Modes of learning + Model selection

CPSC 532S: Modern Statistical Learning Theory
9 February 2022
cs.ubc.ca/~dsuth/532S/22/

Admin

- Hybrid mode starts next week, in DMP 101
- Office hours still online-only this week

- A2 is up, due next Friday night
  - Groups of up to three, allowed separate per question
  - If you don’t have a group and want one, post on Piazza

- A1 grading: hopefully done this week

An analogy about MDL

- Minimum description length says to pick $\operatorname{argmin}_{h \in \mathcal{H}} L_S(h) + f(|h|, n, \delta)$

An analogy about MDL

- Minimum description length says to pick $\operatorname{argmin}_{h \in \mathcal{H}} L_S(h) + f(|h|, n, \delta)$
- But $|h|$ isn't "inherent" to a function h

An analogy about MDL

- Minimum description length says to pick $\operatorname{argmin}_{h \in \mathcal{H}} L_S(h) + f(|h|, n, \delta)$
- But $|h|$ isn't "inherent" to a function h
  - Different ways to implement the same function

An analogy about MDL

- Minimum description length says to pick $\operatorname{argmin}_{h \in \mathcal{H}} L_S(h) + f(|h|, n, \delta)$
- But $|h|$ isn't "inherent" to a function h
  - Different ways to implement the same function
  - Can make up a language where any h you like has $|h| = 1$

An analogy about MDL

- Minimum description length says to pick $\operatorname{argmin}_{h \in \mathcal{H}} L_S(h) + f(|h|, n, \delta)$
- But $|h|$ isn't "inherent" to a function h
  - Different ways to implement the same function
  - Can make up a language where any h you like has $|h| = 1$

Loopholes that are forbidden by default

An analogy about MDL

- Minimum description length says to pick $\operatorname{argmin}_{h \in \mathcal{H}} L_S(h) + f(|h|, n, \delta)$
- But $|h|$ isn't "inherent" to a function h
  - Different ways to implement the same function
  - Can make up a language where any h you like has $|h| = 1$

Loopholes that are forbidden by default

313

Using a made-up language specifically designed for the challenge

This includes any language with commands that "do whatever I choose them to do". Claiming that your answer is written in "MyOwnLanguage", where the command `x` means "read a sequence of numbers, split them into groups of three, and print the last numbers of those groups where the second number is less than the first", was clever the first time it was done. That was a *long* time ago.

An analogy about MDL

- Minimum description length says to pick $\operatorname{argmin}_{h \in \mathcal{H}} L_S(h) + f(|h|, n, \delta)$
- But $|h|$ isn't "inherent" to a function h
  - Different ways to implement the same function
  - Can make up a language where any h you like has $|h| = 1$

HQ9+ is a joke language with four instructions:

- **H**: Print "hello, world"
- **Q**: Print the program's source code
- **9**: Print the lyrics to "99 Bottles of Beer"
- **+**: Increment the accumulator

Loopholes that are forbidden by default

313

Using a made-up language specifically designed for the challenge

This includes any language with commands that "do whatever I choose them to do". Claiming that your answer is written in "MyOwnLanguage", where the command `x` means "read a sequence of numbers, split them into groups of three, and print the last numbers of those groups where the second number is less than the first", was clever the first time it was done. That was a *long* time ago.

An analogy about MDL

- Minimum description length says to pick $\operatorname{argmin}_{h \in \mathcal{H}} L_S(h) + f(|h|, n, \delta)$
- But $|h|$ isn't "inherent" to a function h
  - Different ways to implement the same function
  - Can make up a language where any h you like has $|h| = 1$
  - But you have to do it *ahead of time*

HQ9+ is a joke language with four instructions:

- **H**: Print "hello, world"
- **Q**: Print the program's source code
- **9**: Print the lyrics to "99 Bottles of Beer"
- **+**: Increment the accumulator

Loopholes that are forbidden by default

313

Using a made-up language specifically designed for the challenge

This includes any language with commands that "do whatever I choose them to do". Claiming that your answer is written in "MyOwnLanguage", where the command `x` means "read a sequence of numbers, split them into groups of three, and print the last numbers of those groups where the second number is less than the first", was clever the first time it was done. That was a *long* time ago.

68

I don't have any problem with made-up languages. But the language should already exist when the challenge was posted. Never versions of the language are not allowed (to prevent changing the language e.g adding an extra command to HQ9+).
– Johannes Kuhn Mar 13 2014 at 15:27

An analogy about MDL

- Minimum description length says to pick $\operatorname{argmin}_{h \in \mathcal{H}} L_S(h) + f(|h|, n, \delta)$
- But $|h|$ isn't "inherent" to a function h
  - Different ways to implement the same function
  - Can make up a language where any h you like has $|h| = 1$
  - But you have to do it *ahead of time*

Loopholes that are forbidden by default

313

Using a made-up language specifically designed for the challenge

This includes any language with commands that "do whatever I choose them to do". Claiming that your answer is written in "MyOwnLanguage", where the command `x` means "read a sequence of numbers, split them into groups of three, and print the last numbers of those groups where the second number is less than the first", was clever the first time it was done. That was a *long* time ago.

HQ9+ is a joke language with four instructions:

- **H**: Print "hello, world"
- **Q**: Print the program's source code
- **9**: Print the lyrics to "99 Bottles of Beer"
- **+**: Increment the accumulator

Kolmogorov complexity

From Wikipedia, the free encyclopedia

In algorithmic information theory (a subfield of computer science and mathematics), the **Kolmogorov complexity** of an object, such as a piece of text, is the length of a shortest computer program (in a predetermined programming language) that produces the object as output. It is a

68

I don't have any problem with made-up languages. But the language should already exist when the challenge was posted. Never versions of the language are not allowed (to prevent changing the language e.g adding an extra command to HQ9+).
– Johannes Kuhn Mar 13 2014 at 15:27

Models of learnability

For fixed $\mathscr{H}$, binary classifiers, as $n \to \infty$:

$A(S)$ **competes with** h if, with high prob,

$L_{\mathscr{D}}(A(S)) \leq L_{\mathscr{D}}(h) + \varepsilon$ where $\varepsilon \overset{n\to\infty}{\longrightarrow} 0$

Models of learnability

$A(S)$ **competes with** h if, with high prob,
$L_{\mathscr{D}}(A(S)) \leq L_{\mathscr{D}}(h) + \varepsilon$ where $\varepsilon \xrightarrow{n\to\infty} 0$

For fixed $\mathscr{H}$, binary classifiers, as $n \to \infty$:

- Realizable PAC learners
  - Compete with any $h \in \mathscr{H}$ on any realizable $\mathscr{D}$; needed n depends only on ε and δ

Models of learnability

$A(S)$ **competes with** h if, with high prob,
$L_{\mathscr{D}}(A(S)) \leq L_{\mathscr{D}}(h) + \varepsilon$ where $\varepsilon \xrightarrow{n\to\infty} 0$

For fixed $\mathscr{H}$, binary classifiers, as $n \to \infty$:

- Realizable PAC learners
  - Compete with any $h \in \mathscr{H}$ on any realizable $\mathscr{D}$; needed n depends only on ε and δ
  - $\mathscr{H}$ realizably PAC learnable $\equiv$ ERM works on $\mathscr{H}$ for realizable $\mathscr{D} \equiv \mathrm{VCdim}(\mathscr{H}) < \infty$

Models of learnability

$A(S)$ **competes with** h if, with high prob,
$L_{\mathscr{D}}(A(S)) \leq L_{\mathscr{D}}(h) + \varepsilon$ where $\varepsilon \overset{n\to\infty}{\longrightarrow} 0$

For fixed $\mathscr{H}$, binary classifiers, as $n \to \infty$:

- Realizable PAC learners
  - Compete with any $h \in \mathscr{H}$ on any realizable $\mathscr{D}$; needed n depends only on ε and δ
  - $\mathscr{H}$ realizably PAC learnable $\equiv$ ERM works on $\mathscr{H}$ for realizable $\mathscr{D} \equiv \mathrm{VCdim}(\mathscr{H}) < \infty$
- Agnostic PAC learners
  - Compete with any $h \in \mathscr{H}$ on any $\mathscr{D}$; needed n depends only on ε and δ

Models of learnability

$A(S)$ **competes with** h if, with high prob,
$L_{\mathscr{D}}(A(S)) \leq L_{\mathscr{D}}(h) + \varepsilon$ where $\varepsilon \overset{n\to\infty}{\longrightarrow} 0$

For fixed $\mathscr{H}$, binary classifiers, as $n \to \infty$:

- Realizable PAC learners
  - Compete with any $h \in \mathscr{H}$ on any realizable $\mathscr{D}$; needed n depends only on ε and δ
  - $\mathscr{H}$ realizably PAC learnable $\equiv$ ERM works on $\mathscr{H}$ for realizable $\mathscr{D} \equiv \mathrm{VCdim}(\mathscr{H}) < \infty$
- Agnostic PAC learners
  - Compete with any $h \in \mathscr{H}$ on any $\mathscr{D}$; needed n depends only on ε and δ
  - $\mathscr{H}$ agnostically PAC learnable $\equiv$ ERM works on $\mathscr{H} \equiv \mathrm{VCdim}(\mathscr{H}) < \infty$

Models of learnability

$A(S)$ **competes with** h if, with high prob,
$L_{\mathscr{D}}(A(S)) \leq L_{\mathscr{D}}(h) + \varepsilon$ where $\varepsilon \xrightarrow{n\to\infty} 0$

For fixed $\mathscr{H}$, binary classifiers, as $n \to \infty$:

- Realizable PAC learners
  - Compete with any $h \in \mathscr{H}$ on any realizable $\mathscr{D}$; needed n depends only on ε and δ
  - $\mathscr{H}$ realizably PAC learnable $\equiv$ ERM works on $\mathscr{H}$ for realizable $\mathscr{D} \equiv \mathrm{VCdim}(\mathscr{H}) < \infty$
- Agnostic PAC learners
  - Compete with any $h \in \mathscr{H}$ on any $\mathscr{D}$; needed n depends only on ε and δ
  - $\mathscr{H}$ agnostically PAC learnable $\equiv$ ERM works on $\mathscr{H} \equiv \mathrm{VCdim}(\mathscr{H}) < \infty$
- Nonuniform learners
  - Compete with any $h \in \mathscr{H}$ on any $\mathscr{D}$; needed n depends on ε, δ, and h

Models of learnability

$A(S)$ **competes with** h if, with high prob,
$L_{\mathscr{D}}(A(S)) \leq L_{\mathscr{D}}(h) + \varepsilon$ where $\varepsilon \overset{n\to\infty}{\longrightarrow} 0$

For fixed $\mathscr{H}$, binary classifiers, as $n \to \infty$:

- Realizable PAC learners
  - Compete with any $h \in \mathscr{H}$ on any realizable $\mathscr{D}$; needed n depends only on ε and δ
  - $\mathscr{H}$ realizably PAC learnable $\equiv$ ERM works on $\mathscr{H}$ for realizable $\mathscr{D} \equiv \mathrm{VCdim}(\mathscr{H}) < \infty$
- Agnostic PAC learners
  - Compete with any $h \in \mathscr{H}$ on any $\mathscr{D}$; needed n depends only on ε and δ
  - $\mathscr{H}$ agnostically PAC learnable $\equiv$ ERM works on $\mathscr{H} \equiv \mathrm{VCdim}(\mathscr{H}) < \infty$
- Nonuniform learners
  - Compete with any $h \in \mathscr{H}$ on any $\mathscr{D}$; needed n depends on ε, δ, and h
  - $\mathscr{H}$ nonuniformly learnable $\equiv$ SRM works on $\mathscr{H} \equiv \mathscr{H}$ is countable union of finite-VC sets

Models of learnability

$A(S)$ **competes with** h if, with high prob,
$L_{\mathscr{D}}(A(S)) \leq L_{\mathscr{D}}(h) + \varepsilon$ where $\varepsilon \xrightarrow{n\to\infty} 0$

For fixed $\mathscr{H}$, binary classifiers, as $n \to \infty$:

- Realizable PAC learners
  - Compete with any $h \in \mathscr{H}$ on any realizable $\mathscr{D}$; needed n depends only on ε and δ
  - $\mathscr{H}$ realizably PAC learnable $\equiv$ ERM works on $\mathscr{H}$ for realizable $\mathscr{D} \equiv \mathrm{VCdim}(\mathscr{H}) < \infty$
- Agnostic PAC learners
  - Compete with any $h \in \mathscr{H}$ on any $\mathscr{D}$; needed n depends only on ε and δ
  - $\mathscr{H}$ agnostically PAC learnable $\equiv$ ERM works on $\mathscr{H} \equiv \mathrm{VCdim}(\mathscr{H}) < \infty$
- Nonuniform learners
  - Compete with any $h \in \mathscr{H}$ on any $\mathscr{D}$; needed n depends on ε, δ, and h
  - $\mathscr{H}$ nonuniformly learnable $\equiv$ SRM works on $\mathscr{H} \equiv \mathscr{H}$ is countable union of finite-VC sets
- New today: **consistency**

Models of learnability

$A(S)$ **competes with** h if, with high prob,
$L_{\mathscr{D}}(A(S)) \leq L_{\mathscr{D}}(h) + \varepsilon$ where $\varepsilon \overset{n\to\infty}{\longrightarrow} 0$

For fixed $\mathscr{H}$, binary classifiers, as $n \to \infty$:

- Realizable PAC learners
  - Compete with any $h \in \mathscr{H}$ on any realizable $\mathscr{D}$; needed n depends only on ε and δ
  - $\mathscr{H}$ realizably PAC learnable $\equiv$ ERM works on $\mathscr{H}$ for realizable $\mathscr{D} \equiv \mathrm{VCdim}(\mathscr{H}) < \infty$
- Agnostic PAC learners
  - Compete with any $h \in \mathscr{H}$ on any $\mathscr{D}$; needed n depends only on ε and δ
  - $\mathscr{H}$ agnostically PAC learnable $\equiv$ ERM works on $\mathscr{H} \equiv \mathrm{VCdim}(\mathscr{H}) < \infty$
- Nonuniform learners
  - Compete with any $h \in \mathscr{H}$ on any $\mathscr{D}$; needed n depends on ε, δ, and h
  - $\mathscr{H}$ nonuniformly learnable $\equiv$ SRM works on $\mathscr{H} \equiv \mathscr{H}$ is countable union of finite-VC sets
- New today: **consistency**
  - Compete with any $h \in \mathscr{H}$ on any $\mathscr{D}$; needed n depends on ε, δ, h, and $\mathscr{D}$

Consistency

- Let $\mathscr{P}$ be some set of distributions over our domain $\mathscr{Z}$

Consistency

- Let $\mathscr{P}$ be some set of distributions over our domain $\mathscr{Z}$
- A learning algorithm A is **consistent** w.r.t. $\mathscr{H}$ and $\mathscr{P}$ if

Consistency

- Let $\mathscr{P}$ be some set of distributions over our domain $\mathscr{Z}$
- A learning algorithm A is **consistent** w.r.t. $\mathscr{H}$ and $\mathscr{P}$ if
  - for every $\varepsilon, \delta > 0, h \in \mathscr{H}, \mathscr{D} \in \mathscr{P}$,

Consistency

- Let $\mathscr{P}$ be some set of distributions over our domain $\mathscr{Z}$
- A learning algorithm A is **consistent** w.r.t. $\mathscr{H}$ and $\mathscr{P}$ if
  - for every $\varepsilon, \delta > 0$, $h \in \mathscr{H}$, $\mathscr{D} \in \mathscr{P}$,
  - as long as n is above some threshold $n_{\mathscr{H}}^{CON}(\varepsilon, \delta, h, \mathscr{D})$,

Consistency

- Let $\mathscr{P}$ be some set of distributions over our domain $\mathscr{Z}$
- A learning algorithm A is **consistent** w.r.t. $\mathscr{H}$ and $\mathscr{P}$ if
  - for every $\varepsilon, \delta > 0, h \in \mathscr{H}, \mathscr{D} \in \mathscr{P}$,
  - as long as n is above some threshold $n_{\mathscr{H}}^{CON}(\varepsilon, \delta, h, \mathscr{D})$,
  - we have $\Pr_{S \sim \mathscr{D}^n} \left(L_{\mathscr{D}}(A(S)) \leq L_{\mathscr{D}}(h) + \varepsilon \right) \geq 1 - \delta$

Consistency

- Let $\mathscr{P}$ be some set of distributions over our domain $\mathscr{Z}$
- A learning algorithm A is **consistent** w.r.t. $\mathscr{H}$ and $\mathscr{P}$ if
  - for every $\varepsilon, \delta > 0, h \in \mathscr{H}, \mathscr{D} \in \mathscr{P}$,
  - as long as n is above some threshold $n_{\mathscr{H}}^{CON}(\varepsilon, \delta, h, \mathscr{D})$,
  - we have $\Pr_{S \sim \mathscr{D}^n} \left(L_{\mathscr{D}}(A(S)) \leq L_{\mathscr{D}}(h) + \varepsilon \right) \geq 1 - \delta$

- Means that A eventually works as well as any hypothesis in $\mathscr{H}$, if the truth is anything in $\mathscr{P}$

Consistency

- Let $\mathcal{P}$ be some set of distributions over our domain $\mathcal{Z}$
- A learning algorithm A is **consistent** w.r.t. $\mathcal{H}$ and $\mathcal{P}$ if
  - for every $\varepsilon, \delta > 0, h \in \mathcal{H}, \mathcal{D} \in \mathcal{P}$,
  - as long as n is above some threshold $n_{\mathcal{H}}^{CON}(\varepsilon, \delta, h, \mathcal{D})$,
  - we have $\Pr_{S \sim \mathcal{D}^n} \left(L_{\mathcal{D}}(A(S)) \leq L_{\mathcal{D}}(h) + \varepsilon \right) \geq 1 - \delta$

- Means that A eventually works as well as any hypothesis in $\mathcal{H}$, if the truth is anything in $\mathcal{P}$

- A is **universally consistent** w.r.t. $\mathcal{H}$ if it is consistent wrt $\mathcal{H}$ for the set of all distributions on $\mathcal{Z}$

Memorization

- A silly learning algorithm:
  - To train: just save $S = \big((x_1, y_1), \ldots, (x_n, y_n)\big)$
  - To predict on x: if $x = x_i$, return y_i, else return 0

$x \in S|_x$, return mode $\{y_i : x = x_i\}$

- This is **universally consistent** for binary 0-1 loss wrt the set of **all** binary predictors, if $\mathcal{X}$ is countable

Memorization

- A silly learning algorithm:
  - To train: just save $S = \big((x_1, y_1), \ldots, (x_n, y_n)\big)$
  - To predict on x: if $x = x_i$, return y_i, else return 0

- This is **universally consistent** for binary 0-1 loss wrt the set of **all** binary predictors, if $\mathcal{X}$ is countable
  - Example of countable $\mathcal{X}$: $\{0,1\}^*$, the set of binary strings of any length

Memorization

- A silly learning algorithm:
  - To train: just save $S = \big((x_1, y_1), \ldots, (x_n, y_n)\big)$
  - To predict on x: if $x = x_i$, return y_i, else return 0

- This is **universally consistent** for binary 0-1 loss wrt the set of **all** binary predictors, if $\mathcal{X}$ is countable
  - Example of countable $\mathcal{X}$: $\{0,1\}^*$, the set of binary strings of any length
  - Proof: SSBD exercise 6.6

Memorization

- A silly learning algorithm:
  - To train: just save $S = \big((x_1, y_1), \ldots, (x_n, y_n)\big)$
  - To predict on x: if $x = x_i$, return y_i, else return 0

- This is **universally consistent** for binary 0-1 loss wrt the set of **all** binary predictors, if $\mathcal{X}$ is countable
  - Example of countable $\mathcal{X}$: $\{0,1\}^*$, the set of binary strings of any length
  - Proof: SSBD exercise 6.6

- Making an inconsistent algorithm A consistent:

Memorization

- A silly learning algorithm:
  - To train: just save $S = \big((x_1, y_1), \ldots, (x_n, y_n)\big)$
  - To predict on x: if $x = x_i$, return y_i, else return 0

- This is **universally consistent** for binary 0-1 loss wrt the set of **all** binary predictors, if $\mathcal{X}$ is countable
  - Example of countable $\mathcal{X}$: $\{0,1\}^*$, the set of binary strings of any length
  - Proof: SSBD exercise 6.6

- Making an inconsistent algorithm A consistent:
  - To train: save $S = \big((x_1, y_1), \ldots, (x_n, y_n)\big)$ and learn $h = A(S)$

Memorization

- A silly learning algorithm:
  - To train: just save $S = \big((x_1, y_1), \ldots, (x_n, y_n)\big)$
  - To predict on x: if $x = x_i$, return y_i, else return 0

- This is **universally consistent** for binary 0-1 loss wrt the set of **all** binary predictors, if $\mathcal{X}$ is countable
  - Example of countable $\mathcal{X}$: $\{0,1\}^*$, the set of binary strings of any length
  - Proof: SSBD exercise 6.6

- Making an inconsistent algorithm A consistent:
  - To train: save $S = \big((x_1, y_1), \ldots, (x_n, y_n)\big)$ and learn $h = A(S)$
  - To predict on x: if $x = x_i$, return y_i, else return $h(x)$

Memorization

- A silly learning algorithm:
  - To train: just save $S = \big((x_1, y_1), \ldots, (x_n, y_n)\big)$
  - To predict on x: if $x = x_i$, return y_i, else return 0

- This is **universally consistent** for binary 0-1 loss wrt the set of **all** binary predictors, if $\mathscr{X}$ is countable
  - Example of countable $\mathscr{X}$: $\{0,1\}^*$, the set of binary strings of any length
  - Proof: SSBD exercise 6.6

- Making an inconsistent algorithm A consistent:
  - To train: save $S = \big((x_1, y_1), \ldots, (x_n, y_n)\big)$ and learn $h = A(S)$
  - To predict on x: if $x = x_i$, return y_i, else return $h(x)$
- **Even so**, can be interesting to compare $n_{\mathscr{H}}^{CON}$ for different sub-classes of $\mathscr{D}$

Is memorizing your lunch enough?

- Memorize is universally consistent w.r.t. *all* binary predictors for countable $\mathscr{X}$
  - Implies: for any pattern and $\mathscr{D}$, *eventually* we will learn that pattern
  - But how many examples you need depends on nature's underlying pattern

- No free lunch says:

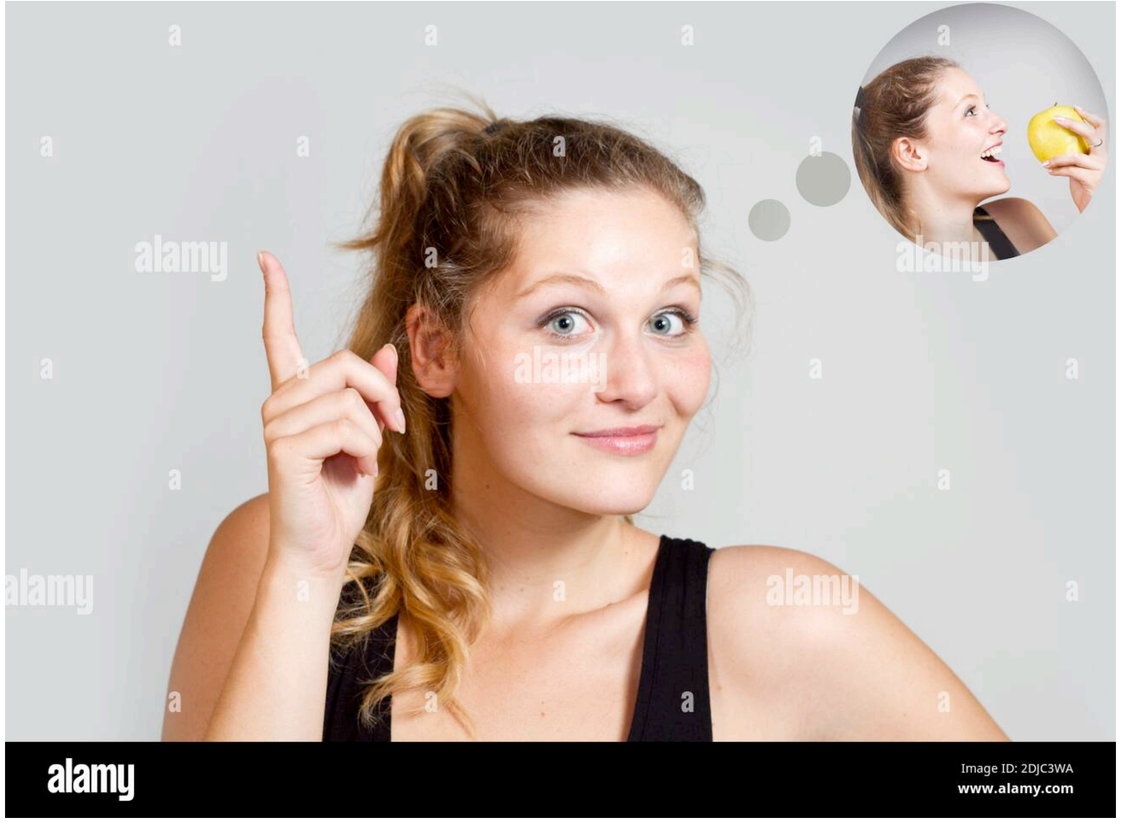
  For any algorithm
  and any training set size n,
  there is a $\mathscr{D}$ where the algorithm fails
  (when using n samples)

Is memorizing your lunch enough?

- Memorize is universally consistent w.r.t. *all* binary predictors for countable $\mathscr{X}$
  - Implies: for any pattern and $\mathscr{D}$, *eventually* we will learn that pattern
  - But how many examples you need depends on nature's underlying pattern

- No free lunch says:

  For any algorithm
  and any training set size n,
  there is a $\mathscr{D}$ where the algorithm fails
  (when using n samples)



Goals of studying learnability

- How good is my particular learned h?
  - Usual best answer (next, today): check a validation set

Goals of studying learnability

- How good is my particular learned h?
  - Usual best answer (next, today): check a validation set

- How many samples will I need to be as good as anything in $\mathscr{H}$?
  - PAC-type bounds can give good (if conservative) answers
  - Nonuniform learning, consistency don't, since we don't know h^*

Goals of studying learnability

- How good is my particular learned h?
  - Usual best answer (next, today): check a validation set

- How many samples will I need to be as good as anything in $\mathscr{H}$?
  - PAC-type bounds can give good (if conservative) answers
  - Nonuniform learning, consistency don't, since we don't know h^*

- What's learnable, and what's inherently hard to learn? What principles work?
  - PAC-type bounds can be more informative
  - Plain "consistent or not" doesn't say much

Goals of studying learnability

- How good is my particular learned h?
  - Usual best answer (next, today): check a validation set

- How many samples will I need to be as good as anything in $\mathscr{H}$?
  - PAC-type bounds can give good (if conservative) answers
  - Nonuniform learning, consistency don't, since we don't know h^*

- What's learnable, and what's inherently hard to learn? What principles work?
  - PAC-type bounds can be more informative
  - Plain "consistent or not" doesn't say much

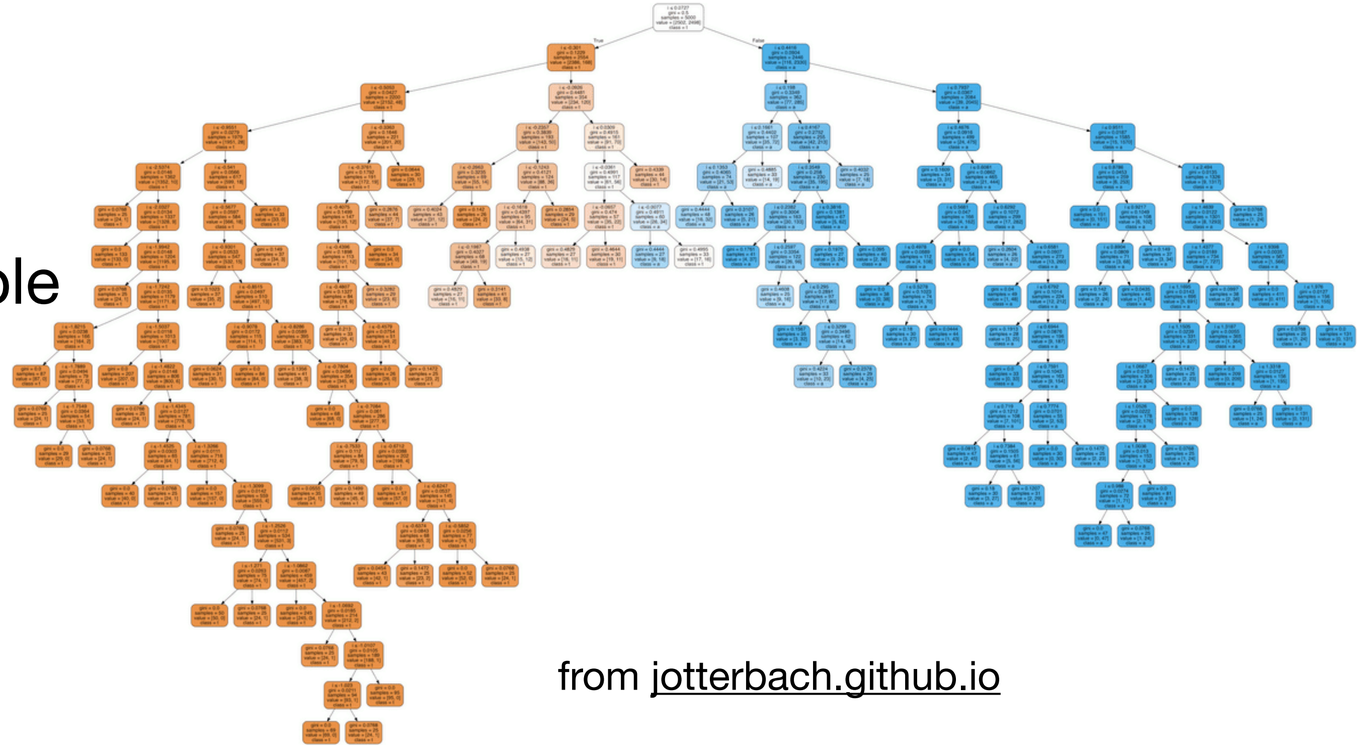
- Lots of work (unlike SSBD) ask about learning *given some assumptions on* $\mathscr{D}$
  - Of SSBD's models of learning, only consistency really allows this

(pause)

Model selection in practice

- Say we want to fit a decision tree,
  and are not sure how deep we should let it be
- Last time: pick $\mathscr{H}_k$ as depth-k trees, run SRM
  - But still need to pick weights…
  - MDL is one way
  - Not always best
  - Bounds can be conservative
  - Not always computationally feasible

- What if we want to fit
  either a decision tree
  or a linear classifier
  or a neural network
  or…?



from jotterbach.github.io

The CPSC 340 solution (i.e., what people actually do)

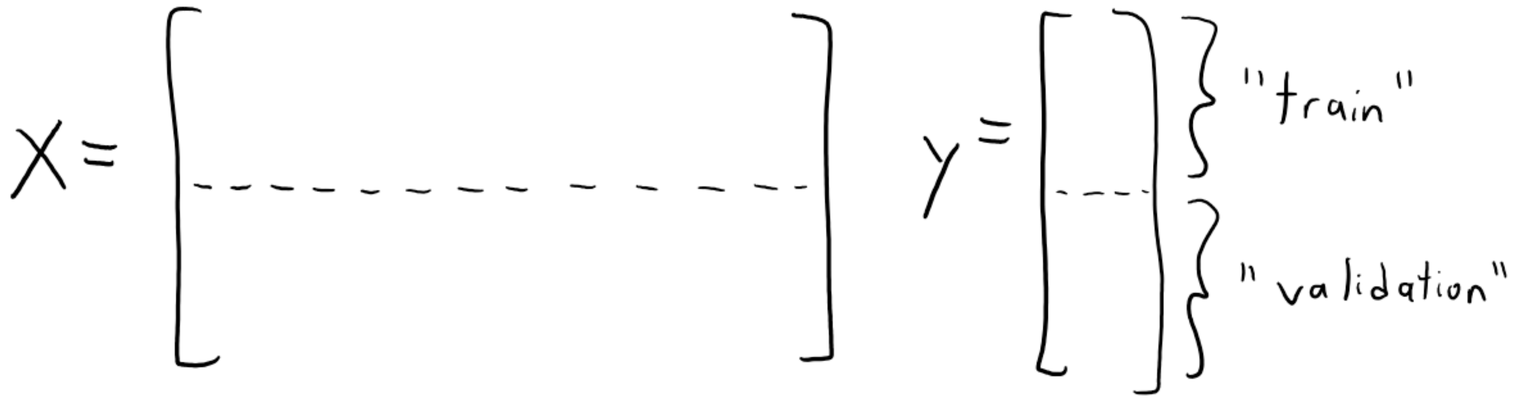
Validation Error

- How do we decide decision tree depth?
- We care about test error.
- But we can't look at test data.
- So what do we do?????

- One answer: Use part of the training data to approximate test error.
- Split training examples into training set and validation set:
  - Train model based on the training data. $L_S(h)$
  - Test model based on the validation data. $L_V(h)$

The CPSC 340 solution (i.e., what people actually do)

Validation Error



Step 1 is training: model = train(X_{train}, y_{train}) e.g. $\text{argmin}_h L_S(h)$

Step 2 is predicting: $\hat{y}$ = predict(model, $X_{validate}$)

Step 3 is validating: error = sum($\hat{y} \neq y_{validate}$) $L_V(h)$

Note: if examples are ordered, split should be random.

$\mathcal{H} = \mathcal{H}_1 \cup \mathcal{H}_2 \cup \mathcal{H}_3 \cup \ldots$

The CPSC 340 solution (i.e., what people actually do)

Notation: Parameters and Hyper-Parameters

- The decision tree rule values are called "parameters".
  - Parameters control how well we fit a dataset.
  - We "train" a model by trying to find the best parameters on training data.

- The decision tree depth is a called a "hyper-parameter".
  - Hyper-parameters control how complex our model is.
  - We can't "train" a hyper-parameter.
    - You can always fit training data better by making the model more complicated.
  - We "validate" a hyper-parameter using a validation score.

- ("Hyper-parameter" is sometimes used for parameters "not fit with data".)

The CPSC 340 solution (i.e., what people actually do)

Choosing Hyper-Parameters with Validation Set

- So to choose a good value of depth ("hyper-parameter"), we could:
  - Try a depth-1 decision tree, compute validation error. $\hat{h}_1 = \text{ERM}_{\mathcal{H}_1}(S)$
  - Try a depth-2 decision tree, compute validation error. $\hat{h}_2 = \text{ERM}_{\mathcal{H}_2}(S)$
  - Try a depth-3 decision tree, compute validation error. $\hat{h}_3 = \text{ERM}_{\mathcal{H}_3}(S)$
  - …
  - Try a depth-20 decision tree, compute validation error. $\hat{h}_{20} = \text{ERM}_{\mathcal{H}_{20}}(S)$
  - Return the depth with the lowest validation error.

$$k = \text{argmin}_k \, L_V(\hat{h}_k)$$

- After you choose the hyper-parameter, we usually re-train on the full training set with the chosen hyper-parameter. $\hat{h} = \text{ERM}_{\mathcal{H}_k}(S \cup V)$

or just take $\text{argmin}_k \, \hat{h}_k$

The CPSC 340 solution (i.e., what people actually do)

Digression: Optimization Bias

- Another name for overfitting is "optimization bias":
  - How biased is an "error" that we optimized over many possibilities?

- Optimization bias of parameter learning: (all of $\mathscr{H}$)
  - During learning, we could search over tons of different decision trees.
  - So we can get "lucky" and find one with low training error by chance.
    - "Overfitting of the training error".

- Optimization bias of hyper-parameter tuning: just $\{\hat{h}_1, \hat{h}_2, \ldots, \hat{h}_{20}\}$
  - Here, we might optimize the validation error over 20 values of "depth".
  - One of the 20 trees might have low validation error by chance.
    - "Overfitting of the validation error".

The CPSC 340 solution (i.e., what people actually do)

Digression: Example of Optimization Bias

- Consider a multiple-choice (a,b,c,d) "test" with 10 questions:
  - If you choose answers randomly, expected grade is 25% (no bias).
  - If you fill out two tests randomly and pick the best, expected grade is 33%.
    - Optimization bias of ~8%.
  - If you take the best among 10 random tests, expected grade is ~47%.
  - If you take the best among 100, expected grade is ~62%.
  - If you take the best among 1,000, expected grade is ~73%.
  - If you take the best among 10,000, expected grade is ~82%.
    - You have so many "chances" that you expect to do well.
- But on new questions the "random choice" accuracy is still 25%.

$\inf_{h \in \mathcal{H}} L_S(h)$ shrinks with $\mathcal{H}$

here $L_{\mathcal{D}}(h)$ is constant

so $\sup L_{\mathcal{D}}(h) - L_S(h)$ grows

The CPSC 340 solution (i.e., what people actually do)

Overfitting to the Validation Set?

- Validation error usually has lower optimization bias than training error.
  - Might optimize over 20 values of "depth", instead of millions+ of possible trees.
- But we can still overfit to the validation error (common in practice):
  - Validation error is only an unbiased approximation if you use it once.
  - Once you start optimizing it, you start to overfit to the validation set.
- This is most important when the validation set is "small":
  - The optimization bias decreases as the number of validation examples increases.
- Remember, our goal is still to do well on the test set (new data), not the validation set (where we already know the labels).

The CPSC 340 thought process

Should you trust them?

- Scenario 1:
  - "I built a model based on the data you gave me."
  - "It classified your data with 98% accuracy."
  - "It should get 98% accuracy on the rest of your data."

- Probably not:
  - They are reporting training error.
  - This might have nothing to do with test error.
  - E.g., they could have fit a very deep decision tree.
- Why 'probably'?
  - If they only tried a few very simple models, the 98% might be reliable.
  - E.g., they only considered decision stumps with simple 1-variable rules.

Should you trust them?

- Scenario 2:
  - "I built a model based on half of the data you gave me."
  - "It classified the other half of the data with 98% accuracy."
  - "It should get 98% accuracy on the rest of your data."

- Probably:
  - They computed the validation error once.
  - This is an unbiased approximation of the test error.
  - Trust them if you believe they didn't violate the golden rule.

Should you trust them?

- Scenario 3:
  - "I built 10 models based on half of the data you gave me."
  - "One of them classified the other half of the data with 98% accuracy."
  - "It should get 98% accuracy on the rest of your data."

- Probably:
  - They computed the validation error a small number of times.
  - Maximizing over these errors is a biased approximation of test error.
  - But they only maximized it over 10 models, so bias is probably small.
  - They probably know about the golden rule.

Should you trust them?

- Scenario 4:
  - "I built 1 billion models based on half of the data you gave me."
  - "One of them classified the other half of the data with 98% accuracy."
  - "It should get 98% accuracy on the rest of your data."

- Probably not:
  - They computed the validation error a huge number of times.
  - They tried so many models, one of them is likely to work by chance.
- Why 'probably'?
  - If the 1 billion models were all *extremely* simple, 98% might be reliable.

Should you trust them?

- Scenario 5:
  - "I built 1 billion models based on the first third of the data you gave me."
  - "One of them classified the second third of the data with 98% accuracy."
  - "It also classified the last third of the data with 98% accuracy."
  - "It should get 98% accuracy on the rest of your data."

- Probably:
  - They computed the first validation error a huge number of times.
  - But they had a second validation set that they only looked at once.
  - The second validation set gives unbiased test error approximation.
  - This is ideal, as long as they didn't violate golden rule on the last third.
  - And assuming you are using IID data in the first place.

Hold-out set accuracy, quantitatively

- If V is independent of choosing h, for loss in $[0,B]$, immediately have

$$\left| L_V(h) - L_{\mathscr{D}}(h) \right| \leq B\sqrt{\frac{1}{2|V|}\log\frac{2}{\delta}}$$

  - No dependence on $\mathscr{H}$, A, etc; doesn't matter how we picked h

Hold-out set accuracy, quantitatively

- If V is independent of choosing h, for loss in $[0,B]$, immediately have

$$\left| L_V(h) - L_{\mathscr{D}}(h) \right| \leq B\sqrt{\frac{1}{2|V|}\log\frac{2}{\delta}}$$

  - No dependence on $\mathscr{H}$, A, etc; doesn't matter how we picked h

- Find $\hat{\mathscr{H}} = \{\hat{h}_1, \ldots, \hat{h}_{|\hat{\mathscr{H}}|}\}$ based on a set S. For an independent V,

$$\left| L_V(h) - L_{\mathscr{D}}(h) \right| \leq B\sqrt{\frac{1}{2|V|}\log\frac{2|\hat{\mathscr{H}}|}{\delta}}$$

sup
$h \in \hat{\mathscr{H}}$

Validation set vs SRM

- Say we have $|S| = n$ and $|V| = m$
- Decompose $\mathscr{H}$ into $\mathscr{H}_1 \cup \mathscr{H}_2 \cup \cdots$; consider SRM with weights $6/(\pi^2 k^2)$
- $\text{Val}(S, V)$ takes $\text{argmin}_{h \in \{\text{ERM}_{\mathscr{H}_k}(S)\}} L_V(h)$

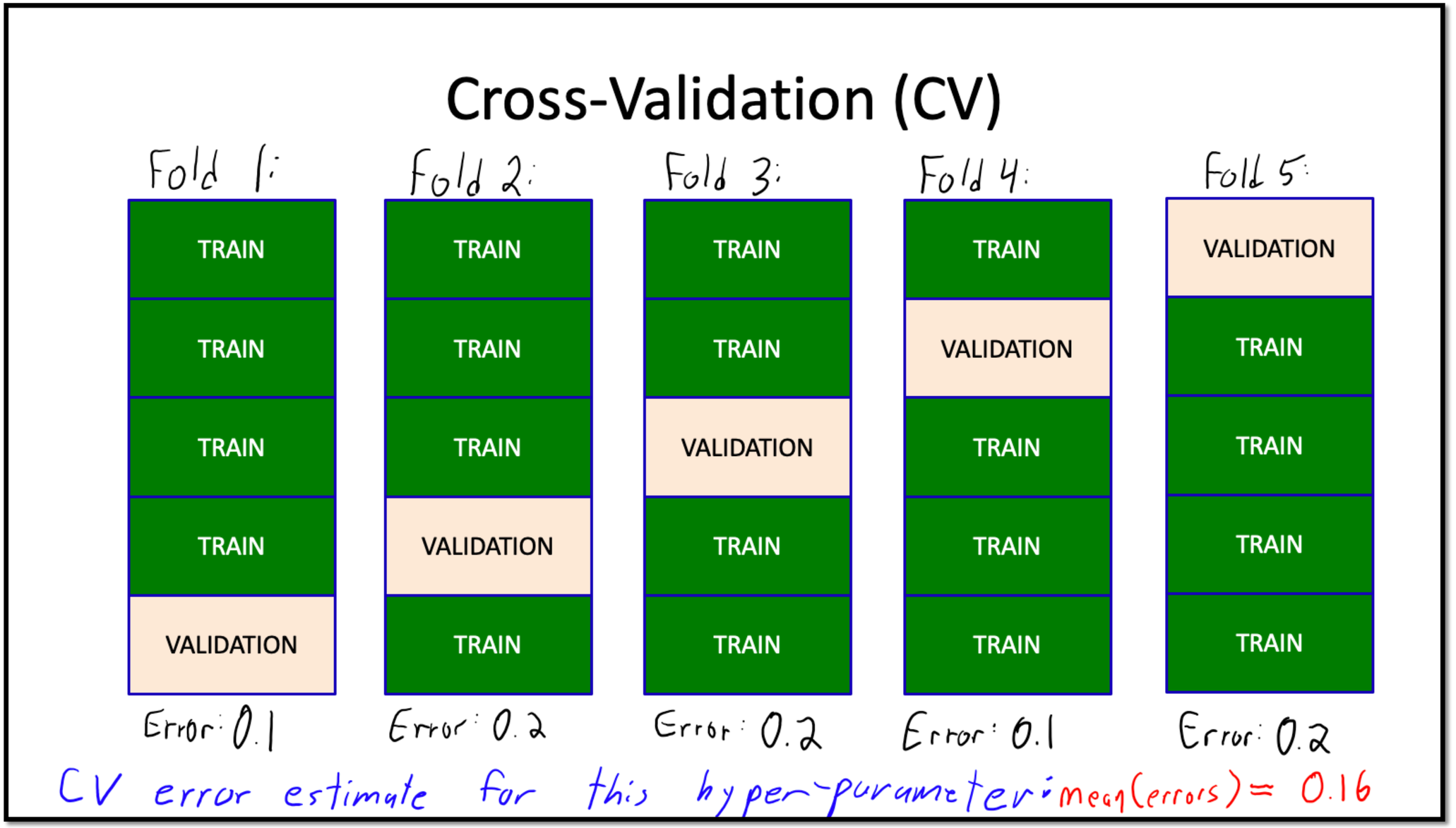
- Can show (MRT prop 4.3: union bound + Hoeffding) that

$$\left| L_{\mathscr{D}}(\text{ERM}_{\mathscr{H}_k}(S)) - L_V(\text{ERM}_{\mathscr{H}_k}(S)) \right| \leq \sqrt{\frac{1}{2m} \log \frac{4}{\delta}} + \sqrt{\frac{1}{m} \log k}$$

- Implies (MRT thm 4.4)

$$L_{\mathscr{D}}(\text{Val}(S, V)) - L_{\mathscr{D}}(\text{SRM}(S)) \leq 2\sqrt{\frac{1}{m} \log \max(k_{\text{Val}(S,V)}, k_{\text{SRM}(S)})} + 2\sqrt{\frac{1}{2m} \log \frac{4}{\delta}}$$

  - i.e. not too much worse than SRM with *part* of the data



We'll come back to analyze cross-validation soon (based on *stability* analyses)

Summary

- Models of learnability
  - Realizable PAC, Agnostic PAC
  - Nonuniform learning
  - Consistency
    - Silly to talk about whether consistent or not,
      but comparing sample complexities makes sense

- Picking models in practice
  - Validation sets
  - Simple bounds based on how many times you look at the val set
    - Hoeffding + union bound
  - Can prove: not too much worse than SRM (which ignores the val set)
  - More practical than SRM, usually

Massart's lemma: for $\mathcal{A} \subset \mathbb{R}^n$, if $\max_{a \in \mathcal{A}} \|a\| \le r$, $\quad \mathbb{E}_\sigma\left[\max_{a \in \mathcal{A}} \frac{1}{n} \boldsymbol{\sigma}^\top a\right] \le \frac{1}{n} r \sqrt{2 \log |\mathcal{A}|}$

$$\exp\left(\lambda \, \mathbb{E}_\sigma \max_a \sigma^\top a\right) \le \mathbb{E}_\sigma \exp\left(\lambda \max_a \sigma^\top a\right)$$

$$= \mathbb{E}_\sigma \max_a \exp(\lambda \, \sigma^\top a) \qquad \leftarrow \text{all} \ge 0$$

$$\le \mathbb{E}_\sigma \sum_a \exp(\lambda \, \sigma^\top a)$$

$$= \sum_a \mathbb{E}_\sigma \exp\left(\lambda \textstyle\sum_i \sigma_i a_i\right)$$

$$= \sum_a \mathbb{E}_\sigma \prod_{i=1}^n \exp(\lambda \sigma_i a_i)$$

$$= \sum_a \prod_i \frac{1}{2}\left(e^{\lambda a_i} + e^{-\lambda a_i}\right)$$

$$\le \sum_a \prod_i e^{\lambda^2 a_i^2/2}$$

$$= \sum_a e^{\lambda^2 \|a\|^2/2}$$

$$\le \sum_a e^{\lambda^2 r^2/2} = |\mathcal{A}| \, e^{\lambda^2 r^2/2}$$

$\mathcal{H}_S = \{(h(z_1), \ldots, h(z_n)) : h \in \mathcal{H}\} \subseteq \mathbb{R}^n$

$\hat{\mathcal{R}}_S(\mathcal{H}) = \mathbb{E}_\sigma \max_{h_S \in \mathcal{H}_S} \frac{1}{n} \sigma^\top h_S$

$\max\{1, 2, 0.5, 3\} \ll 1 + 2 + 0.5 + 3$

but $\max\{1, 2, 1000, 3\} \approx 1 + 20 + 1000 + 2$

$$\frac{e^x + e^{-x}}{2} \le e^{\frac{1}{2}x^2}$$

$$\le \left(\frac{1}{\sqrt{2}} + \frac{\sqrt{2}}{2}\right) r \sqrt{\log |\mathcal{A}|}$$

done!

$$\therefore \mathbb{E}_\sigma \max_a \sigma^\top a \le \frac{1}{\lambda} \log |\mathcal{A}| + \lambda \frac{r^2}{2}$$

$$\mathbb{E}_\sigma \max_a \sigma^\top a \le \inf_{\lambda > 0} \frac{1}{\lambda} \log |\mathcal{A}| + \lambda \frac{r^2}{2}$$

$$\log |\mathcal{A}| = \frac{r^2}{2} \lambda^2$$

$$\lambda = \frac{1}{r} \sqrt{2 \log |\mathcal{A}|}$$