches:
    - Use the data mining algorithm as a black box to find best subset of attributes

Software

- MATLAB
  - Many free “toolboxes” on the Web for regression and prediction
  - e.g., see http://lib.stat.cmu.edu/matlab/
    and in particular the CompStats toolbox
- R
  - General purpose statistical computing environment (successor to S)
  - Free (!)
  - Widely used by statisticians, has a huge library of functions and visualization tools
- Commercial tools
  - SAS, Salford Systems, other statistical packages
  - Various data mining packages
  - Often are not progammable: offer a fixed menu of items

Useful References

T. Hastie, R. Tibshirani, and J. Friedman,
Elements of Statistical Learning, 2nd edition,
Springer Verlag, 2009