Practical considerations

Hung-Hsuan Chen

One-hot encoding

- Consider one-hot encoding categorical data (depending on the learning algorithm)
  - E.g., the values of the ”nationality” feature: “UK”, “Japan”, “Mexico”
  - We may encode UK as “1,0,0”, Japan as “0,1,0”, and “Mexico” as “0,0,1”
  - “Nationality feature” becomes three features: “UK or not”, “Japan or not”, “Mexico or not”

What if we encode categorical features by natural numbers?

- If we use the logistic regression model and naively encode the nationality feature (feature x_i) by numbers (e.g., "UK", "Japan", "Mexico" as 0, 1, 2)

$$y = f(w_0 + w_1 x) = 1/(1 + \exp(-(w_0 + w_1 x)))$$

- The three possible values y's are
  - $f(w_0)$ (UK)
  - $f(w_0 + w_1)$ (Japan)
  - $f(w_0 + 2w_1)$ (Mexico)
- If $w_1 > 0$, $f(w_0) < f(w_0 + w_1) < f(w_0 + 2w_1)$
- If $w_1 < 0$, $f(w_0) > f(w_0 + w_1) > f(w_0 + 2w_1)$

- The distance between (UK, Mexico) is always larger than (UK, Japan)
- It is impossible to group (UK, Mexico) in one class, and (Japan) in the other

Frequency encoding

- Encode each categorical feature by its appearance frequency
  - Useful when frequency is related with the target
  - E.g., predict salary of an employee based on rank
    - A company has few high-ranked positions
    - A company has many low-ranked positions
    - Salary might be inversely correlated with number of such positions in a company

Combining categorical features

Feature 1	Feature 2	Generated feature
A	Y	AY
B	N	BN
A	N	AN
C	Y	CY

AY	AN	BY	BN	CY	CN
1	0	0	0	0	0
0	0	0	1	0	0
0	1	0	0	0	0
0	0	0	0	1	0

Transforming numerical features

- Min-max scaling: linearly scale each feature to the range [-1,1] or [0,1]
  - $\frac{x_{ij} - \min\left(x_{*j}\right)}{\max(x_{*j}) - \min\left(x_{*j}\right)}$
- Standardize: scale each feature to N(0,1)
  - $\frac{x_{ij} - \text{mean}\left(x_{*j}\right)}{\text{std}(x_{*j})}$
- Robust scaling: removing outliers and perform scaling/ standardizing
  - $\frac{x_{ij} - \text{Q1}\left(x_{*j}\right)}{\text{Q3}\left(x_{*j}\right) - \text{Q1}(x_{*j})}$, if x_{ij} is outside the range, the sample is ignored
- Applying log or exponential function

Why scaling or standardize?

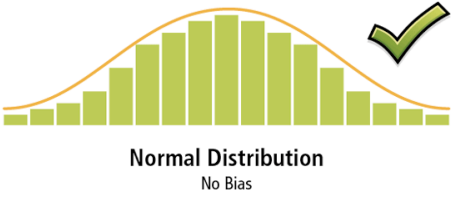
- For KNN, scaling prevents the distance scores being dominated by few features
- For linear regression, logistic regression, SVM, scaling helps
  - Numerical optimization
  - Prevent normalization value being dominated by few parameters
- For decision trees and random forest, scaling or not is not important

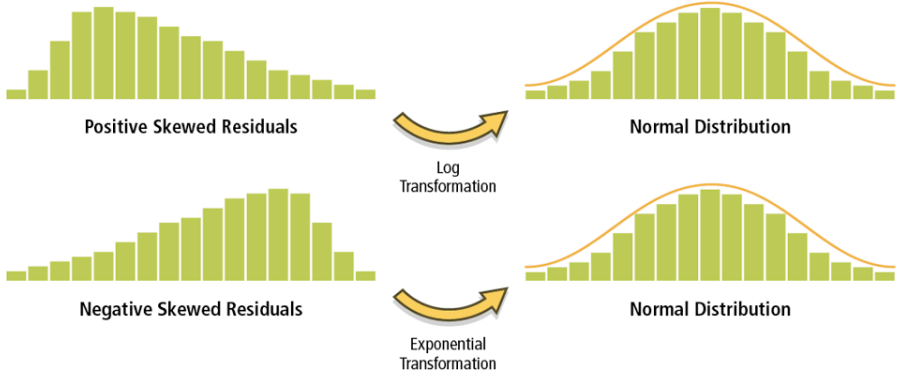
More about log or exp functions

- Apply log or exponential functions to make the skewe features Gaussian-like

If $0 \leq x_{*j} \leq 1$, don't use log directly

- Consider $\log\left(1 + x_{*j}\right)$





http://www.esri.com/news/arcuser/0111/findmodel .html

Thresholding and ranking

- Thresholding: if $x_{*j} > t \Rightarrow 1$ else 0
- Ranking: another way of "scaling" features and handle outliers
  - E.g., [1e5, 0, -1000] $\Rightarrow$ [2, 1, 0] $\Rightarrow$ [1, 0.5, 0]
  - E.g., [0.1, -0.005, π] $\Rightarrow$ [1, 0, 2] $\Rightarrow$[0.5, 0, 1]

Another interesting technique

- Keep only the integer part of the numbers
  - E.g., 3.99 ➔ 3; 3.01 ➔ 3; 1.99 ➔ 1
- Keep only the fractional part of numbers
  - E.g., 3.99 ➔ .99; 3.01 ➔ .01; 1.99 ➔ .99
- Might be useful to capture human's subconsciousness to numbers (e.g., price)

Missing values

- If the type of the feature is categorical
  - Replace the missing values with the most frequent value
- If the type of the feature is numerical
  - Replace the missing values with the mean or median
- Some more advanced techniques
  - Predict the missing values, usually by interpolation and extrapolation

Imbalanced classification

- Most classification algorithms perform optimally when the number of samples in each class is similar
- Common techniques for imbalanced dataset
  - Under-sample the majority class
  - Over-sample the minority class
  - A combination of both under-sampling and over-sampling
- Some more advanced techniques
  - “Generate” simulated instances from the minority class

Selecting user-specified parameters (hyper-parameters)

- Examples of hyper-parameters
  - *K* in KNN
  - *C* in the regularized linear classification
  - Step size in gradient descent
- Select hyper-parameters
  - Strategies
    - Grid search
    - Random search
    - Bayesian optimization
  - Parallelism: train each parameter setting independently
- Testing selected hyper-parameters
  - Cross validation
  - Parallelism: validate each parameter setting independently

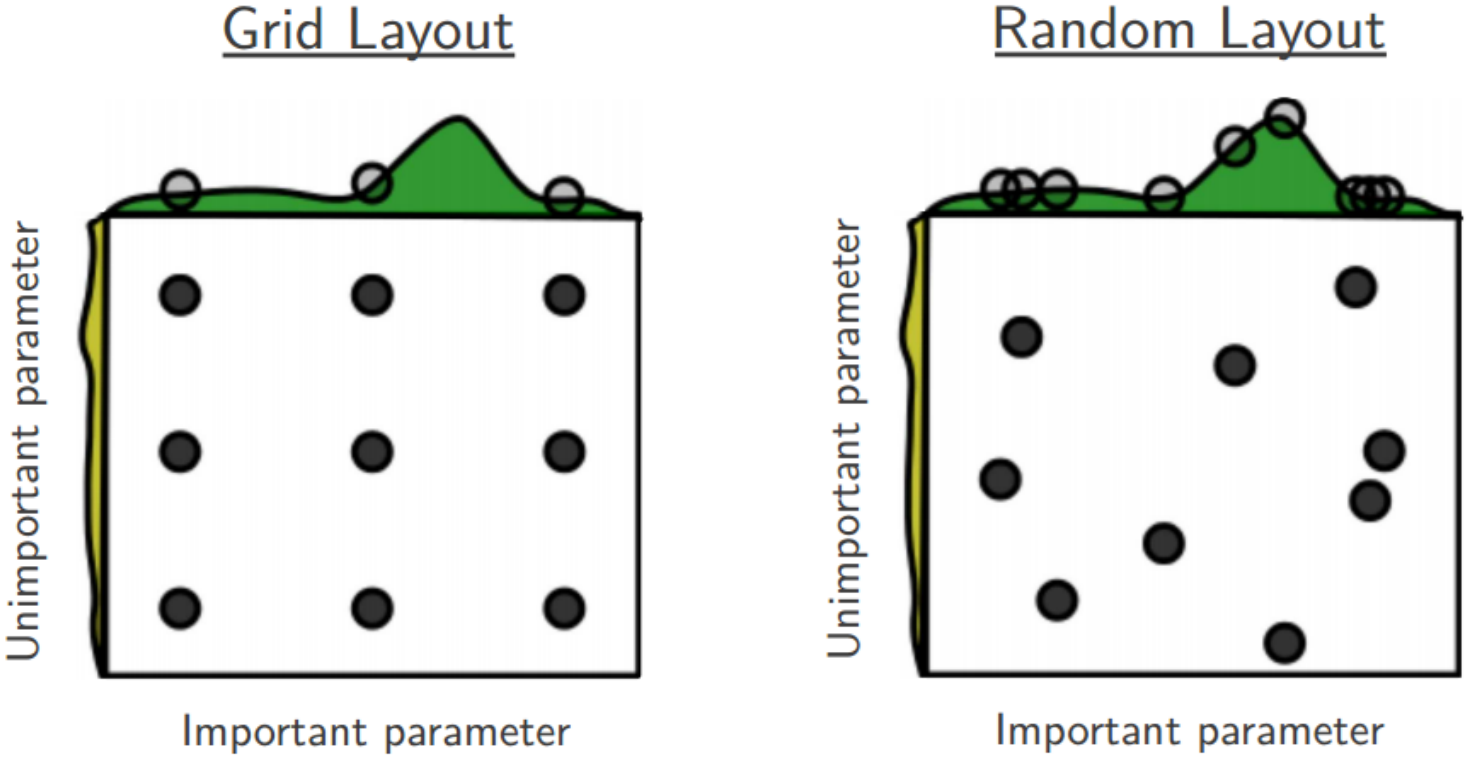
Selecting hyper-parameters

- Grid search
  - A fancy name of exhaustive search
  - Manually specify subset of the hyper-parameter space and step size
  - Try all parameter combinations and check their performance (by cross-validation)
- Random search
  - Surprisingly good performance
    - Among the tunable parameters, important ones are only a few

Selecting hyper-parameters (cont')

- Bayesian optimization
  - Iteratively picking hyper-parameters for experiments
  - Picking strategy: tradeoff between
    - Exploration (hyper-parameters for which the outcome is most uncertain), and
    - Exploitation (hyper-parameters which are expected to have a good outcome)

Random search is surprisingly good



Bergstra and Bengio. "Random search for hyper-parameter optimization." JMLR 2012

Validating selected hyper-parameters

- Use only training data to select hyper-parameters
  - Overfit training data
- Use training data with different hyper-parameters to train models, and test performance based on test data
  - Overfit test data
    - The hyper-parameters are tweaked until the estimator performs well
    - Knowledge of the test data is “leaked”

Validating selected hyper-parameters (cont')

- Training/validation/test datasets
  - Use training data with different hyper-parameters to train the models
  - Use validation set to test the performance of the estimator with different hyper-parameters
    - Obtain $\boldsymbol{\mu}^*$ the set of hyper-parameters that yield the best performance based on the validation set
  - Re-train the model with hyper-parameters $\boldsymbol{\mu}^*$ on **training + validation** datasets
  - Report the final performance based on the test dataset
    - Disadvantage: the # of training and test instances become smaller; not a good news for both training and testing

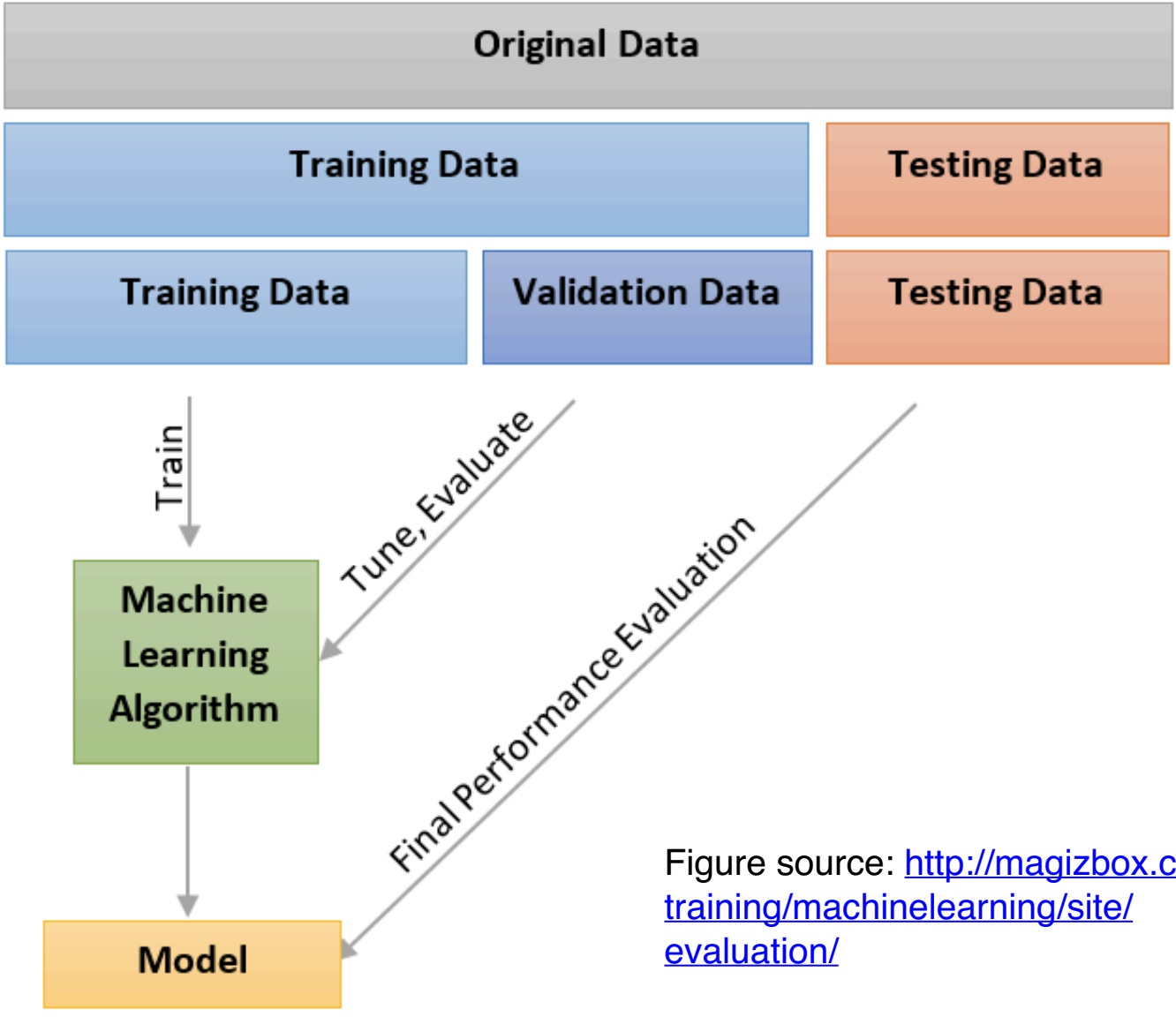
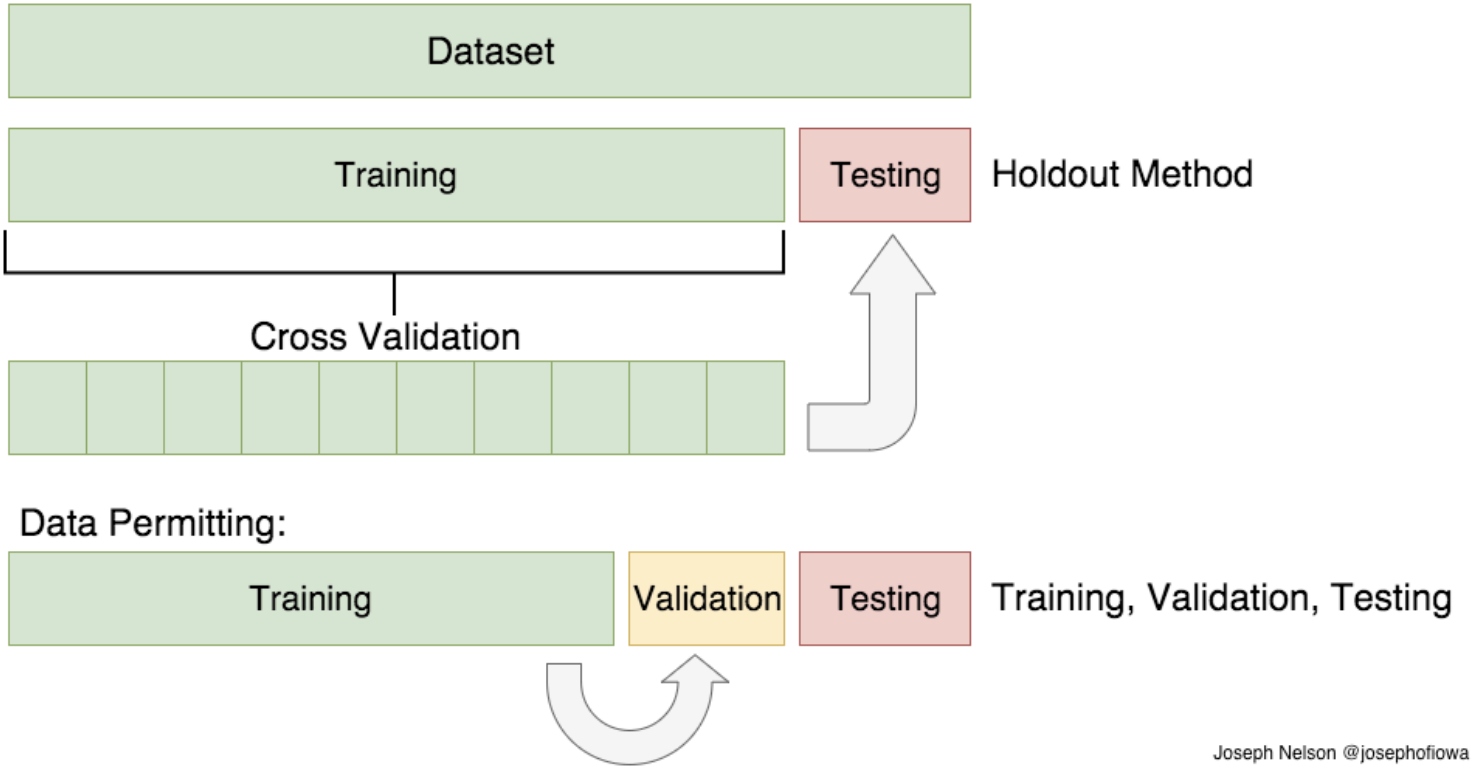


Figure source: http://magizbox.com/training/machinelearning/site/evaluation/

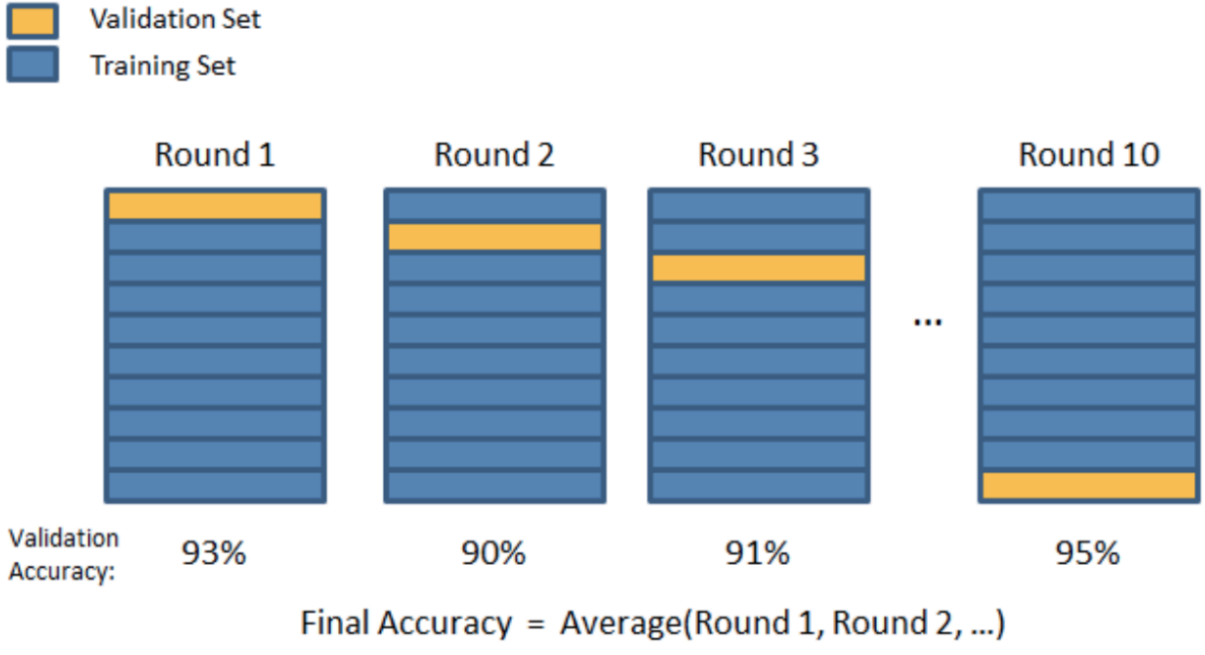
Training/validation/testing



Validating selected hyper-parameters (cont')

- Cross validation (CV)
  - Split the available dataset into training and test
  - The training data is further split into *k* parts (folds)
    - Use (*k*-1) folds as training data
    - Use the remaining fold as the validation data
    - Repeat the above two steps for *k* rounds. The performance of $\boldsymbol{\mu}$ is obtained by averaging the *k* (training, validation) folds
- Advantage: doesn't waste too much data
- Disadvantage: Computationally expensive
  - May boost the efficiency by parallelization

K-fold cross validation



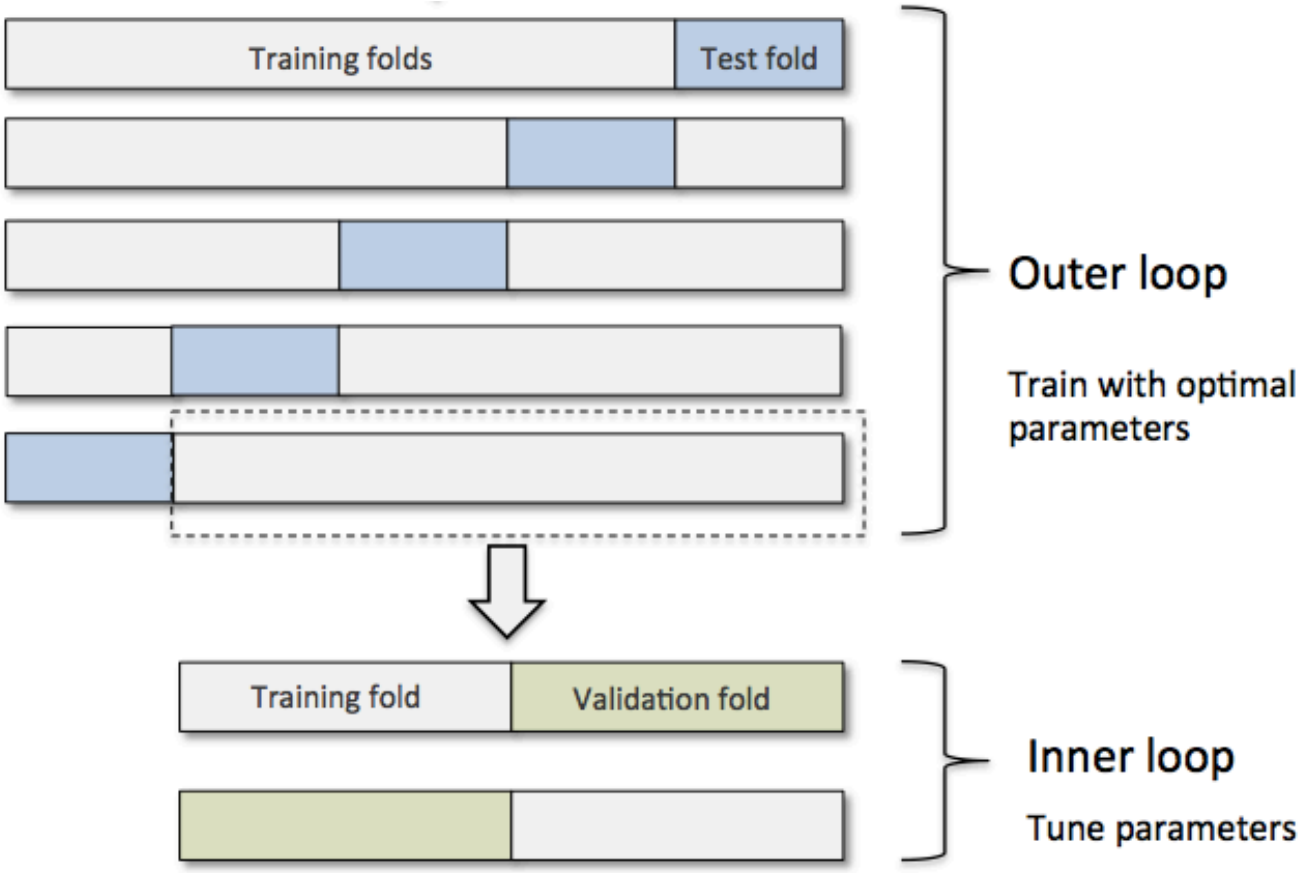
Cross validation

- Types of cross-validation
  - *K*-fold
    - *K* is typically set to 5 or 10
  - Leave-one-out (LOO)
    - Setting *k* equals *n* the number of data instances
  - Leave-P-out (LPO)
    - *P* out of *n* is regarded as the validation set for each round. The procedure produces $C(n, P)$ train-validate pairs.
- Stratification
  - Create splits by preserving the approximately the same percentage of each target label as in the complete dataset
- Time series data
  - Training dataset should contain only the “old” instances; the validation dataset contains the “new” instances

Cross validation (cont')

- An example of the time-series *K*-fold CV
  - $(\mathbf{x}_1, y_1), (\mathbf{x}_2, y_2), \ \ldots, \ (\mathbf{x}_n, y_n)$
  - The data instances are ordered from the oldest to the latest
  - *K*=3
  - The (training, validation) sets are:
    - $[(\mathbf{x}_1, y_1), \ \ldots, \ (\mathbf{x}_{n-3}, \ y_{n-3})], \ [(\mathbf{x}_{n-2}, \ y_{n-2}), \ (\mathbf{x}_{n-1}, \ y_{n-1}), \ (\mathbf{x}_n$
    - $[(\mathbf{x}_1, y_1), \ \ldots, \ (\mathbf{x}_{n-2}, \ y_{n-2})], \ [(\mathbf{x}_{n-1}, \ y_{n-1}), \ (\mathbf{x}_n, y_n)]$
    - $[(\mathbf{x}_1, y_1), \ \ldots, \ (\mathbf{x}_{n-1}, \ y_{n-1})], \ [(\mathbf{x}_n, y_n)]$

Nested cross validation



Three scenarios of evaluation

1. Evaluate a simple model w/o hyper-parameters
   - Split into training and test data
2. Evaluate a model with hyper-parameters
   - Split into training and test data
   - Apply *K*-fold cross validation on the training data to find the best hyper-parameters
   - Re-train the training data based on the best hyper-parameters, and evaluate the performance on test data
3. Compare several models with hyper-parameters
   - Nested cross validation

Source: https://sebastianraschka.com/faq/docs/evaluate-a-model.html

Multi-class classification

- Leverage on binary classifiers
  - One-vs.-rest (aka one-vs-all)
  - One-vs.-one
- Other methods
  - *K*NN
  - Decision trees
  - Neural networks

One-vs.-rest (aka: one-vs.-all)

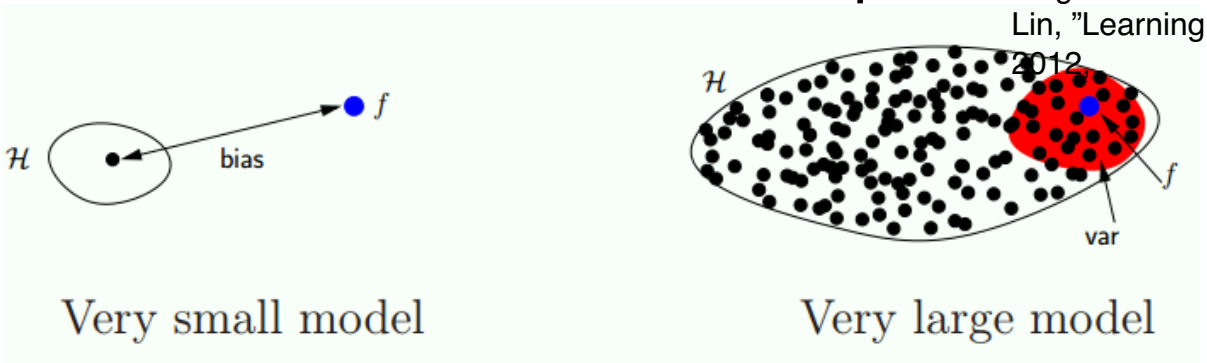
- Train a single classifier for each class
- E.g.,
  - Target labels: “red”, “blue”, or “green”
  - Train three binary classifiers
    - f_1: ”Red” vs “not red”
    - f_2: “Blue” vs “not blue”
    - f_3: “Green” vs “not green”
  - $\hat{y}_i = \arg \max_{k \in \{1,2,3\}} f_k(\mathbf{x}_i)$

One-vs.-one

- Training: for a *k*-nary classification problem, one trains $C(k, 2)$ classifiers
- Test:
  - Feed the test instance to all $C(k, 2)$ classifiers
  - The class receiving the most “+1” predictions is the predicted class

Bias vs variance

- The test error comes from
  - Bias: the error from the difference between the true model and the learning model
  - Variance: the error from sensitivity to the small fluctuations in the training data
  - Noise: the error from the data per se

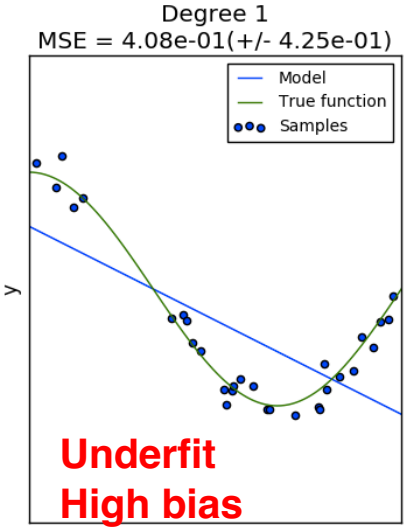


Yaser S. Abu-Mostafa, Malik Magdon-Ismail, Hsuan-Tien Lin, "Learning from data." 2012.

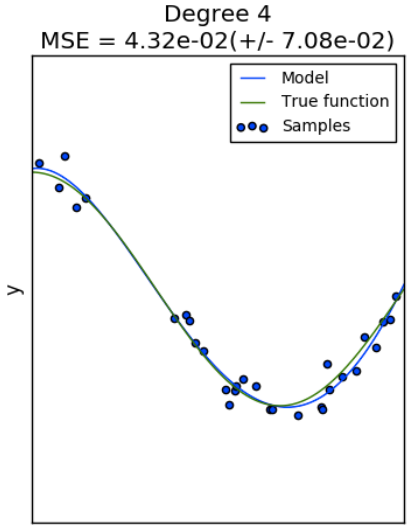
(Bias vs variance) vs (underfit vs overfit)

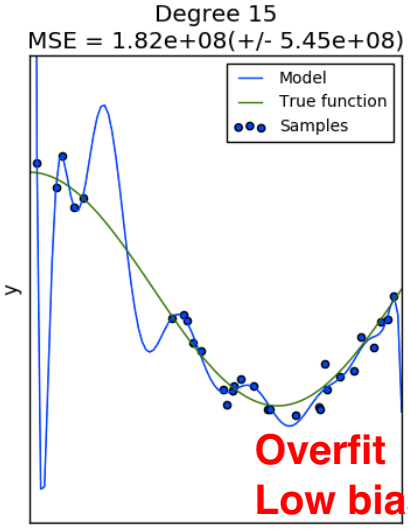
http://scikit-learn.org/stable/auto_examples/model_selection/plot_underfitting_overfitting.html

- True function: $f(x) = \cos\left(\frac{3}{2}\pi x\right) + \epsilon$
- Fitting functions: linear regression with



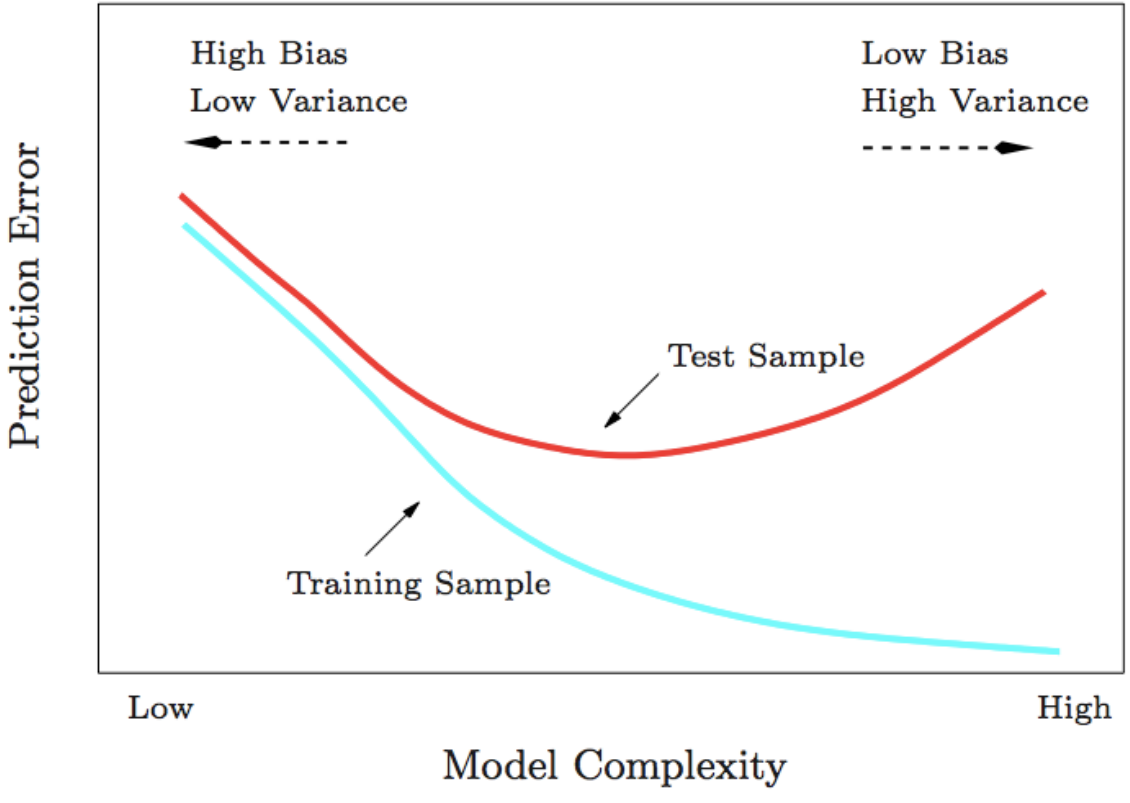
Underfit
High bias
Low variance





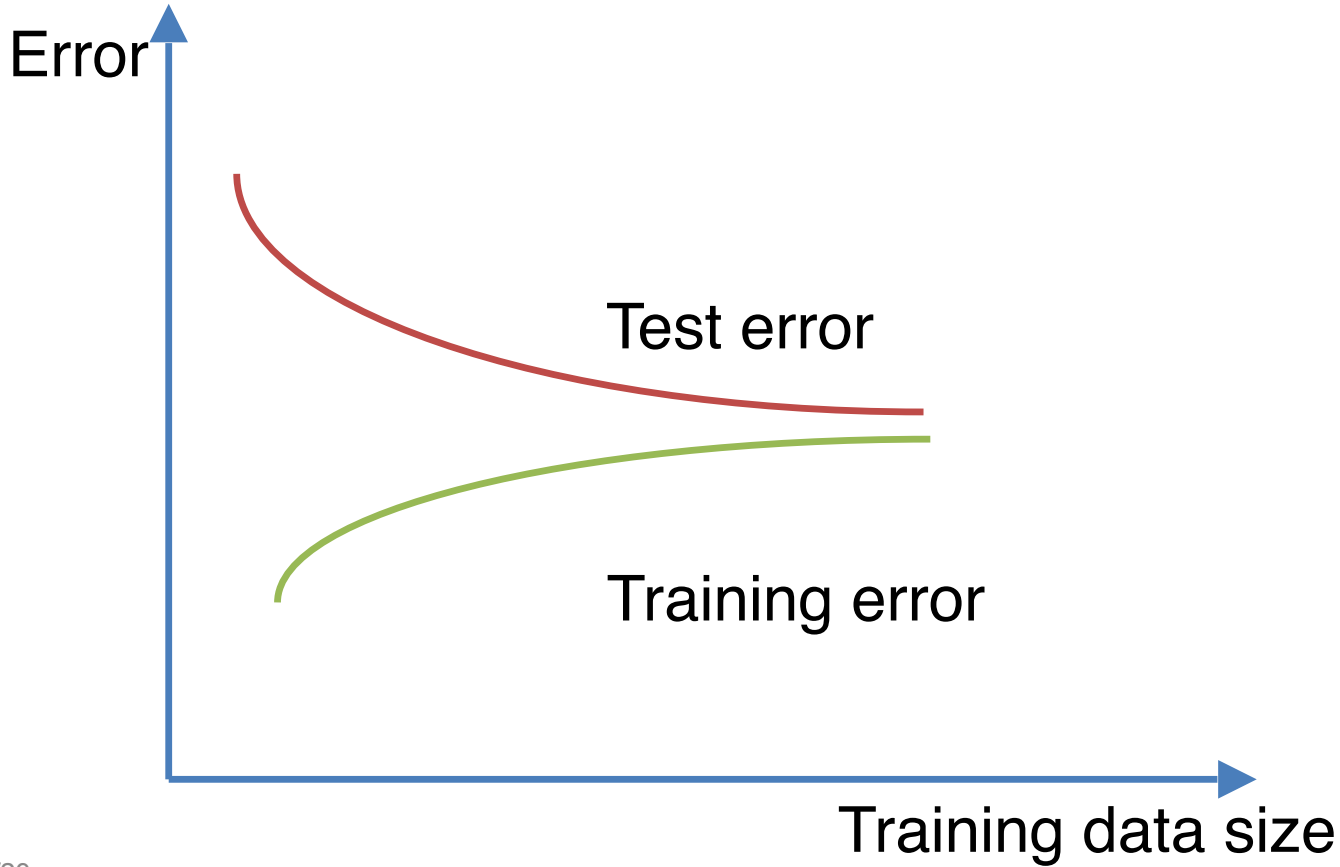
Overfit
Low bias
High variance

Model complexity and over/under-fitting

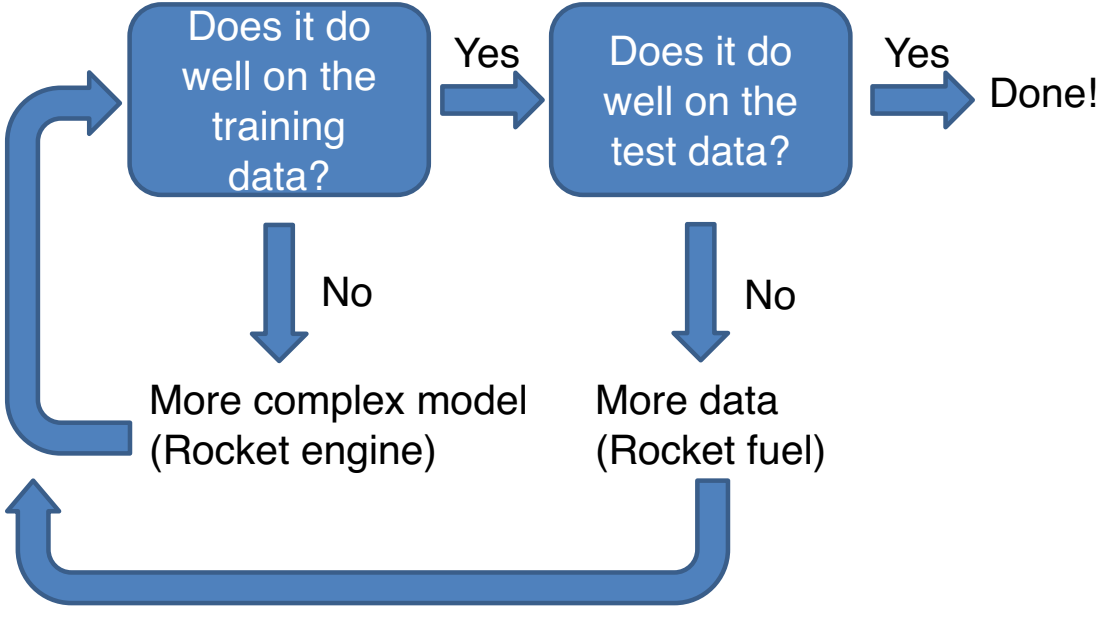


Hastie et al., "The Elements of Statistical Learning." 2001.

Training size and over/under-fitting



Data size vs model complexity



Modified from the slides of Dr. Andrew Ng's talk at the GPU Technology Conference 2015

Model selection

- Domain knowledge is important
  - If you know the relationship between *x* and *y* is linear, why choose quadratic?
- Data size is important
  - Applying complex models on small data tends to over-fitting
  - Applying simple models on massive data tends to under-fitting
- Select an algorithm that is
  - complex enough to fit the training data well, and
  - simple enough to fit the testing (future) data well

Quiz

- If we find a model performs well on training data but poorly on test data
  - Increase or decrease model complexity?
  - Collect more data?
- If we find a model performs roughly the same on both training and testing datasets, and we believe the current accuracy is not good enough
  - Collecting more data?
  - Increasing model complexity and collecting more data for training?

Quiz

- A student write a research paper is the following way

1. Find a problem
2. Collect experimental data for the problem
3. Propose a method for the problem
4. Compare the proposed method with previous methods, based on the collected data
5. If failed, **go to step 4 and try different hyper-parameters**
6. New publication and graduate!

Is this fair?