Model Selection and Optimization

Outline

I. Overfitting and pruning of decision trees

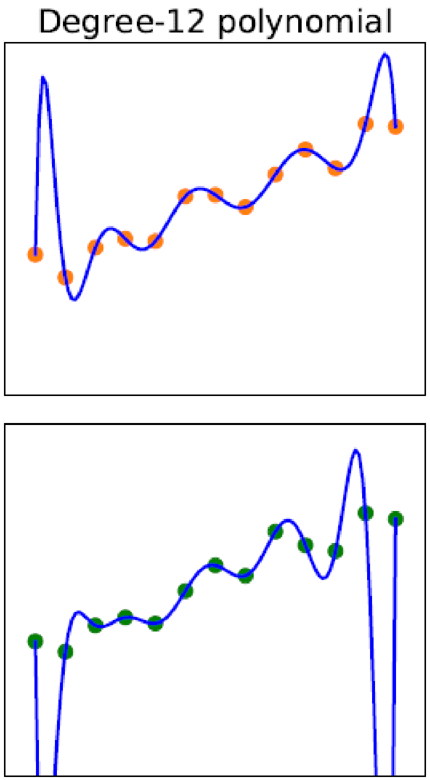
II. Model selection

III. Loss function

IV. Regularization and tuning

* Figures are from the textbook site or plotted by the instructor.

I. Issue of Overfitting



Overfitting

- ♦ Too much attention paid to the training set, and performs poorly on unseen data.
- ♦ More likely to happen as #attributes increases.
- ♦ Less likely with more training examples.

- ♠ Polynomials with higher degrees are more likely to overfit.
- ♠ Decision trees with more nodes have more capacity to overfit.

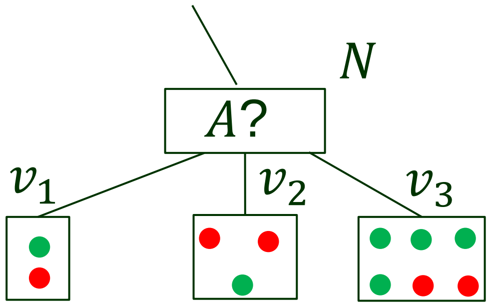
Decision Tree Pruning

- Construct a full decision tree.
- Repeat the following steps:

Decision Tree Pruning

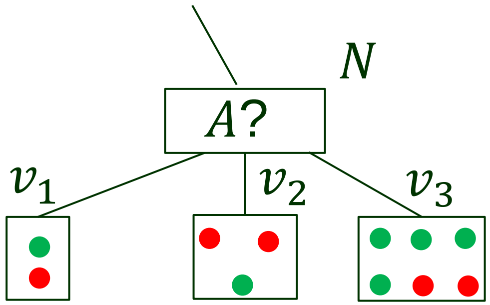
- Construct a full decision tree.
- Repeat the following steps:
  - Consider a test node N with leaf children only.

Decision Tree Pruning



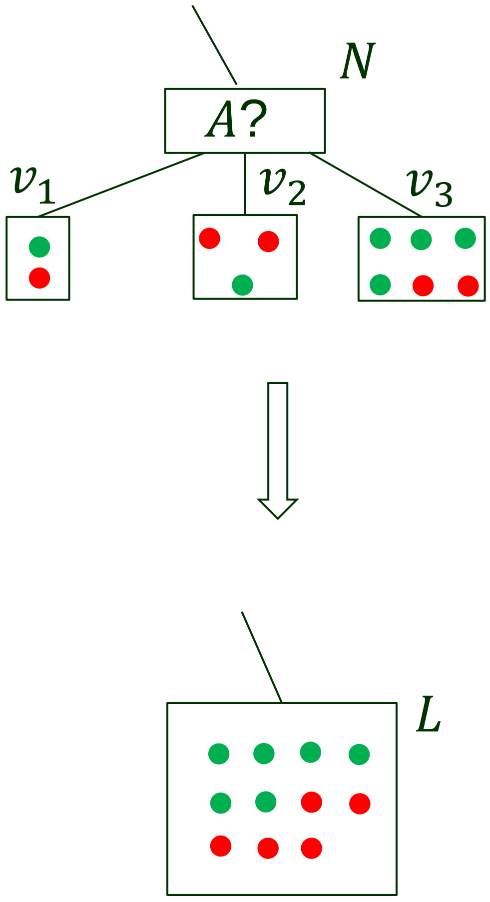
- Construct a full decision tree.
- Repeat the following steps:
  - Consider a test node N with leaf children only.

Decision Tree Pruning



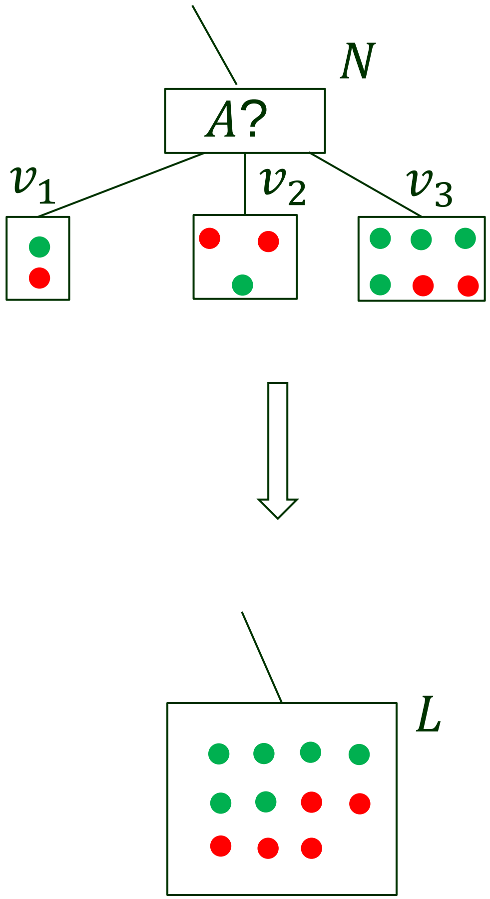
- Construct a full decision tree.
- Repeat the following steps:
  - Consider a test node N with leaf children only.
  - Replace the node with a leaf node L if the test appears to be *irrelevant*.

Decision Tree Pruning



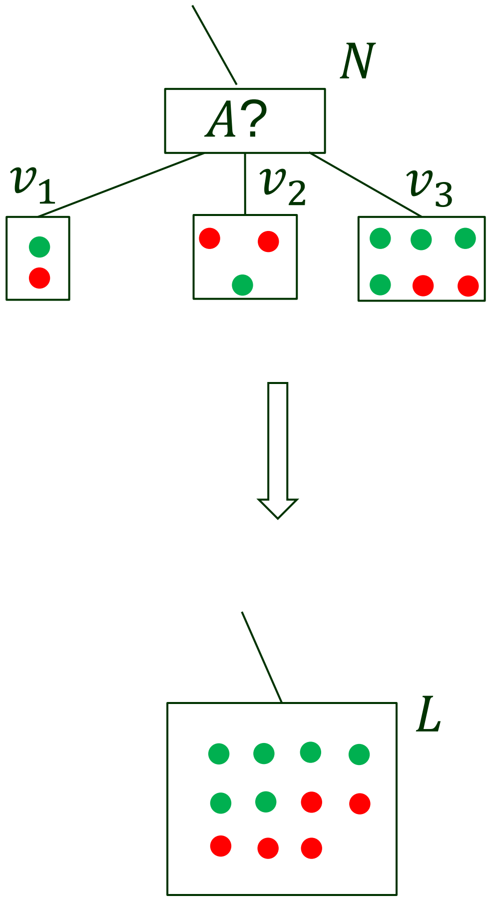
- Construct a full decision tree.
- Repeat the following steps:
    - Consider a test node N with leaf children only.
    - Replace the node with a leaf node L if the test appears to be *irrelevant*.

Decision Tree Pruning



- Construct a full decision tree.
- Repeat the following steps:
  - Consider a test node N with leaf children only.
  - Replace the node with a leaf node L if the test appears to be *irrelevant*.
  - How to decide that the attribute A is irrelevant?

Decision Tree Pruning

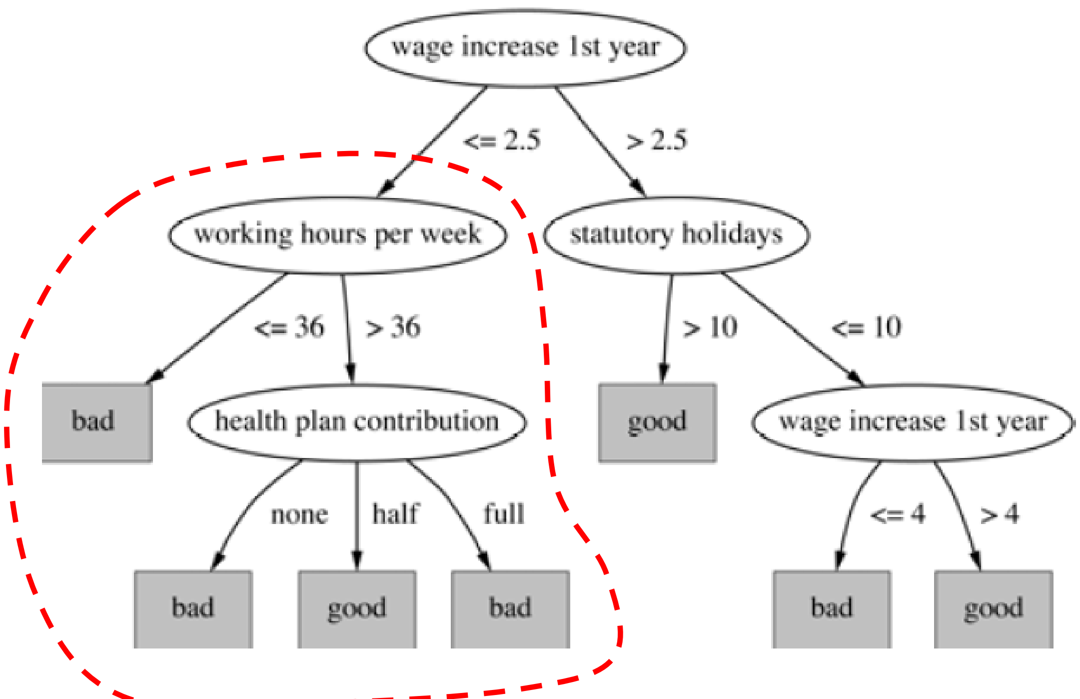


- Construct a full decision tree.
- Repeat the following steps:
  - Consider a test node N with leaf children only.
  - Replace the node with a leaf node L if the test appears to be *irrelevant*.
  - How to decide that the attribute A is irrelevant?

Low information gain.

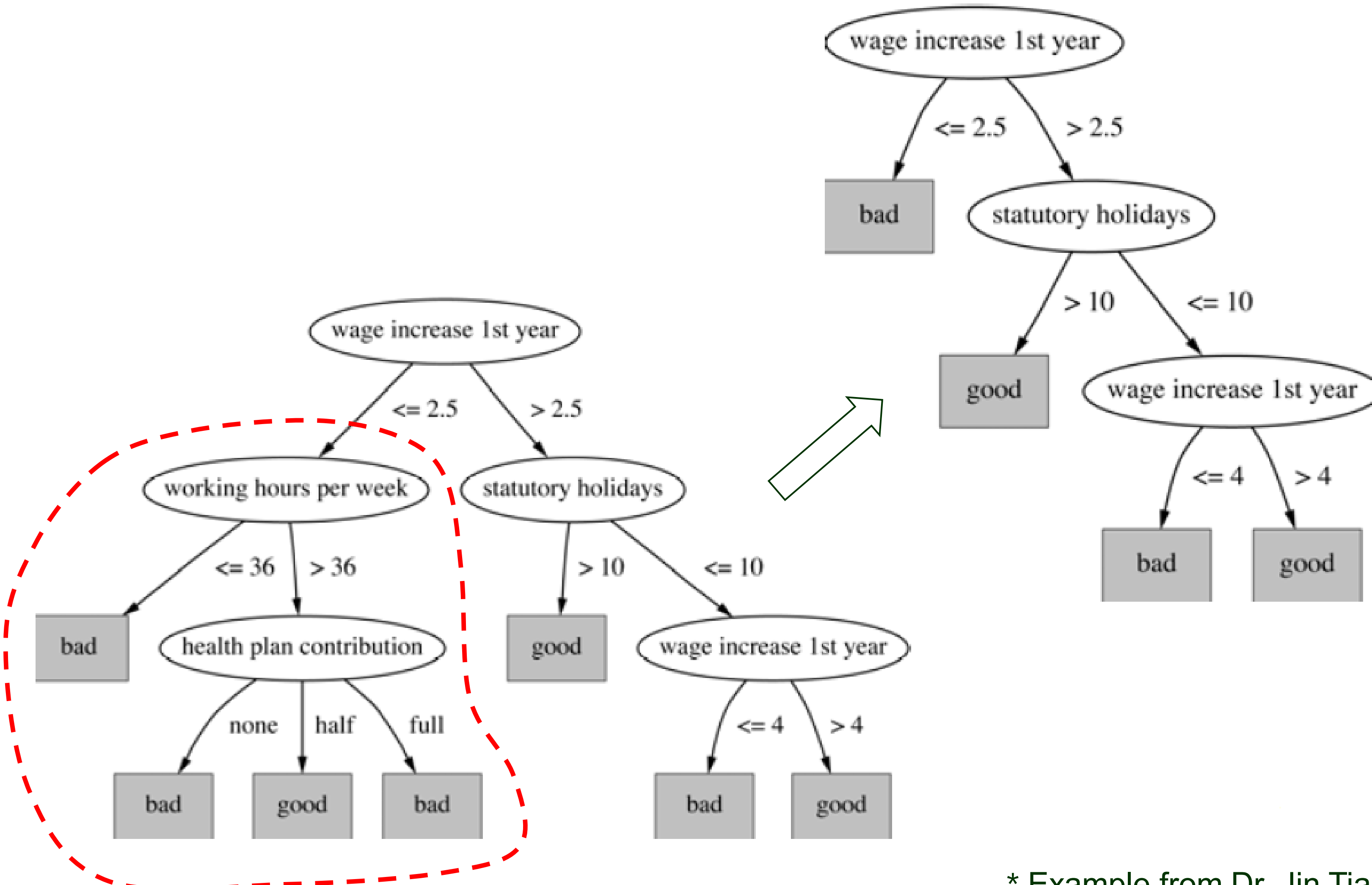
$$Gain(A) = B\left(\frac{p}{p+n}\right) - \sum_{k=1}^{d} \frac{p_k+n_k}{p+n} B\left(\frac{p_k}{p_k+n_k}\right)$$

Example of Pruning



* Example from Dr. Jin Tian's notes.

Example of Pruning



* Example from Dr. Jin Tian's notes.

Significance Test

- The node N has p positive and n negative examples.
- Testing of attribute A at N splits the set into d subsets.
- For $1 \leq k \leq d$, subset k has p_k positive and n_k negative examples.

Significance Test

- The node N has p positive and n negative examples.
- Testing of attribute A at N splits the set into d subsets.
- For $1 \leq k \leq d$, subset k has p_k positive and n_k negative examples.
- Expected numbers of positive and negative examples are

$$\hat{p}_k = p \cdot \frac{p_k + n_k}{p + n} \qquad \hat{n}_k = n \cdot \frac{p_k + n_k}{p + n}$$

Significance Test

- The node N has p positive and n negative examples.
- Testing of attribute A at N splits the set into d subsets.
- For $1 \leq k \leq d$, subset k has p_k positive and n_k negative examples.
- Expected numbers of positive and negative examples are

$$\hat{p}_k = p \cdot \frac{p_k + n_k}{p + n} \qquad \hat{n}_k = n \cdot \frac{p_k + n_k}{p + n}$$

- Measure of the total deviation is given by

$$\Delta = \sum_{k=1}^{d} \left(\frac{(p_k - \hat{p}_k)^2}{\hat{p}_k} + \frac{(n_k - \hat{n}_k)^2}{\hat{n}_k} \right)$$

Significance Test

- The node N has p positive and n negative examples.
- Testing of attribute A at N splits the set into d subsets.
- For $1 \leq k \leq d$, subset k has p_k positive and n_k negative examples.
- Expected numbers of positive and negative examples are

$$\hat{p}_k = p \cdot \frac{p_k + n_k}{p + n} \qquad \hat{n}_k = n \cdot \frac{p_k + n_k}{p + n}$$

- Measure of the total deviation is given by

$$\underbrace{\Delta = \sum_{k=1}^{d} \left(\frac{(p_k - \hat{p}_k)^2}{\hat{p}_k} + \frac{(n_k - \hat{n}_k)^2}{\hat{n}_k} \right)}$$

χ^2 distribution with $d - 1$ degrees of freedom

Significance Test

- The node N has p positive and n negative examples.
- Testing of attribute A at N splits the set into d subsets.
- For $1 \le k \le d$, subset k has p_k positive and n_k negative examples.
- Expected numbers of positive and negative examples are

$$\hat{p}_k = p \cdot \frac{p_k + n_k}{p+n} \qquad \hat{n}_k = n \cdot \frac{p_k + n_k}{p+n}$$

- Measure of the total deviation is given by

$$\underbrace{\Delta = \sum_{k=1}^{d} \left(\frac{(p_k - \hat{p}_k)^2}{\hat{p}_k} + \frac{(n_k - \hat{n}_k)^2}{\hat{n}_k} \right)}_{\chi^2 \text{ distribution with } d-1 \text{ degrees of freedom}}$$

Apply χ^2 pruning:

$\Delta = 7.82$ would mean the attribute is irrelevant at the 5% level. (The higher the value of Δ, the more significant the attribute is.)

II. Model Selection

Goal: Select a hypothesis that will *optimally fit* *future examples*.

- ♦ Future examples are assumed to be like the past.

Stationarity:

$$\boldsymbol{P}(E_j) = \boldsymbol{P}(E_{j+1}) = \boldsymbol{P}(E_{j+2}) = \cdots$$

// every example has the
// the same prior probability.

$$\boldsymbol{P}(E_j) = \boldsymbol{P}(E_j \mid E_{j-1}, E_{j-2}, \ldots)$$

// every example is independent
// of the previous examples.

Optimal Fit

Split the examples into three sets:

- a *training set* to train candidate models (hypotheses)
- a *validation set* to evaluate the candidate models and choose the best one
- a *test set* to do a final unbiased evaluation of the best model

Optimal Fit

Split the examples into three sets:

- a *training set* to train candidate models (hypotheses)
- a *validation set* to evaluate the candidate models and choose the best one
- a *test set* to do a final unbiased evaluation of the best model

Two tasks:

- *Model selection* chooses a good hypothesis space.
- *Optimization* (aka *training*) finds the best hypothesis within that space.

Model Selection via Cross Validation

function MODEL-SELECTION($Learner$, $examples$, k) **returns** a (hypothesis, error rate) pair

$err \leftarrow$ an array, indexed by $size$, storing validation-set error rates
$training\_set$, $test\_set \leftarrow$ a partition of $examples$ into two sets
for $size$ = 1 **to** ∞ **do**
  $err[size] \leftarrow$ CROSS-VALIDATION($Learner$, $size$, $training\_set$, k)
  if err is starting to increase significantly **then**
    $best\_size \leftarrow$ the value of $size$ with minimum $err[size]$
    $h \leftarrow Learner(best\_size, training\_set)$
    return h, ERROR-RATE(h, $test\_set$)

function CROSS-VALIDATION($Learner$, $size$, $examples$, k) **returns** error rate

$N \leftarrow$ the number of $examples$
$errs \leftarrow 0$
for i = 1 **to** k **do**
  $validation\_set \leftarrow examples[(i - 1) \times N/k{:}i \times N/k]$
  $training\_set \leftarrow examples - validation\_set$
  $h \leftarrow Learner(size, training\_set)$
  $errs \leftarrow errs$ + ERROR-RATE(h, $validation\_set$)
return $errs$ / k // *average error rate on validation sets, across k-fold cross-validation*

Model Selection via Cross Validation

```
function MODEL-SELECTION(Learner, examples, k) returns a (hypothesis, error rate) pair

  err ← an array, indexed by size, storing validation-set error rates
  training_set, test_set ← a partition of examples into two sets
  for size = 1 to ∞ do                                              // size is a hyperparameter
     err[size] ← CROSS-VALIDATION(Learner, size, training_set, k)   // of the model class, e.g.,
     if err is starting to increase significantly then              // #nodes in a decision tree
        best_size ← the value of size with minimum err[size]       // degree of polynomials.
        h ← Learner(best_size, training_set)
        return h, ERROR-RATE(h, test_set)

function CROSS-VALIDATION(Learner, size, examples, k) returns error rate

  N ← the number of examples
  errs ← 0
  for i = 1 to k do
     validation_set ← examples[(i − 1) × N/k:i × N/k]
     training_set ← examples − validation_set
     h ← Learner(size, training_set)
     errs ← errs + ERROR-RATE(h, validation_set)
  return errs / k          // average error rate on validation sets, across k-fold cross-validation
```

Model Selection via Cross Validation

function MODEL-SELECTION($Learner, examples, k$) **returns** a (hypothesis, error rate) pair

$err \leftarrow$ an array, indexed by $size$, storing validation-set error rates
$training\_set,\ test\_set \leftarrow$ a partition of $examples$ into two sets
for $size = 1$ **to** ∞ **do**
  $err[size] \leftarrow$ CROSS-VALIDATION($Learner, size, training\_set, k$)
  if err is starting to increase significantly **then**
    $best\_size \leftarrow$ the value of $size$ with minimum $err[size]$
    $h \leftarrow Learner(best\_size, training\_set)$
    return h, ERROR-RATE($h,\ test\_set$)

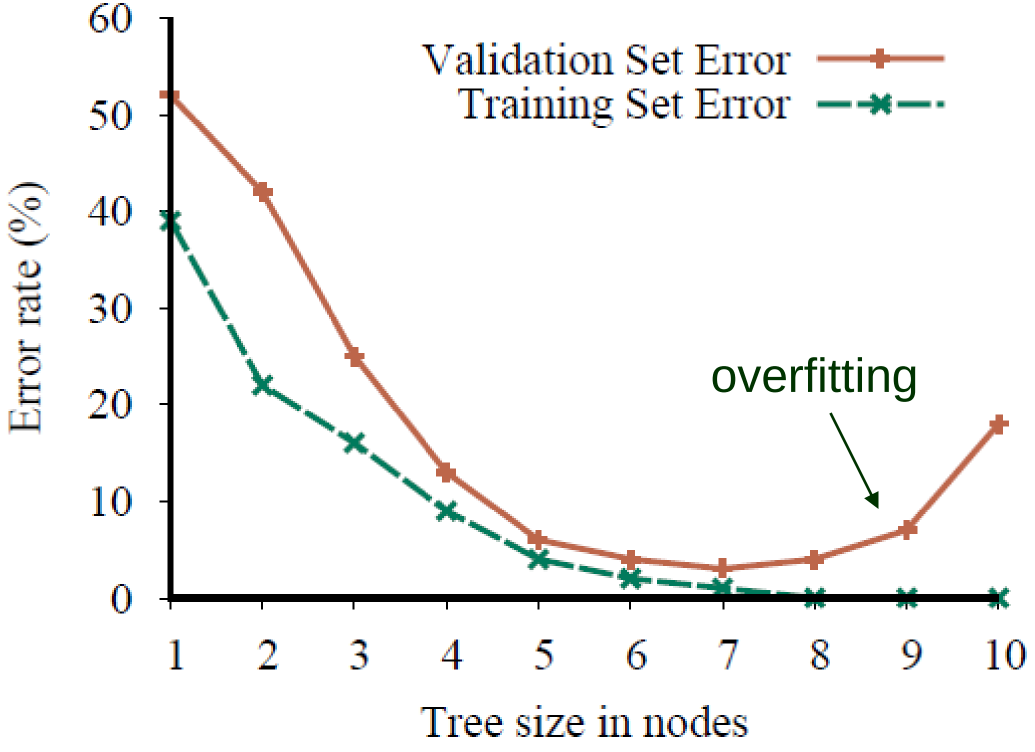
// *size* is a *hyperparameter*
// of the model class, e.g.,
// #nodes in a decision tree
// degree of polynomials.

function CROSS-VALIDATION($Learner, size, examples, k$) **returns** error rate

$N \leftarrow$ the number of $examples$
$errs \leftarrow 0$
for $i = 1$ **to** k **do**
  $validation\_set \leftarrow examples[(i - 1) \times N/k{:}i \times N/k]$
  $training\_set \leftarrow examples - validation\_set$
  $h \leftarrow Learner(size, training\_set)$
  $errs \leftarrow errs +$ ERROR-RATE($h, validation\_set$)
return $errs$ / k // *average error rate on validation sets, across k-fold cross-validation*

// validation set rotates among
// k equally partitioned subsets
// of examples. A large error due
// to overfitting is expected for
// a large k value.

Training and Validation Errors (1)

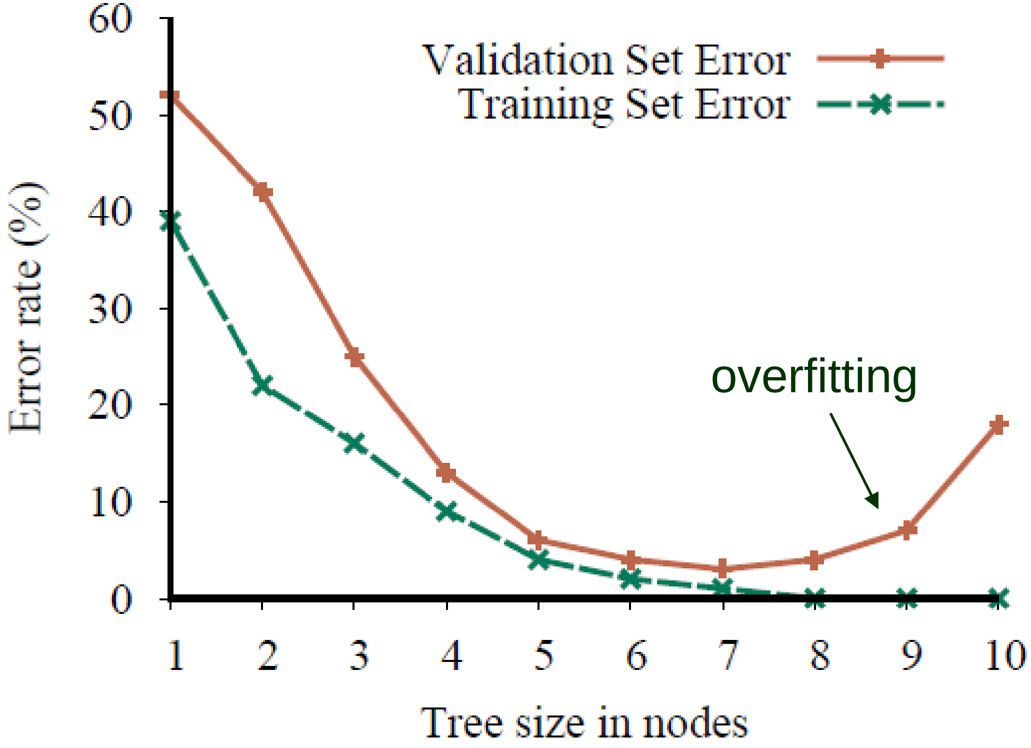


Data: obtained from a version of the restaurant problem

Model class: decision trees

Hyperparameter: #nodes

Training and Validation Errors (1)



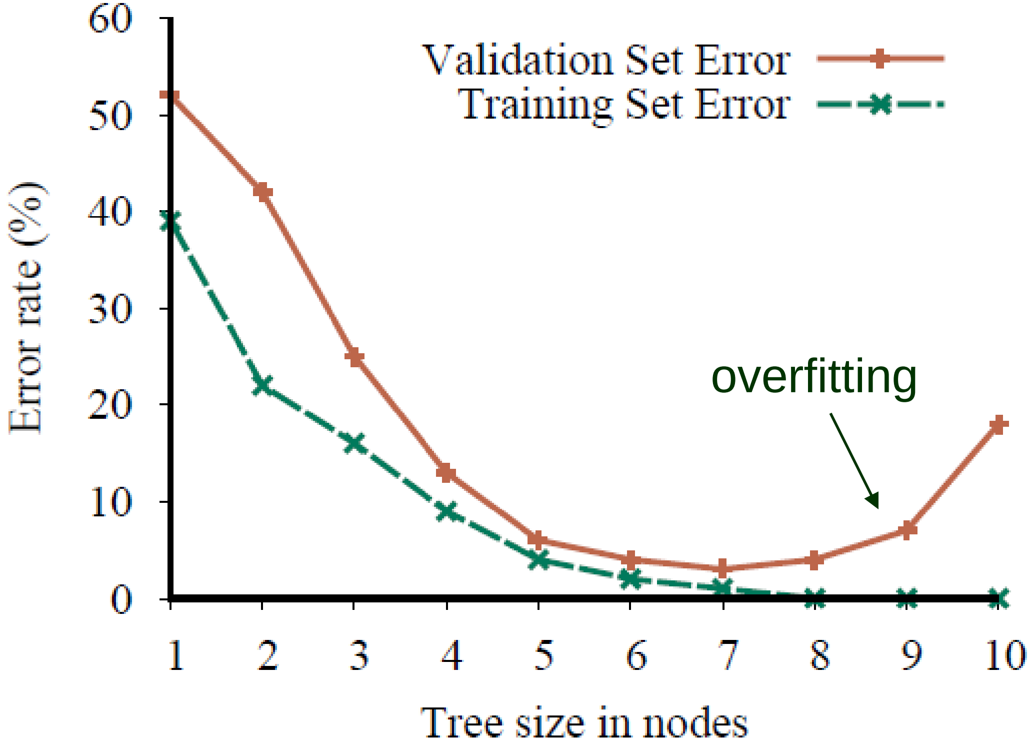
Data: obtained from a version of the restaurant problem

Model class: decision trees

Hyperparameter: #nodes

- The training set error decreases monotonically as the model complexity (tree size) increases.

Training and Validation Errors (1)



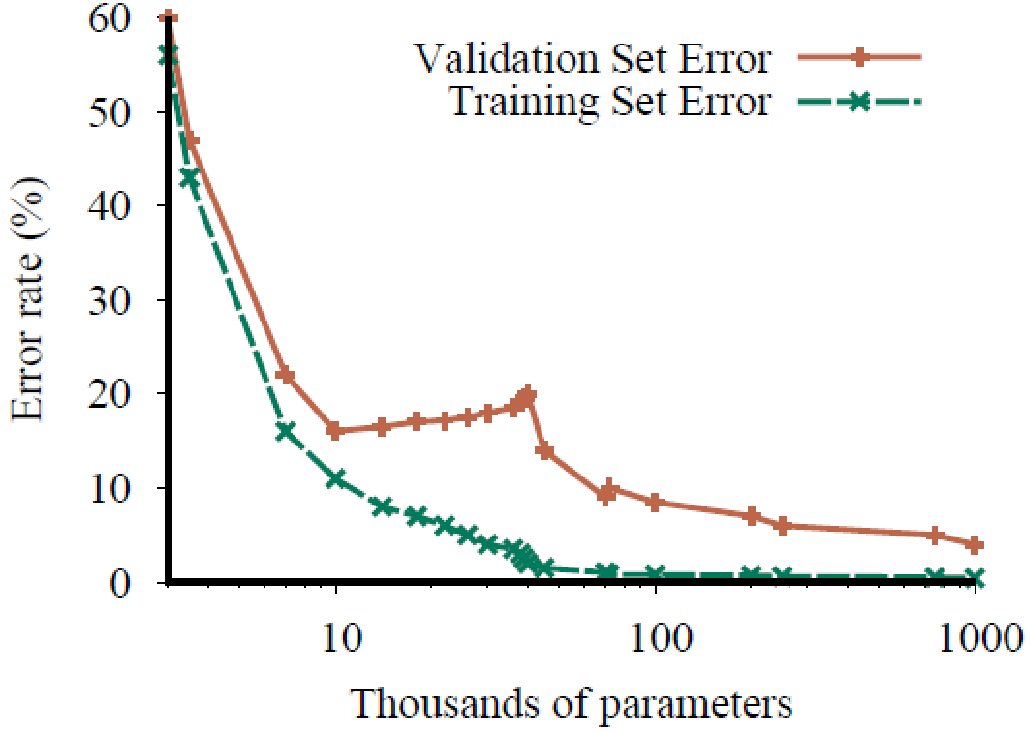
Data: obtained from a version of the restaurant problem

Model class: decision trees

Hyperparameter: #nodes

- ♦ The training set error decreases monotonically as the model complexity (tree size) increases.
- ♠ As the decision tree's size increases, the validation error initially decreases (until tree size = 7), and later increases since the model starts to overfit.

Training and Validation Errors (2)

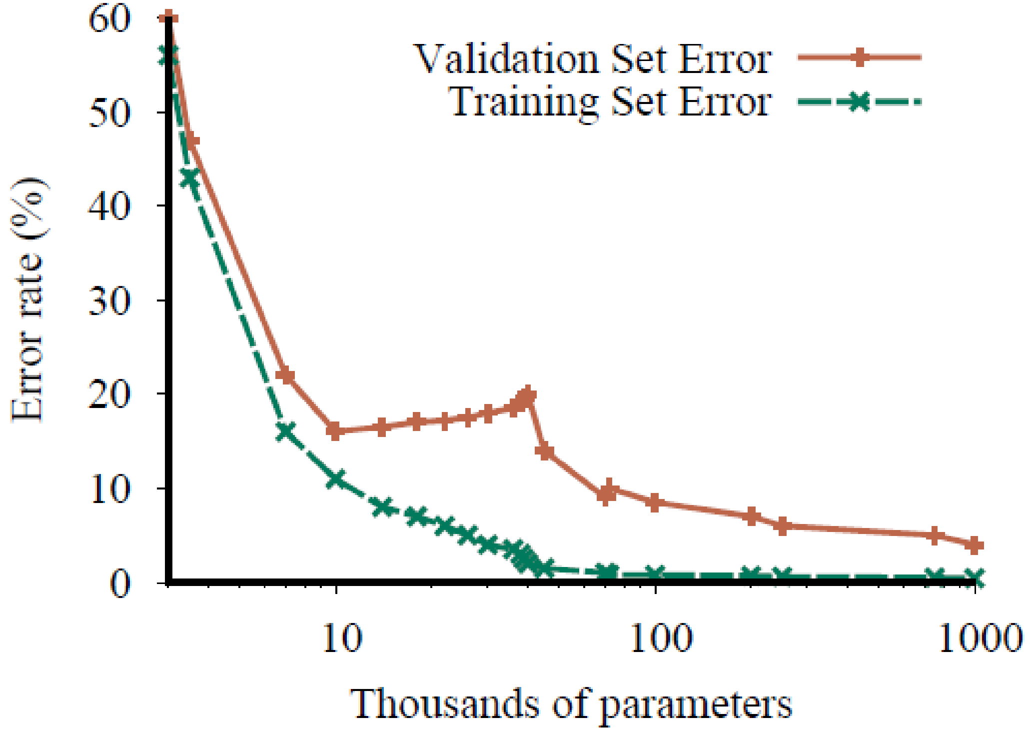


Data: MNIST data set of images of digits

Model class: convolutional neural networks (CNNs)

Hyperparameter: number of regular parameters

Training and Validation Errors (2)



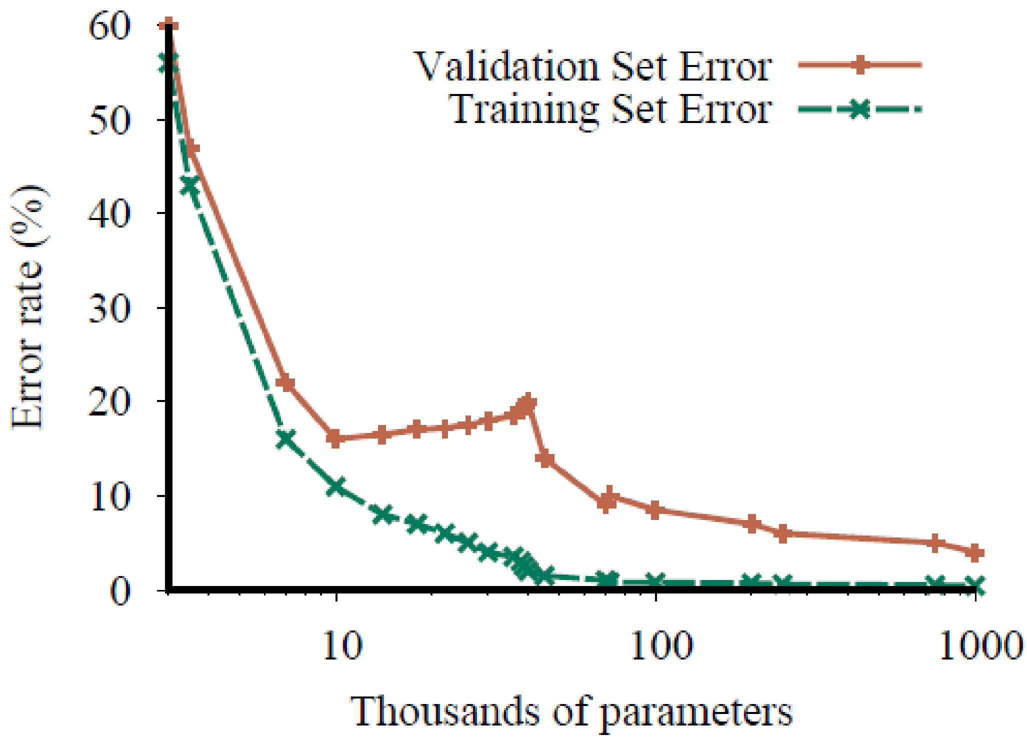
Data: MNIST data set of images of digits

Model class: convolutional neural networks (CNNs)

Hyperparameter: number of regular parameters

- The training set error decreases monotonically.

Training and Validation Errors (2)



Data: MNIST data set of images of digits

Model class: convolutional neural networks (CNNs)

Hyperparameter: number of regular parameters

- The training set error decreases monotonically.
- The validation error shows an initial U-shaped curve and then starts to decrease again, reaching the lowest value at the max number (1,000,000) of parameters.

III. Loss Function

- Not all errors should be weighted equally.

  Spam e-mail filtering: It’s worse to classify non-spam as spam than to classify spam as non-spam.

III. Loss Function

- Not all errors should be weighted equally.

  Spam e-mail filtering: It’s worse to classify non-spam as spam than to classify spam as non-spam.

- Rather than maximizing a utility function, ML minimizes a *loss function*.

III. Loss Function

- Not all errors should be weighted equally.

  Spam e-mail filtering: It's worse to classify non-spam as spam than to classify spam as non-spam.

- Rather than maximizing a utility function, ML minimizes a *loss function*.

  $L(x, y, \hat{y})$: the amount of utility lost by predicting $h(x) = \hat{y}$ when the correct answer is $f(x) = y$.

  $L(x, y, \hat{y}) =$ Utility (result of using y given an input x)
  $-$ Utility (result of using $\hat{y}$ given x)

III. Loss Function

- Not all errors should be weighted equally.

  Spam e-mail filtering: It's worse to classify non-spam as spam than to classify spam as non-spam.

- Rather than maximizing a utility function, ML minimizes a *loss function*.

  $L(x, y, \hat{y})$: the amount of utility lost by predicting $h(x) = \hat{y}$ when the correct answer is $f(x) = y$.

$L(y, \hat{y})$ ⇐ simplified to

$L(x, y, \hat{y})$ = Utility (result of using y given an input x)
– Utility (result of using $\hat{y}$ given x)

III. Loss Function

- Not all errors should be weighted equally.

  Spam e-mail filtering: It's worse to classify non-spam as spam than to classify spam as non-spam.

- Rather than maximizing a utility function, ML minimizes a *loss function*.

  $L(x, y, \hat{y})$: the amount of utility lost by predicting $h(x) = \hat{y}$ when the correct answer is $f(x) = y$.

$L(y, \hat{y})$ ⇐ simplified to

$L(x, y, \hat{y}) =$ Utility (result of using y given an input x)
$-$ Utility (result of using $\hat{y}$ given x)

$L(\mathit{spam}, \mathit{nospam}) = 1$

III. Loss Function

- Not all errors should be weighted equally.

  Spam e-mail filtering: It's worse to classify non-spam as spam than to classify spam as non-spam.

- Rather than maximizing a utility function, ML minimizes a *loss function*.

  $L(x, y, \hat{y})$: the amount of utility lost by predicting $h(x) = \hat{y}$ when the correct answer is $f(x) = y$.

$L(y, \hat{y})$ $\Leftarrow$ simplified to

$L(x, y, \hat{y})$ = Utility (result of using y given an input x)
– Utility (result of using $\hat{y}$ given x)

$L(\underbrace{spam}_{\text{truth}}, \underbrace{nospam}_{\text{prediction}}) = 1$

III. Loss Function

- Not all errors should be weighted equally.

  Spam e-mail filtering: It's worse to classify non-spam as spam than to classify spam as non-spam.

- Rather than maximizing a utility function, ML minimizes a *loss function*.

  $L(x, y, \hat{y})$: the amount of utility lost by predicting $h(x) = \hat{y}$ when the correct answer is $f(x) = y$.

$L(y, \hat{y}) \Longleftarrow$ simplified to

$L(x, y, \hat{y})$ = Utility (result of using y given an input x)
− Utility (result of using $\hat{y}$ given x)

$L(\underbrace{spam}_{\text{truth}}, \underbrace{nospam}_{\text{prediction}}) = 1$ $\quad$ $L(nospam, spam) = 10$

Generalization Loss

Absolute-value loss: $L_1(y, \hat{y}) = |y - \hat{y}|$

Squared-error loss: $L_2(y, \hat{y}) = (y - \hat{y})^2$

0/1 loss: $L_{0/1}(y, \hat{y}) = 0$ if $y = \hat{y}$, and 1 otherwise

Generalization Loss

Absolute-value loss: $L_1(y, \hat{y}) = |y - \hat{y}|$

Squared-error loss: $L_2(y, \hat{y}) = (y - \hat{y})^2$

0/1 loss: $L_{0/1}(y, \hat{y}) = 0$ if $y = \hat{y}$, and 1 otherwise

ML chooses the hypothesis that minimizes *expected loss* over the set $\mathcal{E}$ of all input-output pairs.

Generalization Loss

Absolute-value loss: $L_1(y, \hat{y}) = |y - \hat{y}|$

Squared-error loss: $L_2(y, \hat{y}) = (y - \hat{y})^2$

0/1 loss: $L_{0/1}(y, \hat{y}) = 0$ if $y = \hat{y}$, and 1 otherwise

ML chooses the hypothesis that minimizes *expected loss* over the set $\mathcal{E}$ of all input-output pairs.

Generalization loss for a hypothesis h w.r.t. loss function L:

Generalization Loss

Absolute-value loss: $L_1(y, \hat{y}) = |y - \hat{y}|$

Squared-error loss: $L_2(y, \hat{y}) = (y - \hat{y})^2$

0/1 loss: $L_{0/1}(y, \hat{y}) = 0$ if $y = \hat{y}$, and 1 otherwise

ML chooses the hypothesis that minimizes *expected loss* over the set $\mathcal{E}$ of all input-output pairs.

Generalization loss for a hypothesis h w.r.t. loss function L:

$$\mathit{GenLoss}_L(h) = \sum_{(x,y)\in\mathcal{E}} L\big(y, h(x)\big)P(x, y)$$

Generalization Loss

Absolute-value loss: $L_1(y, \hat{y}) = |y - \hat{y}|$

Squared-error loss: $L_2(y, \hat{y}) = (y - \hat{y})^2$

0/1 loss: $L_{0/1}(y, \hat{y}) = 0$ if $y = \hat{y}$, and 1 otherwise

ML chooses the hypothesis that minimizes *expected loss* over the set $\mathcal{E}$ of all input-output pairs.

Generalization loss for a hypothesis h w.r.t. loss function L:

$$\mathit{GenLoss}_L(h) = \sum_{(x,y)\in\mathcal{E}} L\big(y, h(x)\big)\underbrace{P(x, y)}_{\text{prior joint probability}}$$

Generalization Loss

Absolute-value loss: $L_1(y, \hat{y}) = |y - \hat{y}|$

Squared-error loss: $L_2(y, \hat{y}) = (y - \hat{y})^2$

0/1 loss: $L_{0/1}(y, \hat{y}) = 0$ if $y = \hat{y}$, and 1 otherwise

ML chooses the hypothesis that minimizes *expected loss* over the set $\mathcal{E}$ of all input-output pairs.

Generalization loss for a hypothesis h w.r.t. loss function L:

$$\mathit{GenLoss}_L(h) = \sum_{(x,y)\in\mathcal{E}} L\big(y, h(x)\big)\underbrace{P(x,y)}_{\text{prior joint probability}}$$

Best hypothesis is chosen as

$$h^* = \operatorname*{argmin}_{h\in\mathcal{H}} \mathit{GenLoss}_L(h)$$

Empirical Loss

- $GenLoss_L(h)$ can only be estimated since $P(x, y)$ is often unknown.

Empirical loss on a set of examples E of size N:

$$EmpLoss_{L,E}(h) = \sum_{(x,y)\in\mathcal{E}} L\big(y, h(x)\big)\frac{1}{N}$$

Empirical Loss

- $GenLoss_L(h)$ can only be estimated since $P(x, y)$ is often unknown.

Empirical loss on a set of examples E of size N:

$$EmpLoss_{L,E}(h) = \sum_{(x,y)\in\mathcal{E}} L\big(y, h(x)\big)\frac{1}{N}$$

This defines an estimated best hypothesis:

$$\hat{h}^* = \operatorname*{argmin}_{h\in\mathcal{H}} EmpLoss_L(h)$$

Difference from the True Function

A learning problem is *realizable* if the *true function* f lies in the hypothesis space $\mathcal{H}$.

Difference from the True Function

A learning problem is *realizable* if the *true function* f lies in the hypothesis space $\mathcal{H}$.

The learned best hypothesis $\hat{h}^*$ may differ from f for several reasons:

♠ Unrealizability. No amount of data will recover f if $f \notin \mathcal{H}$.

Difference from the True Function

A learning problem is *realizable* if the *true function* f lies in the hypothesis space $\mathcal{H}$.

The learned best hypothesis $\hat{h}^*$ may differ from f for several reasons:

- ♠ Unrealizability. No amount of data will recover f if $f \notin \mathcal{H}$.
- ♠ Variance. Different hypotheses are returned from learning for different sets of examples.

Difference from the True Function

A learning problem is *realizable* if the *true function* f lies in the hypothesis space $\mathcal{H}$.

The learned best hypothesis $\hat{h}^*$ may differ from f for several reasons:

- ♠ Unrealizability. No amount of data will recover f if $f \notin \mathcal{H}$.
- ♠ Variance. Different hypotheses are returned from learning for different sets of examples.
- ♠ Noise. f may be nondeterministic or noisy, returning different values $f(x)$ for the same x (e.g., measurements of a person's height).

Difference from the True Function

A learning problem is *realizable* if the *true function* f lies in the hypothesis space $\mathcal{H}$.

The learned best hypothesis $\hat{h}^*$ may differ from f for several reasons:

- ♠ Unrealizability. No amount of data will recover f if $f \notin \mathcal{H}$.
- ♠ Variance. Different hypotheses are returned from learning for different sets of examples.
- ♠ Noise. f may be nondeterministic or noisy, returning different values $f(x)$ for the same x (e.g., measurements of a person's height).
- ♠ Computational complexity. Searching large hypothesis space $\mathcal{H}$ systematically can be computationally intractable. Often, a subspace search is conducted to return a reasonably good (but not optimal) hypothesis.

IV. Regularization

Model selection as a minimization:

$$\hat{h}^* = \operatorname*{argmin}_{h \in \mathcal{H}} \mathit{Cost}(h)$$

where

$$\mathit{Cost}(h) = \mathit{EmpLoss}(h) + \lambda \mathit{Complexity}(h)$$

IV. Regularization

Model selection as a minimization:

$$\hat{h}^* = \operatorname*{argmin}_{h \in \mathcal{H}} \mathit{Cost}\,(h)$$

where

$$\mathit{Cost}(h) = \mathit{EmpLoss}(h) + \lambda \mathit{Complexity}(h)$$

positive hyperparameter to balance
loss and hypothesis complexity

IV. Regularization

Model selection as a minimization:

$$\hat{h}^* = \operatorname*{argmin}_{h \in \mathcal{H}} \mathit{Cost}(h)$$

where

$$\mathit{Cost}(h) = \mathit{EmpLoss}(h) + \lambda \underbrace{\mathit{Complexity}(h)}_{\textit{regularization function}}$$

positive hyperparameter to balance loss and hypothesis complexity

regularization function

IV. Regularization

Model selection as a minimization:

$$\hat{h}^* = \operatorname*{argmin}_{h \in \mathcal{H}} Cost(h)$$

where

$$Cost(h) = EmpLoss(h) + \lambda \underbrace{Complexity(h)}_{\textit{regularization function}}$$

positive hyperparameter to balance loss and hypothesis complexity

regularization function

Regularization refers to the process of explicitly penalizing complex hypotheses to favor functions that are more regular.

IV. Regularization

Model selection as a minimization:

$$\hat{h}^* = \operatorname*{argmin}_{h \in \mathcal{H}} \mathit{Cost}(h)$$

where

$$\mathit{Cost}(h) = \mathit{EmpLoss}(h) + \lambda \underbrace{\mathit{Complexity}(h)}_{\textit{regularization function}}$$

positive hyperparameter to balance loss and hypothesis complexity

Regularization refers to the process of explicitly penalizing complex hypotheses to favor functions that are more regular.

- Choice of regularization function depends on the hypothesis space.

  For polynomials, we could use the sum of the squares of the coefficients, of which a small value would prevent a wiggling behavior.

Hyperparameter Tuning

It is difficult to choose good values for multiple hyperparameters via cross validation.

Hyperparameter Tuning

It is difficult to choose good values for multiple hyperparameters via cross validation.

- *Hand-tuning*. Guess some values, train a model, measure its performance, and then suggest new values.

Hyperparameter Tuning

It is difficult to choose good values for multiple hyperparameters via cross validation.

- *Hand-tuning*. Guess some values, train a model, measure its performance, and then suggest new values.

- *Grid search*. Try all combinations of values and choose one which performs the best on the validation data. Use random sampling if the size of the space is big or the values are continuous.

Hyperparameter Tuning

It is difficult to choose good values for multiple hyperparameters via cross validation.

- *Hand-tuning*. Guess some values, train a model, measure its performance, and then suggest new values.
- *Grid search*. Try all combinations of values and choose one which performs the best on the validation data. Use random sampling if the size of the space is big or the values are continuous.
- *Bayesian optimization*. Treat hyperparameter tuning as a machine learning problem itself.

Hyperparameter Tuning

It is difficult to choose good values for multiple hyperparameters via cross validation.

- *Hand-tuning*. Guess some values, train a model, measure its performance, and then suggest new values.

- *Grid search*. Try all combinations of values and choose one which performs the best on the validation data. Use random sampling if the size of the space is big or the values are continuous.

- *Bayesian optimization*. Treat hyperparameter tuning as a machine learning problem itself.

  Input: the vector $\boldsymbol{x}$ of hyperparameter values

  Output: the total loss y on the validation set for the resulting model

Hyperparameter Tuning

It is difficult to choose good values for multiple hyperparameters via cross validation.

- *Hand-tuning*. Guess some values, train a model, measure its performance, and then suggest new values.

- *Grid search*. Try all combinations of values and choose one which performs the best on the validation data. Use random sampling if the size of the space is big or the values are continuous.

- *Bayesian optimization*. Treat hyperparameter tuning as a machine learning problem itself.

  Input: the vector $\boldsymbol{x}$ of hyperparameter values

  Output: the total loss y on the validation set for the resulting model

  Training set: $(x_1, y_1), (x_2, y_2), \ldots, (x_n, y_N)$ generated from training runs

Hyperparameter Tuning

It is difficult to choose good values for multiple hyperparameters via cross validation.

- *Hand-tuning*. Guess some values, train a model, measure its performance, and then suggest new values.

- *Grid search*. Try all combinations of values and choose one which performs the best on the validation data. Use random sampling if the size of the space is big or the values are continuous.

- *Bayesian optimization*. Treat hyperparameter tuning as a machine learning problem itself.

  Input: the vector $\boldsymbol{x}$ of hyperparameter values

  Output: the total loss y on the validation set for the resulting model

  Training set: $(x_1, y_1), (x_2, y_2), \ldots, (x_n, y_N)$ generated from training runs

  Task: find a function $h(\boldsymbol{x})$ to minimize y over the N pairs $(y_i, h(\boldsymbol{x_i}))$