ML and Numerical Software Development

Machine Learning-II

Organon Analytics

December 10, 2019

Agenda

- A generic ML pipeline
- Choosing the algorithm
- Feature Extraction
- Pre-processing
- Managing model complexity
- Hyper-parameter optimization
- Classification&Regression Trees: The Algorithm

A generic ML pipeline

1. Collect Data
2. Create unique identifier(s)
3. Extract features from raw data
4. Check&correct data quality issues
5. Integrate data from multiple files(tables)
6. Pre-processing
7. Sample data to generate development dataset(s)
8. Train, validate, and test model
9. Prepare model documentation and save the model
10. Score the model on new dataset
11. Monitor model performance and re-build it if necessary

A generic ML pipeline

Collect Data

- Not necessary if it is already collected
- Collect everything that could be collected
- Collect the data in its rawest form. No summarization!

Create unique identifiers

- Not necessary if unique identifiers exist
- Use Statistical Record Linkage methodology for creating ids

Extract features from raw data

- Derive aggregated features from raw data
- Build models to select important features

A generic ML pipeline

Check and correct data quality issues

- Analyse distribution stabilities over time
- Delete unstable variables

Integrate data from multiple files(tables)

- Create ETL flows to integrate data under relevant entities

Pre-processing

- Handling missing values
- Handling outliers in scale variables
- Handling categorical variables
- Handling non-linearities between inputs and output
- Standardizing scale data

A generic ML pipeline

Sample data to generate development dataset(s)

- Do temporal sampling to best reflect future distribution
- Do stratified sampling to reduce the data in case of big data

Train, validate, and test model

- Choose the algorithms you will use depending on prior constraints
- Build models with optimal complexity for each algorithm
- If the interpretability is a MUST choose the model with the best performance among interpretable models
- If the interpretability is not needed, build an ensemble model out of models built with different algorithms, and different hyper-parameters
- Use a separate test sample to estimate generalization error

A generic ML pipeline

Prepare model documentation and save the model

- Document information about development samples
- Document information abut the model
- Save the model for future use

Score the model on new dataset

- Create ETL flows to create scoring database
- Implement batch scoring and on-demand scoring functionalities

Monitor&Re-build

- Data distributions and dependencies will change over time
- Monitor variable distributions, change in model parameters, change in overall model performance
- Re-build the model if changes are significant

Model Building: Choosing the algorithm

(Assuming pre-processing and feature extraction steps done)
If the model **MUST** be interpretable:

- Generalized Additive Modelling(GAM) trained with a variable selection strategy
- GAM trained with L1-regularization (Lasso)

If the model **DOES NOT NEED** to be interpretable:

- Gradient Boosting Machines(GBM) with hyper-parameter optimization
- Artificial Neural Networks(ANN) with hyper-parameter optimization
- Ensemble models where each individual model is either a GBM or an ANN

Model Building: Choosing the algorithm

Feature	GLM	GAM	CRT	GBM	ANN
Ability to handle mixed data types	🔴	🔴	🟢	🟢	🔴
Ability to handle missing values	🔴	🔴	🟢	🟢	🔴
Robustness to outliers in inputs	🔴	🟢	🟢	🟢	🔴
Ability to handle non-linear relationships	🔴	🟢	🟢	🟢	🟢
Ability to select relevant input(s)	🔴	🔴	🟢	🟢	🔴
Computational complexity with N	🟢	🟢	🟢	🟢	🔴
Interpretability	🟢	🟢	🟡	🔴	🔴
Predictive power	🔴	🟡	🔴	🟢	🟢

🔴: Poor, 🟡: Fair, 🟢: Good

Feature Extraction

Type-1: Variables are already in aggregated form. The goal is to create multivariate transformations of these aggregated variables to improve model performance. Examples:

- Create interaction variables of the polynomial forms $X_i^{p_i} X_j^{p_j}$
- Use ratios of the variables, e.g. X_i/X_j as a new variable
- Use a different basis: Spline basis, radial basis, Fourier basis, etc.

Type-2: Create variables from so called *transactional data* that has a logical view in the tabular form as follows:

Entity-id	Date	Category	Quantity
1	Nov-11-2019	Retail	129.3
1	Nov-18-2019	Pharmacy	389.3
1	Dec-01-2019	Tourism	1500
⋮	⋮	⋮	⋮
138	Dec-03-2019	E-commerce	753.3

Feature Extraction from Transactional Data

- 4 columns to process: Entity-id, Date, Category, Quantity
- Per entity, per-category a time-series exists
- Run mathematical aggregations on these time series: AVG, MAX, MIN, SUM, Time since, Time last, ratio variables
- The aggregations could be done on various time scales: Last month, last 3 months, last 12 months, etc.
- Parameter space to search for a single quantity:
  1. Operations: MAX, MIN, AVG, etc.
  2. Time-window: last week, last 3 weeks, last 12 weeks, etc
  3. Category set: Any combination of categories
- Resulting features:

OP_i in QUANTITY_j on windows $\{W_1, \cdots W_k\}$ in categories $\{C_{j_1}, \cdots C_{j_m}\}$

Feature Extraction from Transactional Data

- Examples:
  - The average of quantity-1 over the last one month in Retail AND Tourism categories
  - The days since the last quantity-1 transaction in E-Commerce and Pharmacy categories
- Cardinality of possible features per quantity-column: $N_{Op} \cdot N_{Time} \cdot 2^{N_{Category}}$

$$\begin{aligned} N_{Op} &: \# \text{ of mathematical operations} \\ N_{Time} &: \# \text{ of time windows} \\ N_{Category} &: \# \text{ of category values} \end{aligned}$$

- It is practically infinite for real-life datasets
- Use genetic algorithms to generate and select best features

Pre-processing

Pre-processing (for structured data) is needed for

- Handling missing values
- Handling outliers in scale variables
- Handling categorical variables
- Handling non-linearities between inputs and output
- Standardizing scale data

Pre-processing: Handling missing values (categorical)

1. If a single value represents all missing values do nothing.
2. If multiple missing value categories exist:
   a) Collapse sparse categories under a single category. Leave the dense categories intact
   b) Collapse all categories under a single missing-values category
3. (Advanced): Build another model to predict the missing value

Pre-processing: Handling missing values (scale)

1. Replace the missing value with the average of the (non-missing) values
2. Separate the variable into two variables: 1) An indicator variable for the missing values 2) Another variable for the non-missing values
3. (Advanced): Build another model to predict the missing value

Pre-processing: Handling outliers in scale variables

Outliers in the input:

1. Extract the distribution of missing values. Merge them under a single category if they do not corresponds to different reasons
2. If the density of missing values is very high(more than 95%), question the data source and validity of the variable: You might skip it for further analysis
3. Replace the missing value:
   - *i* With central value: Replace with the central value (sample mean of non-missing values)
   - *ii* By imputation(Advanced): Fit a model by using the variable as output
   - *iii* Weight-of-evidence variable: Use $E[Y|X = x]$ instead of X. You can compute $E[Y|X = x]$ by any univariate smoother
   - *iv* Create two variables: a) A scale variable of non-missing values b) A scale variable with two values (1 if missing; 0 if not-missing)

Pre-processing: Handling outliers in scale variables

Outliers in the output: After handling the outliers in the inputs, run a multivariate regression. Use Cook's distance to detect highly influential observations. Delete these observations or clip them to a constant value

Pre-processing: Handling categorical variables

1. Extract the histogram
2. Cardinality:
   (i) Low: If sparse categories exist, merge them. If not, do not do anything. If used for regression purposes convert it to other variables using One-hot-encoding
   (ii) High(More than 10 categories): Most problematic from a modelling point-of-view. Merge sparse categories under a common category. After that, use one-hot-encoding or weight-of-evidence transformations
3. Watch for multiple missing values: Merge them if they do not represent different reasons
4. Note-1: Merging for ordinal data should only be done for consecutive categories
5. Note-2: If an output variable Y exists, a supervised-merging of the categories is possible (Optimal binning)

Pre-processing: Handling categorical variables

One-hot-encoding: Create N indicator variables from an N-category variable:

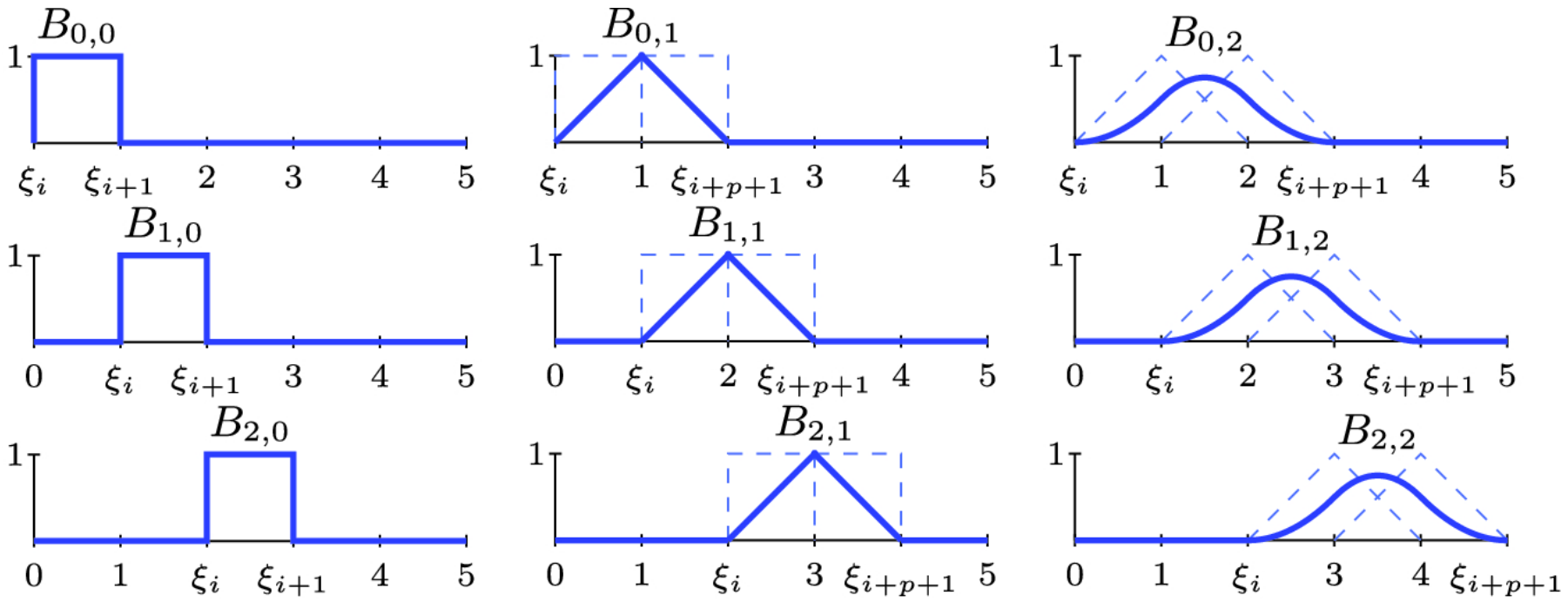
GENDER	GENDER_M	GENDER_F	GENDER_U
M	1	0	0
F	0	1	0
F	0	1	0
U	0	0	1
⋮	⋮	⋮	⋮
M	1	0	0

Pre-processing: Handling non-linearities

If the algorithm does not handle non-linearities in its specification, then use a basis expansion of the original variables to extract non-linear relationships between the inputs and the output

- Instead of using X as the input, use a basis expansion of it $\{\phi_j(X), j = 1, 2, \cdots, k\}$ as new inputs
- Various good bases exist: Polynomial bases(Hermite, Legendre, etc.), Fourier basis, wavelet basis, spline basis, etc.
- B-Spline basis works well due to its good local and smoothness properties. Learn them well. Use them exclusively

Pre-processing: B-splines expansion



(a) $p = 0$ (b) $p = 1$ (c) $p = 2$

Pre-processing: Standardizing scale data

- Needed when there are constraints on the norms of the parameters, e.g. L_1, L_2 regularization
- Needed before ANN training
- Map to unit interval:

$$X^{\text{norm}} = \frac{X - X_{\text{min}}}{X_{\text{max}} - X_{\text{min}}}$$

- Map to zero-mean&unit variance:

$$X^{\text{norm}} = \frac{X - \bar{X}}{s_X}$$

- Not needed if a basis expansion has already been done

Managing model complexity

Given a functional specification, a model get more complex IF

1. It uses more variables in the dataset
2. The magnitudes of the parameters get larger
3. The number of the parameters (degrees of freedom) gets larger
4. (In general) The magnitudes of the function and its derivatives gets larger

   Generalized Additive Modelling:

   - Number of variables
   - Number of basis functions
   - The magnitude of model parameters
   - The smoothness and shape of the bivariate relationships

   Classification&Regression Trees

   - Number of terminal nodes
   - Number of samples in each terminal node
   - Magnitude of prediction in each node

Managing model complexity

Gradient Boosting Machines

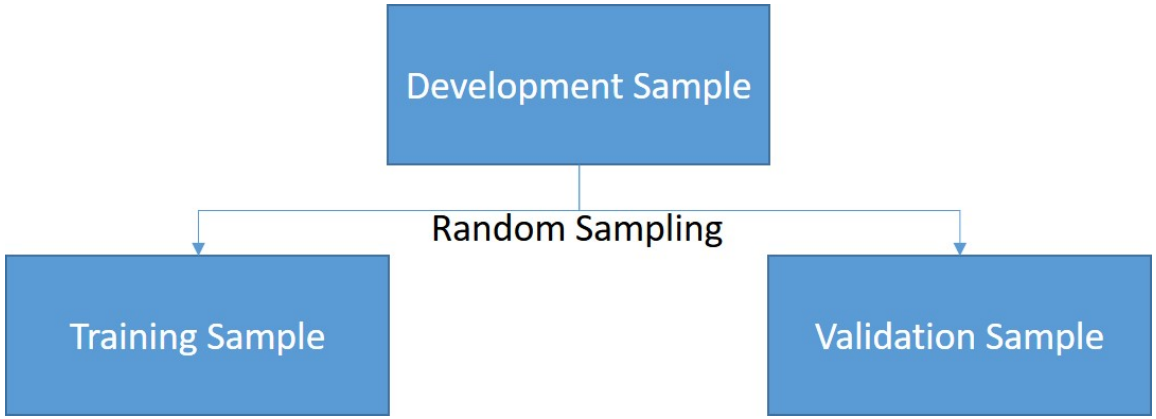
- Number of terminal nodes
- Number of samples in each terminal node
- Number of iterations

Artificial Neural Networks

- Number of variables
- Number of layers
- Number of nodes in each layer
- Magnitudes of parameters

Managing model complexity

Use cross-validation:



1. (Random) Sample the data into Training and Validation samples
2. Build models of increasing complexity, record the Training and Validation losses

$$
\begin{aligned}
M_1 &\subset M_2 \subset \cdots \subset M_k \text{ Model complexity} \\
T1 &\leq T_2 \leq \cdots \leq T_k \text{ Training loss} \\
V_1 &, V_2, \cdots V_k \text{ Validation loss}
\end{aligned}
$$

3. Choose the model with the smallest validation loss

Managing model complexity: Bias-Variance Trade-Off

Remember the data generating process

$$Y(x) = F(X) + \epsilon$$

We want to approximate $F(X)$ as close as possible from a finite samples of data $\mathcal{D}$
The solution is a function of

1. The specific set of samples: $\mathcal{D}$
2. The complexity of the function: $\mathcal{C}$

Hence, denote the solution by $g_{\mathcal{D}}^{\mathcal{C}}$
Now, keep the complexity fixes, and change the samples:

$$\mathcal{D}_1, \mathcal{D}_1, \ldots \mathcal{D}_M, \text{ Datasets}$$
$$g_{\mathcal{D}_1}^{\mathcal{C}}, g_{\mathcal{D}_2}^{\mathcal{C}}, \ldots, g_{\mathcal{D}_M}^{\mathcal{C}}, \text{ Solutions}$$
$$\mathcal{L}_{\mathcal{D}_1}^{\mathcal{C}}, \mathcal{L}_{\mathcal{D}_2}^{\mathcal{C}}, \ldots, \mathcal{L}_{\mathcal{D}_M}^{\mathcal{C}}, \text{ Losses}$$

Managing model complexity: Bias-Variance Trade-Off

For least square loss, the expected loss over data (generalization error) could be decomposed as

$$\begin{aligned} g^{\mathcal{C}} &\equiv E_{\mathcal{D}}[g_{\mathcal{D}}^{\mathcal{C}}] \\ E_{\mathcal{D}}[\mathcal{L}_{\mathcal{D}}^{\mathcal{C}}] &= (F - g^{\mathcal{C}})^2 + E_{\mathcal{D}}[(g_{\mathcal{D}}^{\mathcal{C}} - g^{\mathcal{C}})^2] \\ E_{\mathcal{D}}[\mathcal{L}_{\mathcal{D}}^{\mathcal{C}}] &= (\text{bias})^2 + \text{variance} \end{aligned}$$

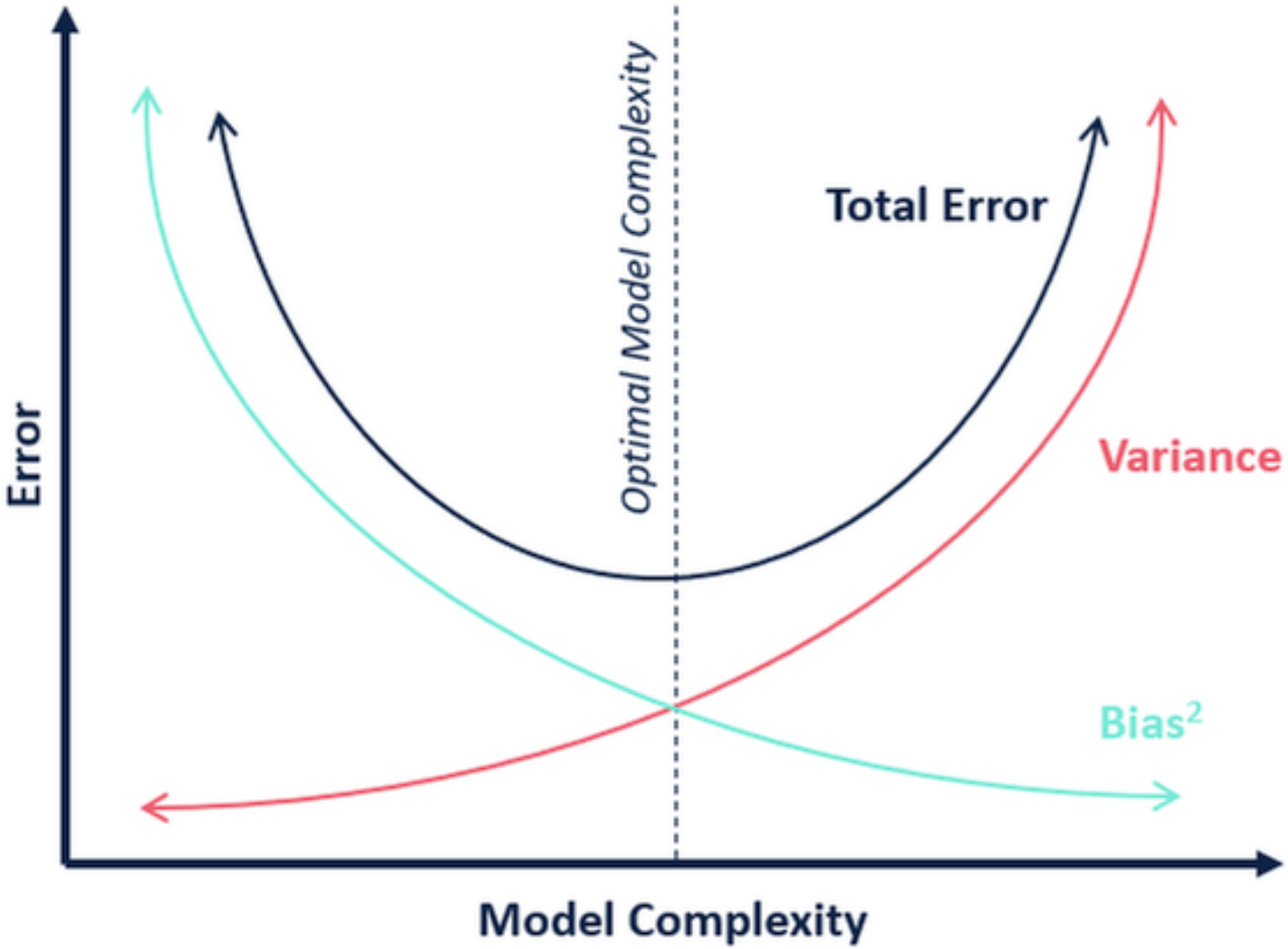
where

$$\begin{aligned} (\text{bias})^2 &= \text{distance between the truth and the average of the solution} \\ \text{variance} &= \text{volatility of the solution between the samples} \end{aligned}$$

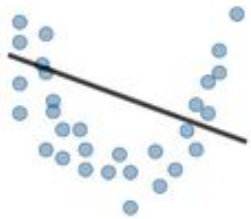
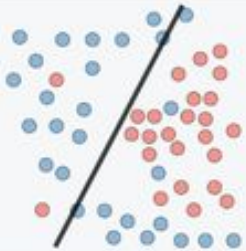
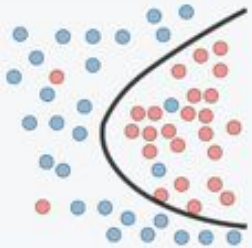
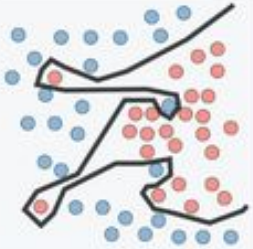
Notes:

1. As the model gets more complex, bias decreases and variance increases
2. As the model gets less complex, bias increases and variance decreases
3. Optimal model complexity must be searched for

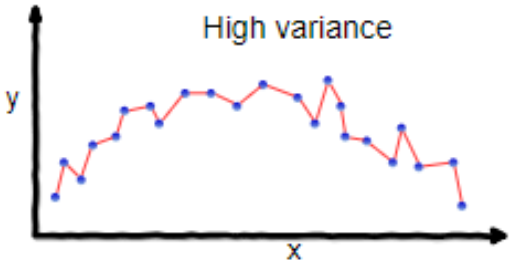
Managing model complexity: Bias-Variance Trade-Off



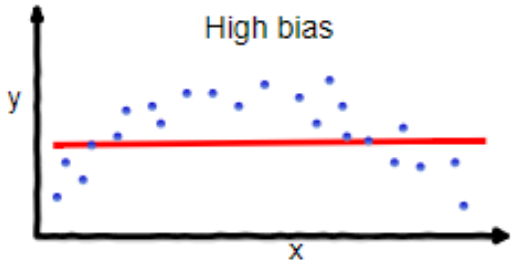
Managing model complexity: Bias-Variance Trade-Off

	Underfitting	Just right	Overfitting
Symptoms	• High training error • Training error close to test error • High bias	• Training error slightly lower than test error	• Very low training error • Training error much lower than test error • High variance
Regression illustration			
Classification illustration			
Deep learning illustration			
Possible remedies	• Complexify model • Add more features • Train longer		• Perform regularization • Get more data

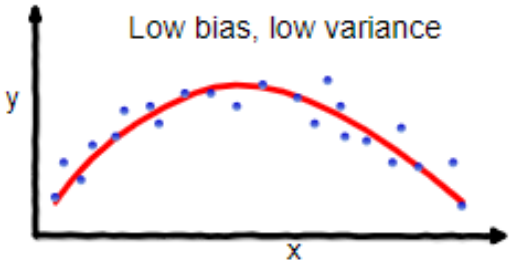
Managing model complexity: Bias-Variance Trade-Off



overfitting



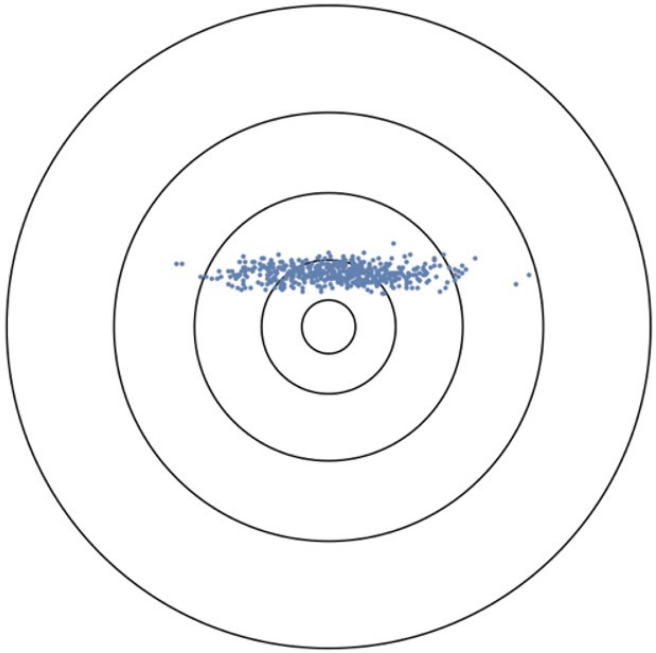
underfitting



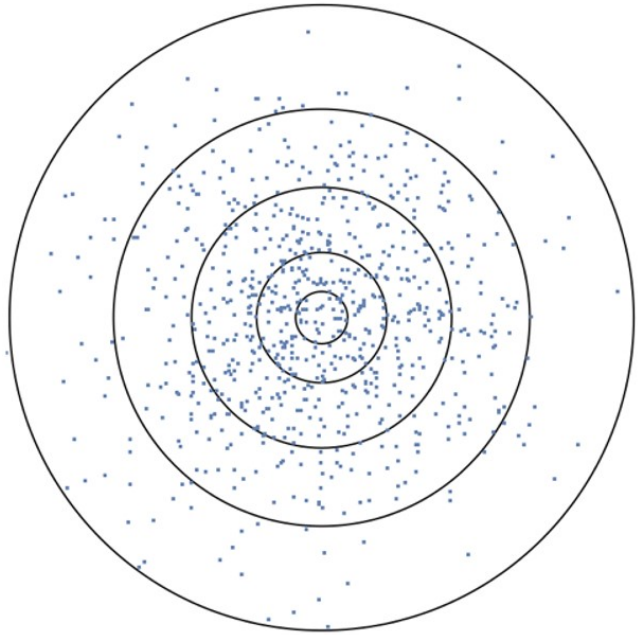
Good balance

Managing model complexity: Bias-Variance Trade-Off

The truth is at the center; The dots are its estimates



Non-zero bias BUT low-variance ➔ LOW error



Zero bias BUT **high**-variance ➔ **HIGH** error

Hyper-parameter optimization

1. **Model parameters**: θ
   - They appear in the model formula
   - Their optimal values are learned from training data
2. **Hyper parameters**: γ
   - They DO NOT appear in the model formula
   - Their optimal values are learned from validation data
   - They reflect information a) About *a priori* constraints on model complexity b) Arise as par of the learning algorithm (e.g. learning rate in Gradient Descent)

Hint: If the parameter to be learned does not exist in the model formula, it is a hyper-parameter.

Hyper-parameters: Examples

Generalized Additive Modelling

- Number of knots in each spline
- L_2 regularization coefficient if L_2 regularization is used

$$\mathcal{L}(\theta, \lambda) \equiv -\sum_i \log p(Y_i|X_i;\theta) + \lambda \sum_j \theta_j^2$$

- L_1 regularization coefficient if L_1 regularization is used

$$\mathcal{L}(\theta, \lambda) \equiv -\sum_i \log p(Y_i|X_i;\theta) + \lambda \sum_j |\theta_j|$$

- Learning rate λ, if SGD is used to train the model
- Number of variables in the model if a variable selection strategy is used

Hyper-parameters: Examples

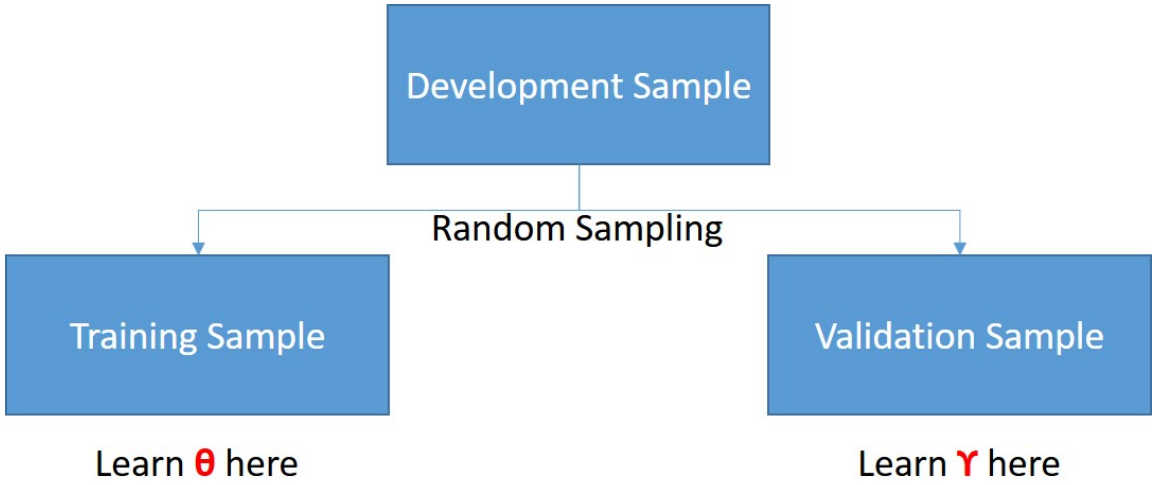
Artificial Neural Networks

- Number of layers in the network
- Number of nodes in each layer
- Activation function in each layer
- Regularization parameters if used
- Learning rate (as a constant or as a function of learning epochs)

Gradient Boosting Machines

- The depth of the CRT tree
- The number of iterations
- Learning rate (as a constant or as a function of learning epochs)
- Sampling rate for columns
- Sampling rate for rows

Hyper-parameter optimization



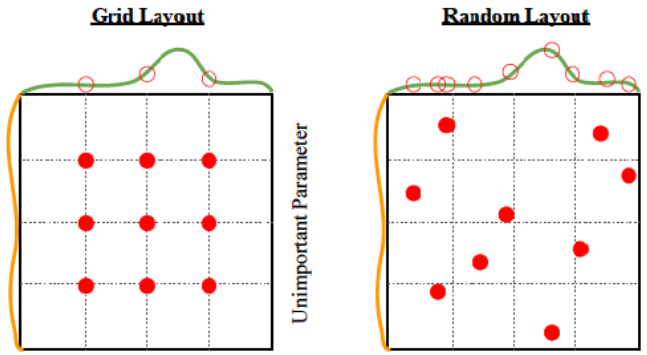
Pseudo-code for hyper-parameter optimization:

1. Generate a set of hyper-parameters $\{\gamma_i, i = 1, 2, \cdots, m\}$ (Note: $\{\gamma_i\}$ is a vector)
2. For each hyper-parameter γ_i, train the model, find optimal θ_i, and record the loss on validation set
3. The optimal hyper-parameter is the one with the minimum loss on the validation set

Hyper-parameter optimization

Strategies for generating hyper-parameters

1. Grid search: Generate $\{\gamma_i, i = 1, 2, \cdots, m\}$ on a cartesian grid
2. Random search: Generate $\{\gamma_i, i = 1, 2, \cdots, m\}$ randomly
3. Sequential model-based search:
   - *a* Start with a random set $\{\gamma_i\}$
   - *b* Find loss functions $\mathcal{L}_i$ for each γ_i
   - *c* Build a meta-model between the loss $\mathcal{L}$ and γ
   - *d* Narrow the search space to a grid where meta-model has low values
   - *e* Generate a new random set $\{\gamma_i\}$ in the narrowed grid, and go back to Step-*b*
   - *f* Iterate till convergence

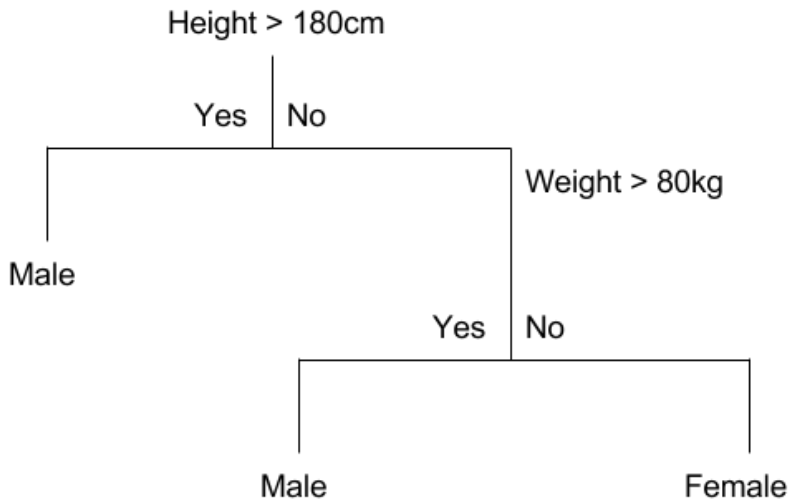


Classification and Regression Trees

A recursive (and cartesian) partitioning of input space
Define the loss function

$$\mathcal{L} \equiv \sum_i \beta_i I_{A_i}(x)$$

where $I_{A_i}(x) = 1$ if $x \in A_i, 0$ otherwise. An optimal solution is computationally intractable (NP-Hard actually)
CRT is a greedy heuristics:



Classification and Regression Trees

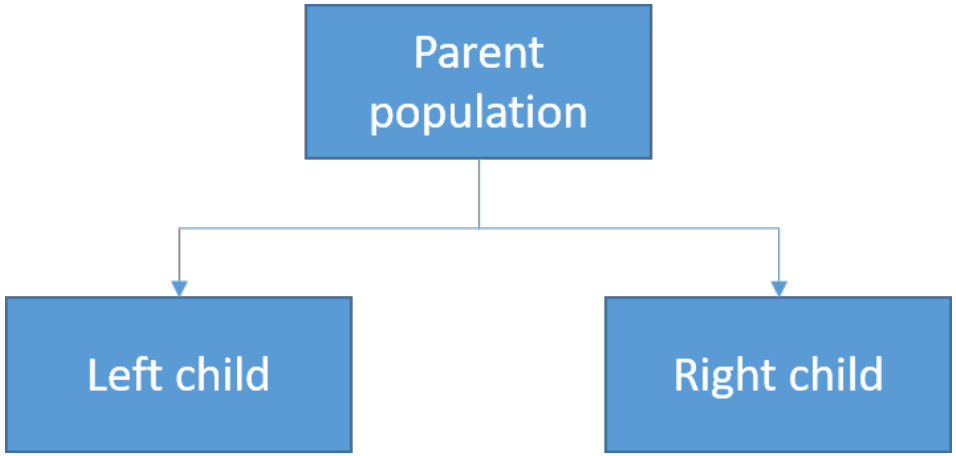
Each A_i is a product of boolean propositions:

$$A_i = P_1 \wedge P_1 \wedge P_{m_i}$$

where P_i is a boolean proposition of the following 2 forms:

1. Categorical variables: $X_k \in \{C_{j_1}, C_{j_2}, \cdots, C_{j_k}\}$, e.g Gender $\in \{M, U\}$
2. Ordinal/Scale variables: $X_k \in (a, b)$, where interval boundaries can be open or closed

Classification and Regression Trees: The Algorithm



At a given node of samples:

1. Find the best split for each input
2. Find the node's best split: Among the best splits found in Step-1, choose the one that gives the minimum loss
3. Split the node using its best split found in Step-2 if the termination rules are not satisfied

Let's define splitting criteria and termination rules $\Longrightarrow$

CRT: Splitting for scale output

Let $\{p_L, p_R\}$ represent the ratios of sample going to the left and right. Note $(p_L + p_R) = 1$

$$
\begin{aligned}
\mathcal{L}_{Parent} &= \sum_i (Y_i - \bar{Y}_{Parent})^2 \\
\mathcal{L}_{Left} &= \sum_{i \in Left} (Y_i - \bar{Y}_{Left})^2 \\
\mathcal{L}_{Right} &= \sum_{i \in Right} (Y_i - \bar{Y}_{Right})^2 \\
\Delta\mathcal{L} &\equiv \mathcal{L}_{Parent} - (p_L \cdot \mathcal{L}_{Left} + p_R \cdot \mathcal{L}_{Left})
\end{aligned}
$$

CRT: Splitting for categorical output

Use one of the following homogenity criterion at each node

1. **Gini Loss**: 1 - $\sum_i p_i(1 - p_i)$
2. **Entropy Loss**: -$\sum_i p_i \log(p_i)$

where p_i is the percentage of samples in output category C_i

In both cases, delta-loss is defined as

$$\Delta\mathcal{L} \equiv \mathcal{L}_{Parent} - (p_L\mathcal{L}_{Left} + p_R\mathcal{L}_{Left})$$

CRT: Splitting for scale output

Find the best split for the input X_j
If X_j is ordinal or scale:

1. Sort the input variable X_j
2. Starting from the minimum value, calculate $\Delta\mathcal{L}$ for each sample $i = 1, 2, \cdots, N$. That give you N values.
3. Pick the point X_j^{Best} where $\Delta\mathcal{L}$ is maximum
4. Left-child rule: $X_j <= X_j^{Best}$. Right-child rule: $X_j > X_j^{Best}$

If X_j is categorical:

1. Sort the values of X_j in ascending order of $E[Y|X_j = C_{k_j}]$
2. Starting from the category that gives the minimum value, calculate $\Delta\mathcal{L}$ for each split-point $k_j = 1, 2, \cdots, M_j$. That give you M values.
3. Pick the split point where $\Delta\mathcal{L}$ is maximum
4. Left rule: $X_j \in \{C_{j_1}, \cdots, C_{j_{Left}}\}$. Right rule: $X_j \in \{C_{j_1}, \cdots, C_{j_{Right}}\}$.

CRT: Termination rules

1. Minimum loss improvement: $\Delta\mathcal{L}_{Training} \geq \Delta\mathcal{L}_{min}$
2. Maximum tree-depth is reached
3. Maximum number of nodes reached
4. Sample size per node drops below minimum sample per node
4. $\Delta\mathcal{L}_{Validation} > \epsilon$ is not satisfied

CRT: A Summary

- A weak learner. Almost never used alone as a final model
- It does not need any pre-processing other than specifying missing values for each scale variable
- Good for an initial exploratory data analysis, and understanding important variables
- Could be used for supervised binning of an input variable in pre-processing stage
- It is used as the go-to base learner algorithm for the following meta-learning algorithms:
  - *i* Random Forest: Simple averaging of multiple trees
  - *ii* Gradient Boosting Machines: Uses boosting to average multiple trees