CPSC 340:
Machine Learning and Data Mining

Probabilistic Classification

Summer 2021

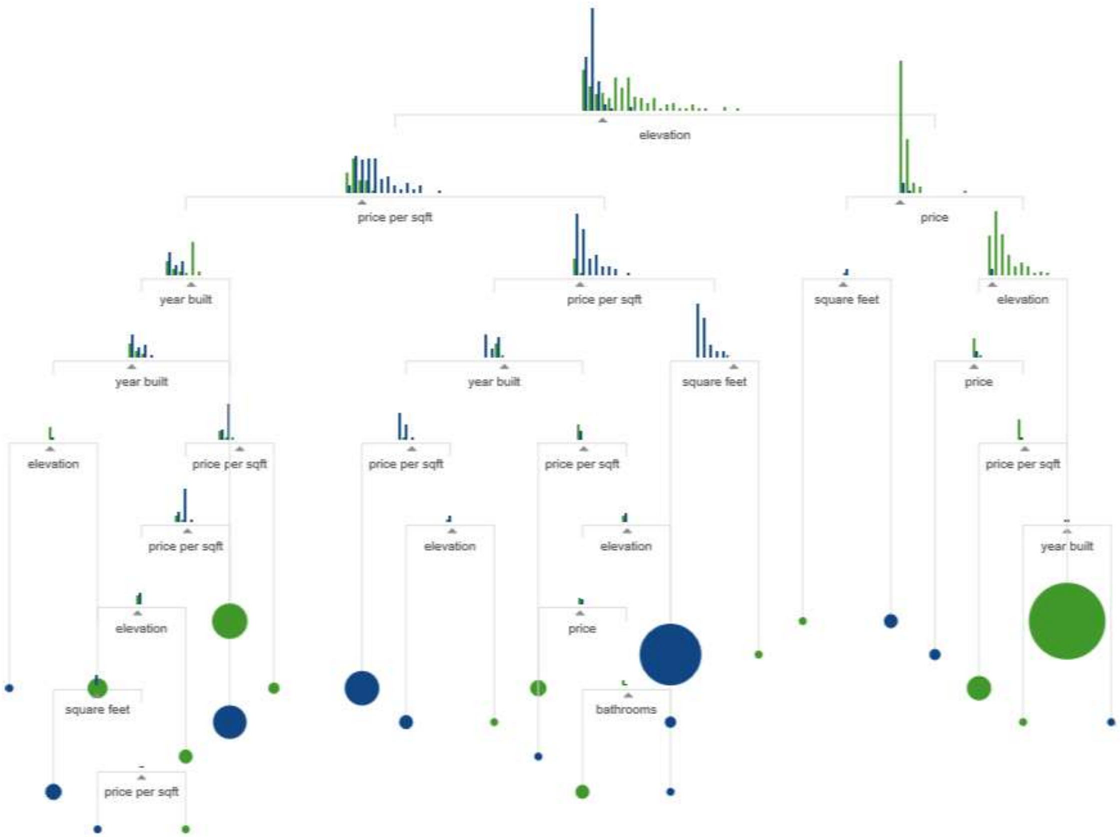
Admin

- Monday:
  - Assignment 1 due
  - Assignment 2 out, due the following Monday
- Next Friday: Assignment 3 out
  - Due the following Friday
  - To make enough time for you to study for midterm
- Midterm will be Tuesday, June 1, 2021
  - Canvas for autograded portion
  - Gradescope for manually graded portion
  - Stay tuned for instructions
- Piazza: partner search post is up.
  - See my recommendations for teamwork.
- Contact us on Piazza if you need help with Gradescope.

In This Lecture

- More on Optimization Bias (10 minutes)
- Cross-Validation (10 minutes)
- “Best” Machine Learning Model (10 minutes)
- Naïve Bayes (20 minutes)

Last Time: Decision Trees

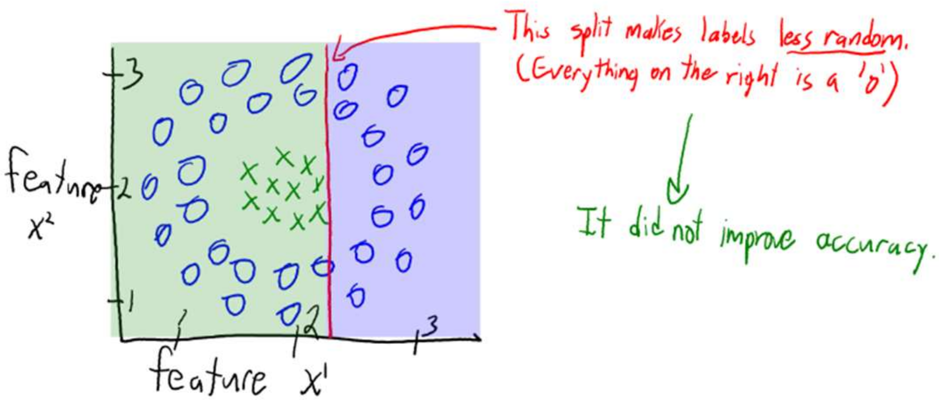


Clarification: Score

- Be careful about how scores are implemented in code.
  - Maximizing accuracy = Minimizing _____________
  - We want to (maximize/minimize) information gain
  - Baseline accuracy is ________________________.
  - Baseline information gain is _____.

Clarification: Baseline

Example Where Accuracy Fails



- Recall: my baseline is return-the-mode.
- When searching for a decision stump with accuracy score, we should try to beat the baseline, not "accuracy=0"
- Using "accuracy=0" as baseline, you will get a different behaviour.
  - E.g. GRS will actually continue splitting, since we get accuracy > 0 from above split.

Last Time: Training, Testing, and Validation

- Training step:

  Input: set of 'n' training examples x_i with labels y_i

  Output: a model that maps from arbitrary x_i to a $\hat{y}_i$

- Prediction step:

  Input: set of 't' testing examples $\tilde{x}_i$ and a model.

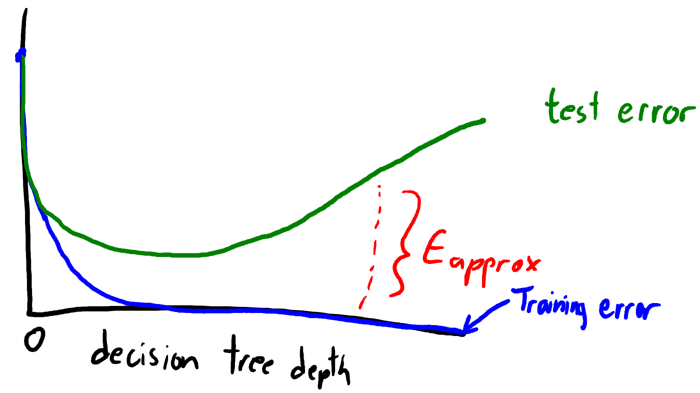
  Output: predictions $\hat{y}_i$ for the testing examples.

- What we are interested in is the test error:
  - Error made by prediction step on new data.

Last Time: Fundamental Trade-Off

- We decomposed test error to get a fundamental trade-off:

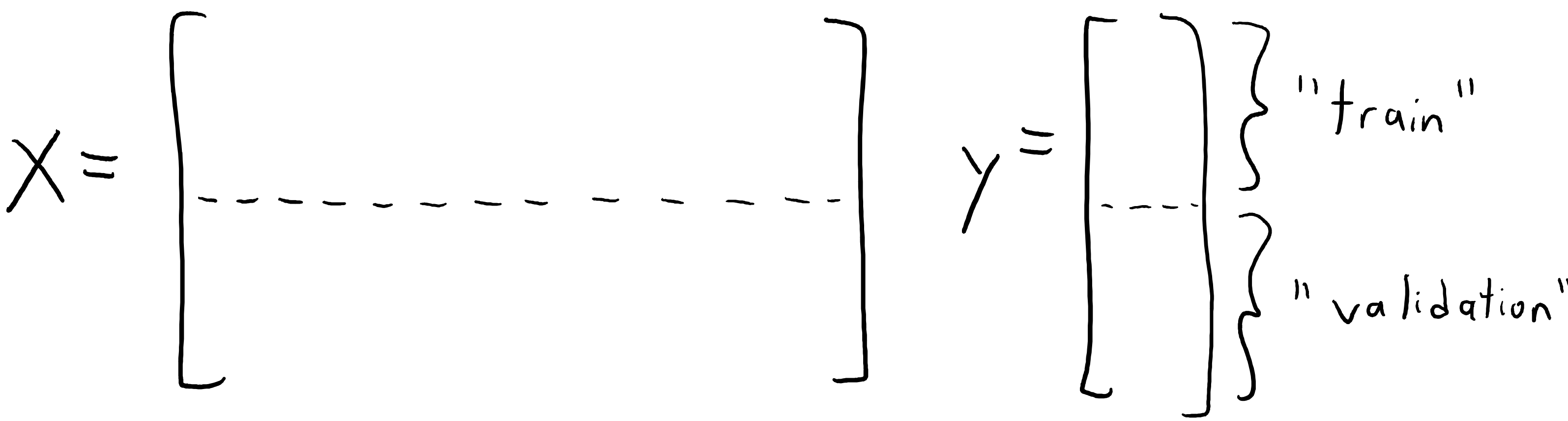
$$\underbrace{E_{test}}_{\text{"test error"}} = \underbrace{E_{approx}}_{\text{"approximation error"}} + \underbrace{E_{train}}_{\text{"training error"}}$$



  - Where $E_{approx} = (E_{test} - E_{train})$.
- E_{train} goes down as model gets complicated:
  - Training error goes down as a decision tree gets deeper.
- But E_{approx} goes up as model gets complicated:
  - Training error becomes a worse approximation of test error.

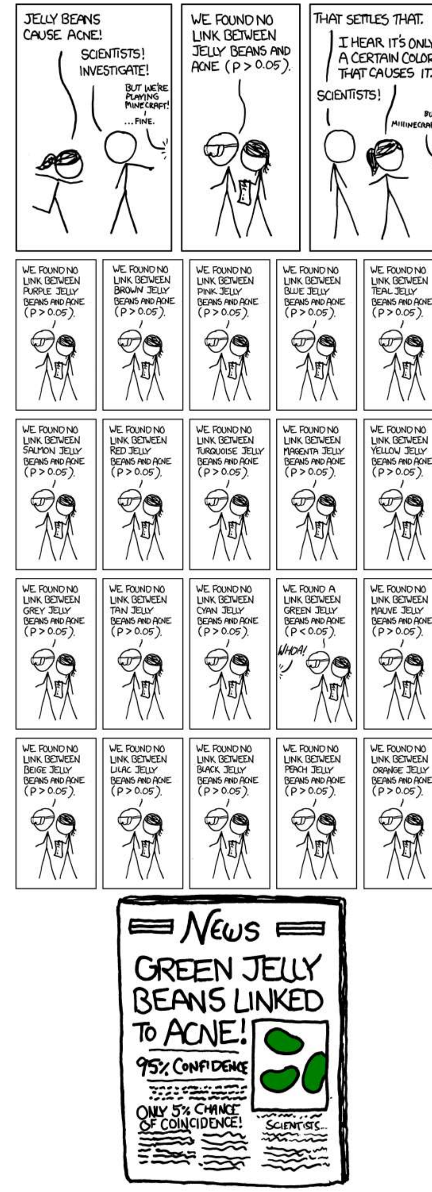
Last Time: Validation Error

- Golden rule: we can't look at test data during training.
- But we can approximate E_{test} with a validation error:
  - Error on a set of training examples we "hid" during training.



  - Find the decision tree based on the "train" rows.
  - Validation error is the error of the decision tree on the "validation" rows.
    - We typically choose "hyper-parameters" like depth to minimize the validation error.

P-value hacking:
One instance of optimization bias
https://xkcd.com/882/



Coming Up Next

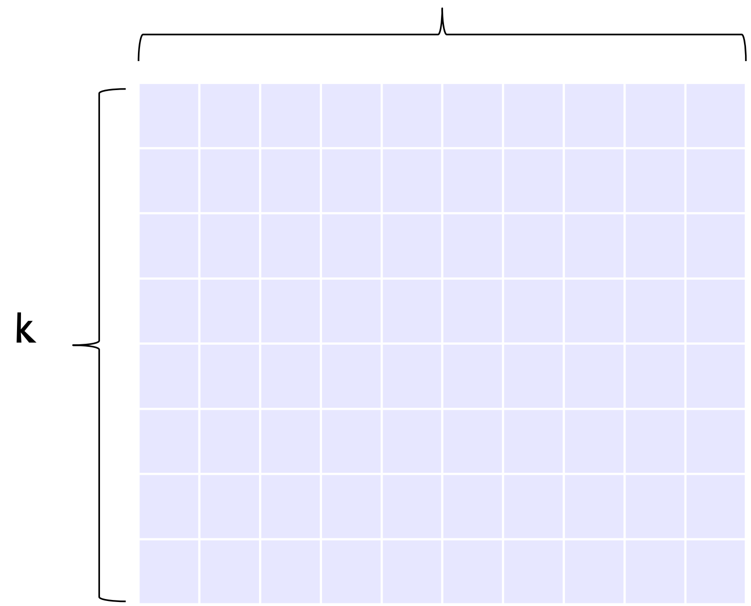
MORE ON OPTIMIZATION BIAS

“Search Space”

- Search space := the space of objects that are evaluated

Q: What is the search space for a decision stump?

- We looked at the grid of all possible {j,t} values
- j ∈ {1, 2, ..., d}, t ∈ {1, 2, ..., k}
- Search space is a d-by-k grid
  - Enumerating all possible decision stumps
- We evaluated all of the d-by-k grid
  - i.e. we evaluate the training error d*k times
- You could make the search space smaller
  - i.e. only look at certain j,t values



Space of possible decision stumps

"Search Space"

Q: Between training error and validation error,
which one has lower optimization bias for decision trees?

Q: What are the search spaces associated with
training error and validation error?

Used to optimize	Used to optimize
____________	____________

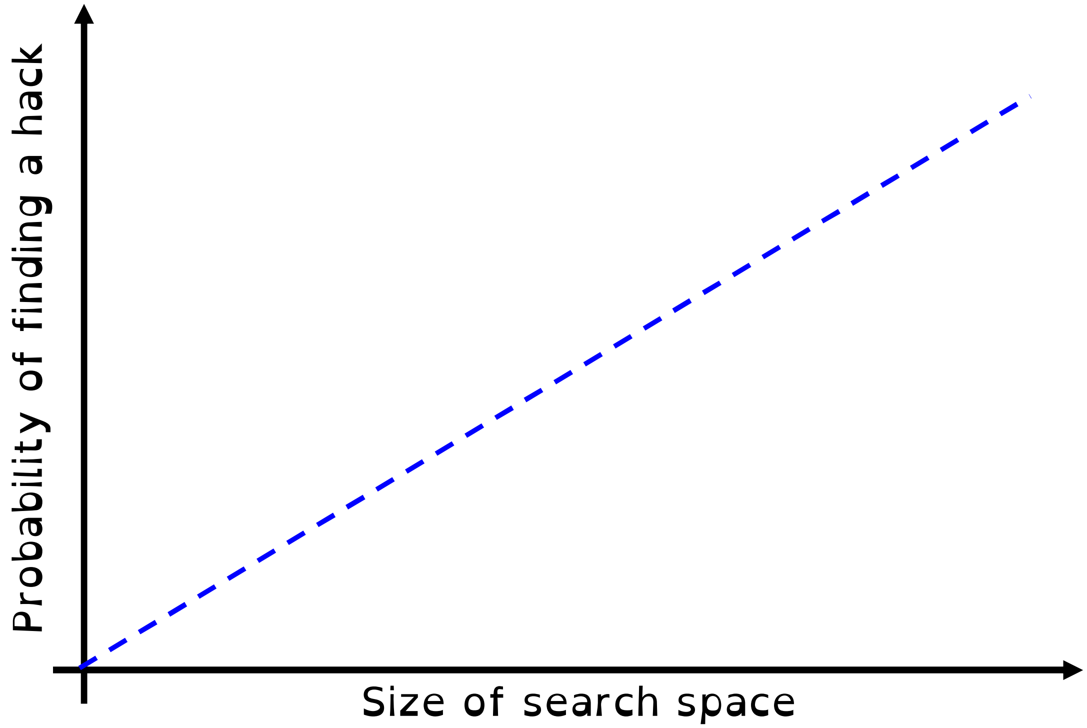
Search space for training error:
Space of possible _____

Search space for validation error:
Space of possible ______

Larger search space => more optimization bias

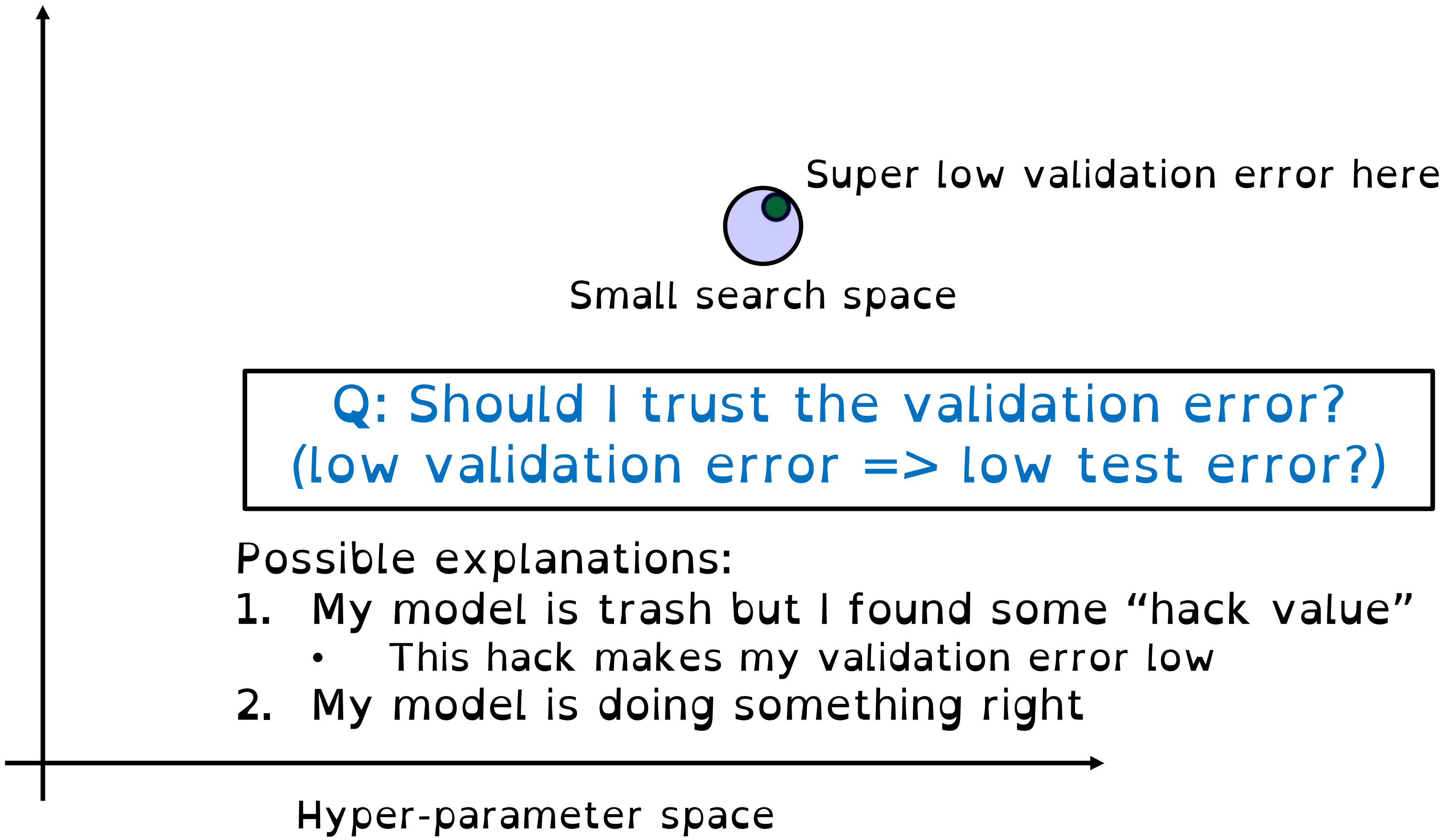
Finding a "Hack" Instead of Learning

- Achieve high score in boat-racing
  - Not by finishing race
  - Circle around infinitely collecting bonus
  - "reward hacking"



https://openai.com/blog/faulty-reward-functions/

Search Space and Optimization Bias

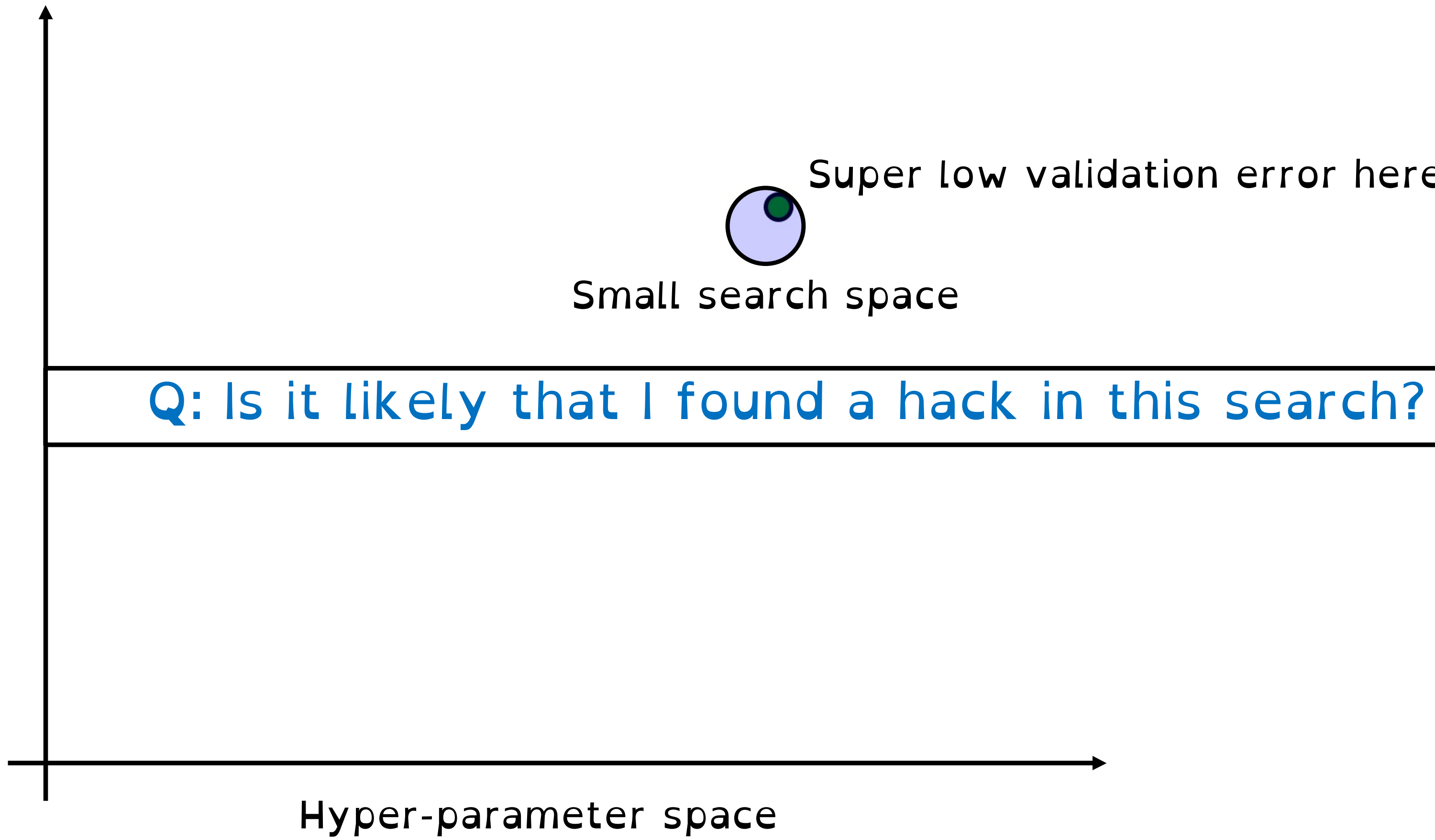


Q: Should I trust the validation error?
(low validation error => low test error?)

Possible explanations:

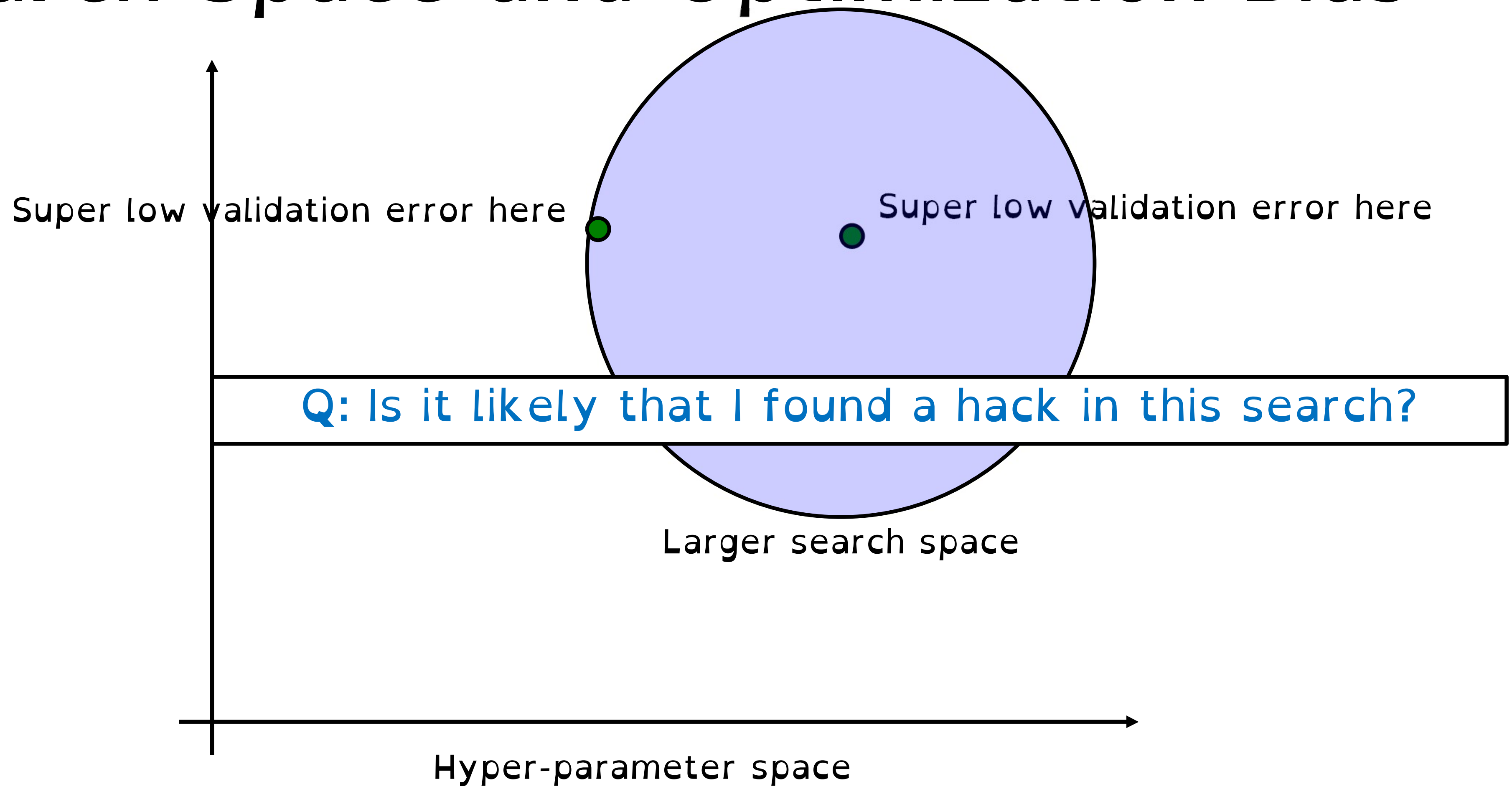
1. My model is trash but I found some “hack value”
   - This hack makes my validation error low
2. My model is doing something right

Search Space and Optimization Bias

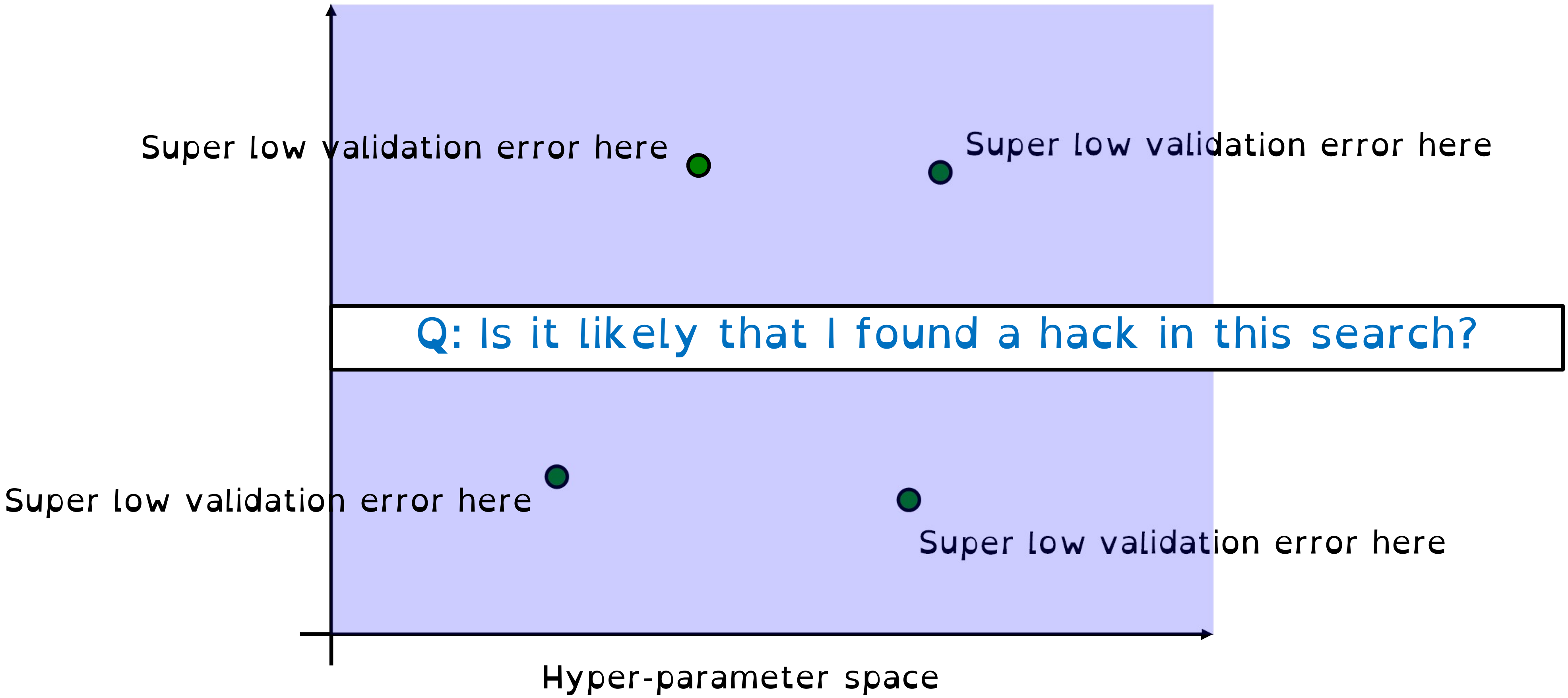


Q: Is it likely that I found a hack in this search?

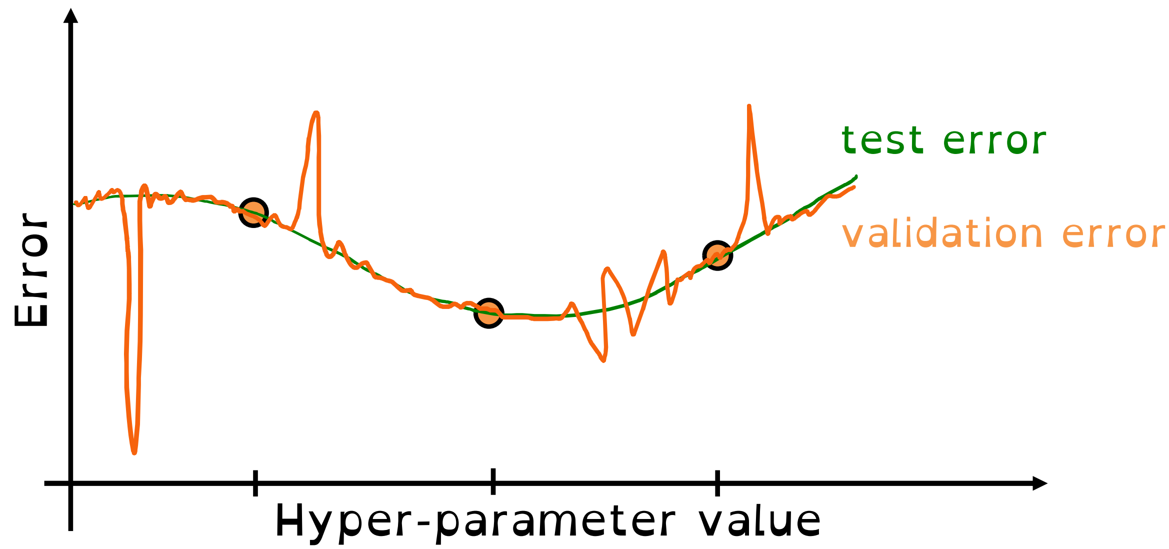
Search Space and Optimization Bias



Search Space and Optimization Bias

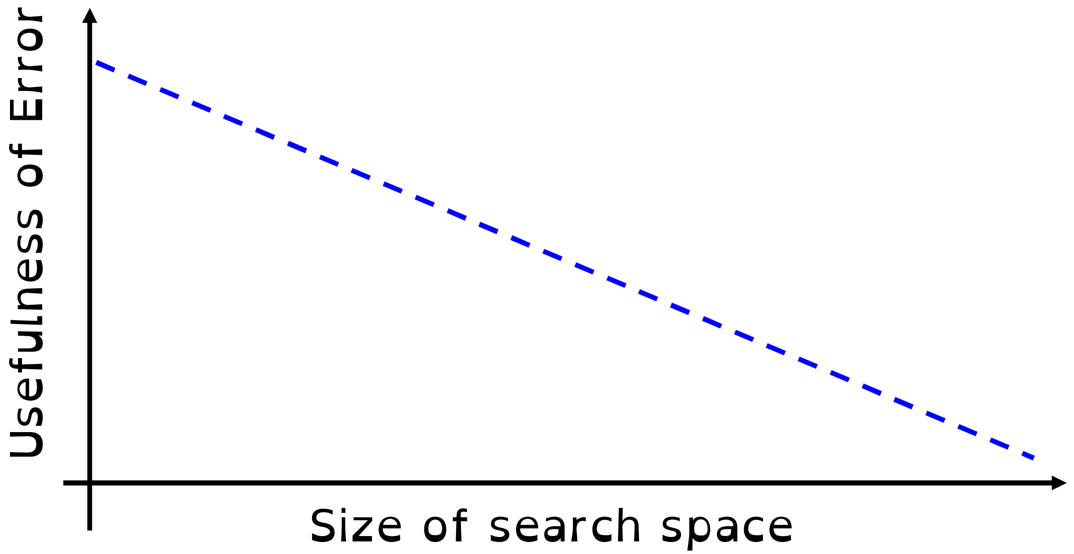


Validation Error Might Do This



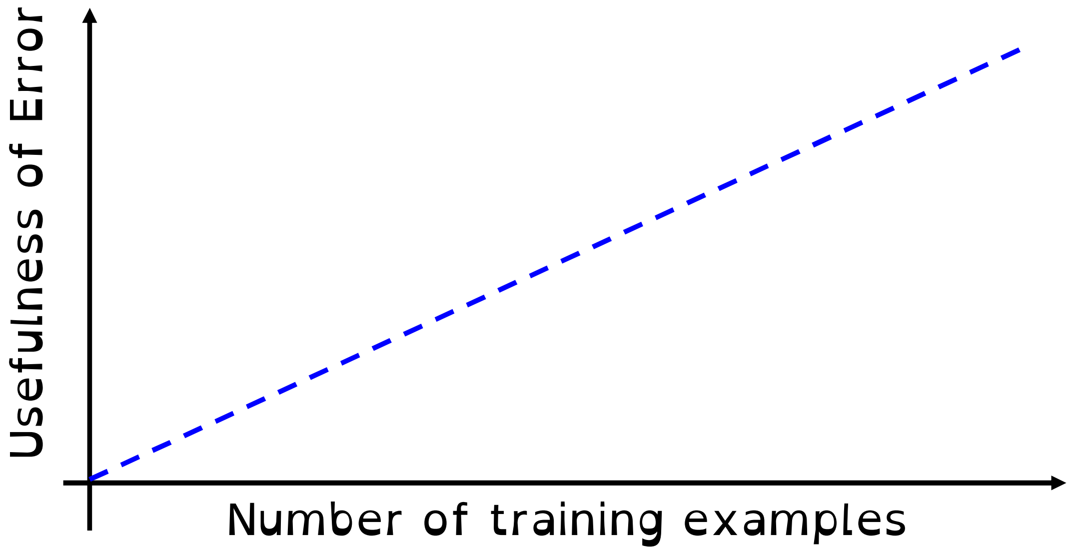
- Noise in the data can make validation error behave strangely in a very fine scale

Is Validation Error Trustworthy?



- Large search space => training error is not trustworthy
- Smaller search space => validation error is more trustworthy
- The more you look validation error, it becomes less trustworthy
- It's best to look the validation error only once
  - In practice, a "small" number of times is good enough

Is Validation Error Trustworthy?



- More training examples => better representation of distribution
  - Under IID, training examples and test examples become more similar
  - Likewise, validation examples and test examples become more similar
- It becomes harder to find a “lucky” case with more training examples

Train/Validation/Test Terminology

- Training set: used (a lot) to set parameters.
- Validation set: used (a few times) to set hyper-parameters.
- Testing set: used (once) to evaluate final performance.
- Deployment (real-world): what you really care about.

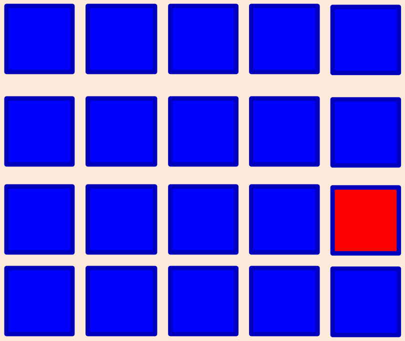
	fit	score	predict
Train	✓	✓	✓
Validation		✓	✓
Test		once	once
Deployment			✓

Validation Error and Optimization Bias

- Optimization bias is small if you only compare a few models:
  - Best decision tree on the training set among depths 1, 2, 3,..., 10.
  - Risk of overfitting to validation set is low if we try 10 things.
- Optimization bias is large if you compare a lot of models:
  - All possible decision trees of depth 10 or less.
  - Here we're using the validation set to pick between a billion+ models:
    - Risk of overfitting to validation set is high: could have low validation error by chance.
  - If you did this, you might want a second validation set to detect overfitting.
- And optimization bias shrinks as you grow size of validation set.

Optimization Bias leads to Publication Bias

- Suppose that 20 researchers perform the exact same experiment:



- They each test whether their effect is "significant" ($p < 0.05$).
  - 19/20 find that it is not significant.
  - But the 1 group finding it's significant publishes a paper about the effect.
- This is again optimization bias, contributing to publication bias.
  - A contributing factor to many reported effects being wrong.

Coming Up Next

CROSS-VALIDATION

Recall: E_{valid} and E_{test}

$$\mathbb{E}[E_{valid}] = \mathbb{E}[E_{test}]$$

E_{valid} is an unbiased approximator of E_{test} ...
... as long as it's evaluated only once.

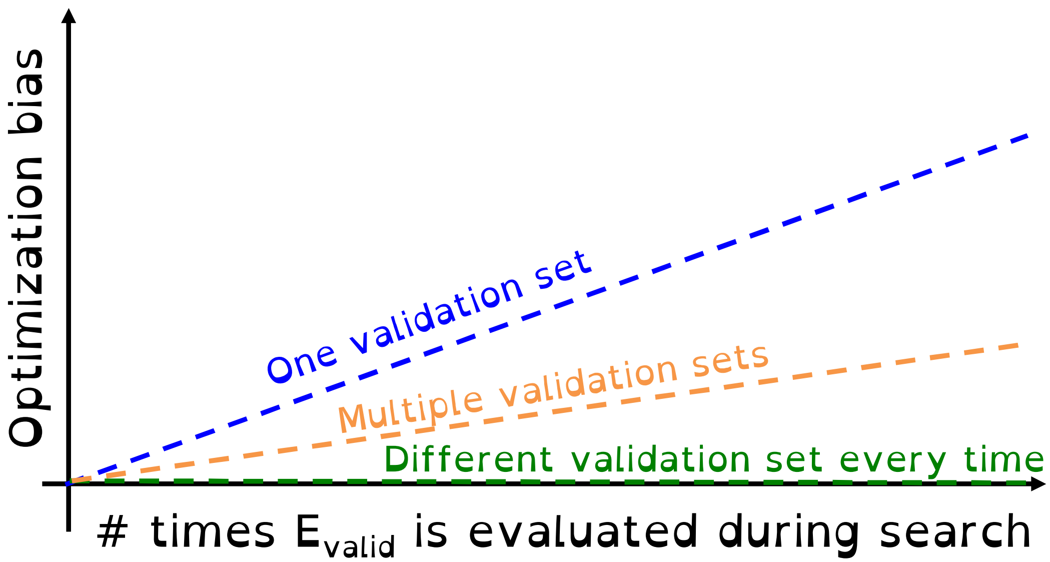
$$E_{valid} = E_{test} + \varepsilon$$

$$\mathbb{E}[\varepsilon] = \_\_\_\_$$

if E_{valid} is unbiased.

- The mean of error ϵ is a function of ________________

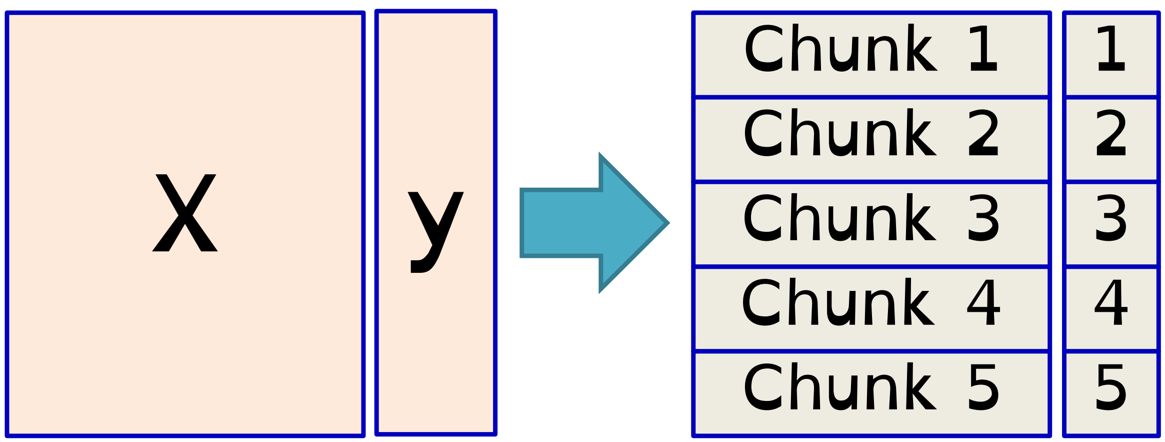
Recall: E_{valid} and E_{test}



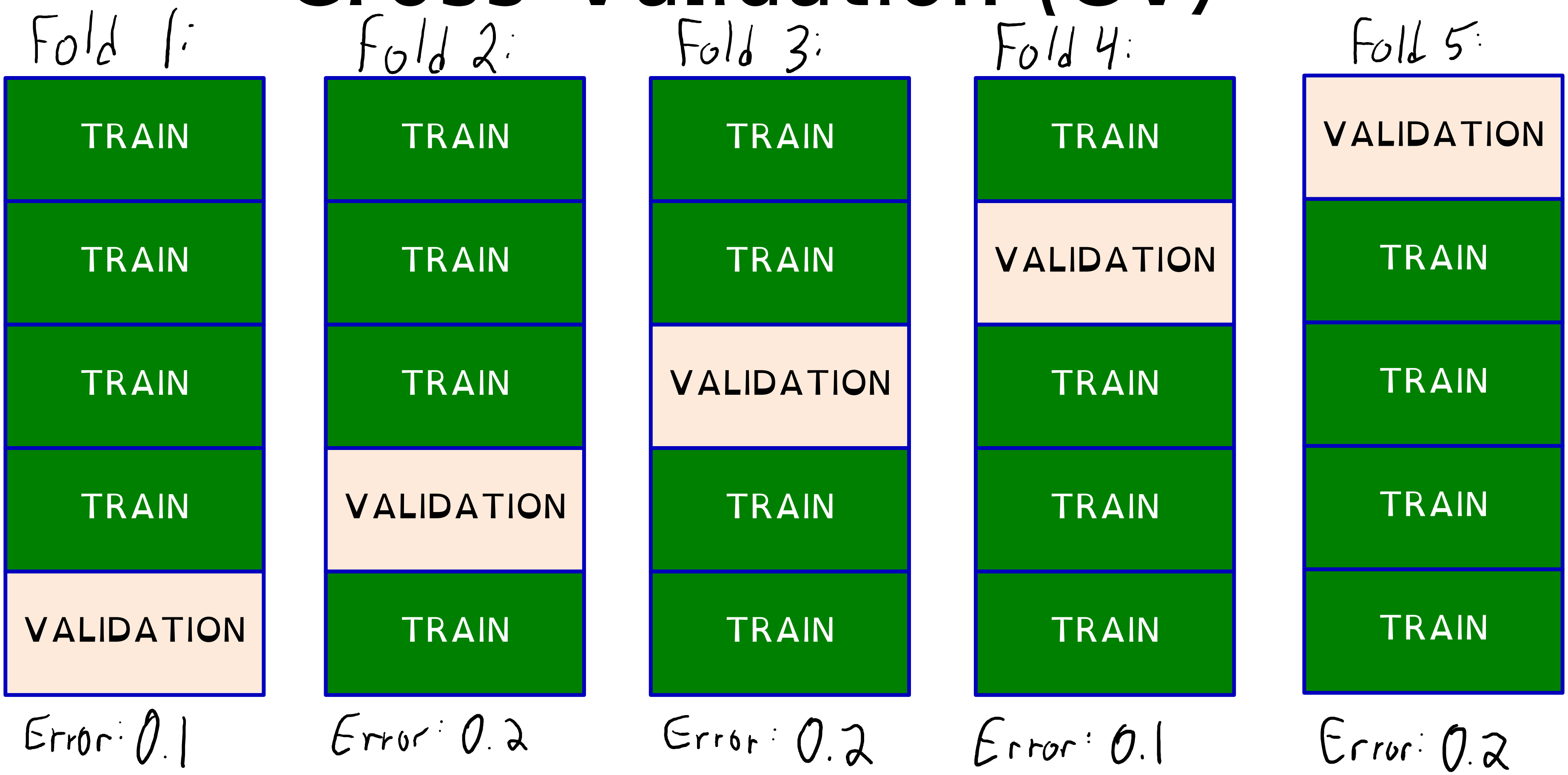
Cross-Validation (CV)

Q: How do we make **multiple validation sets** from the same training data?

- Idea: let's create multiple subsets of X and y.
  - 80% of data → training set X_{train} and y_{train}
  - 20% of data → validation set $X_{validate}$ and $y_{validate}$
  - We can do this split 5 times
- To do this, let's divide X and y into 5 chunks

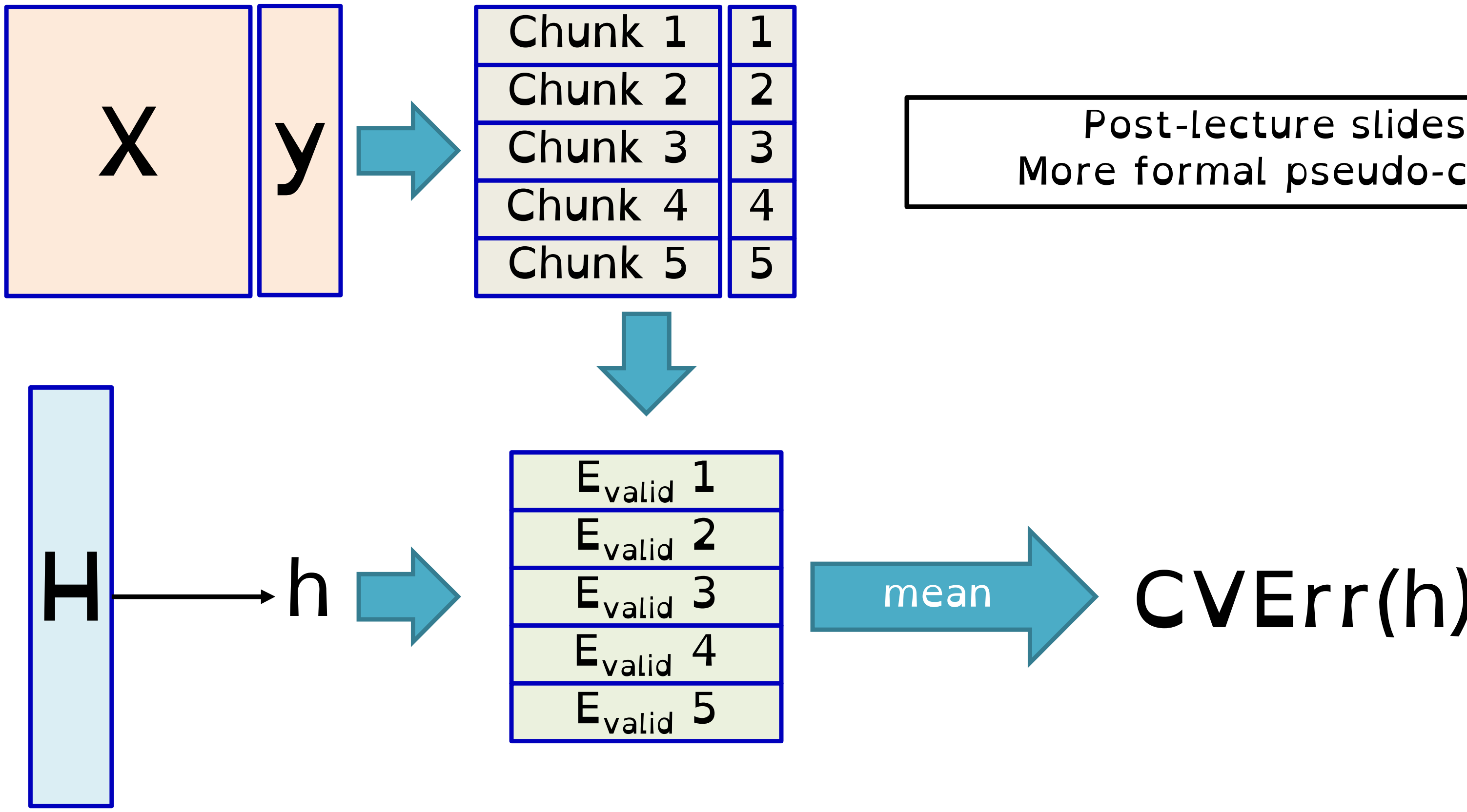


Cross-Validation (CV)



CV error estimate for this hyper-parameter: mean(errors) = 0.16

Hyper-Parameter Tuning with CV Pseudo-Code



Post-lecture slides:
More formal pseudo-code

Choose h that has smallest CVErr(h)

Cross-Validation (CV)

- You can take this idea further ("k-fold cross-validation"):
  - 10-fold cross-validation: train on 90% of data and validate on 10%.
    - Repeat 10 times and average (test on fold 1, then fold 2,..., then fold 10),
  - Leave-one-out cross-validation: train on all but one training example.
    - Repeat n times and average.
- Gets more accurate but more expensive with more folds.
  - To choose depth we compute the cross-validation score for each depth.

- As before, if data is ordered then folds should be random splits.
  - Randomize first, then split into fixed folds.

Cross-Validation Theory

- Does CV give unbiased estimate of test error?
  - Yes!
    - Since each data point is only used once in validation, expected validation error on each data point is test error.
  - But again, if you use CV to select among models then it is no longer unbiased.

- What about variance of CV?
  - Hard to characterize.
  - CV variance on ‘n’ data points is worse than with a validation set of size ‘n’.
    - But we believe it is close.

- Does cross-validation remove optimization bias?
  - No, but the bias might be smaller since you have more “test” points.

Me waiting to hear about the best ML model so I can make lots of money

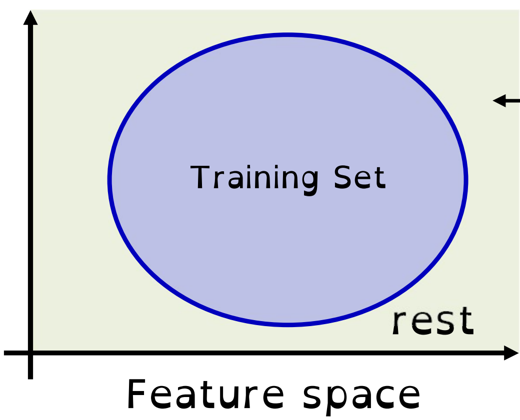
Coming Up Next

"BEST" MACHINE LEARNING MODEL

There is None

The “Best” Machine Learning Model

- Decision trees are not always most accurate on test error.
- What is the “best” machine learning model?
- An alternative measure of performance is the generalization error:
  - Average error over all x_i vectors that are not seen in the training set.
  - “How well we expect to do for a *completely unseen* feature vector”.

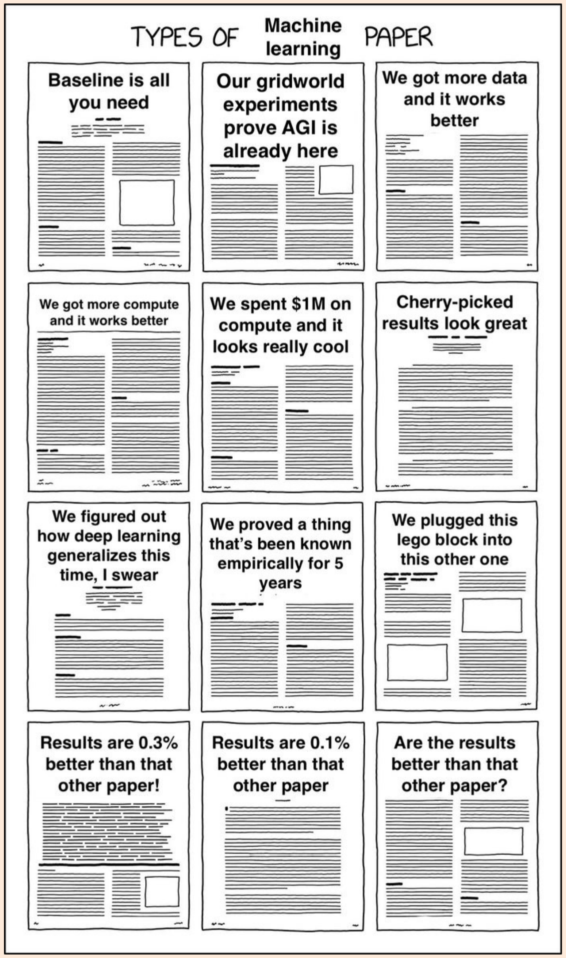


Generalization error:
Go through all of “rest” region,
Make prediction, compute error

The "Best" Machine Learning Model

- **No free lunch theorem** (proof in bonus slides):
  - There is **no** "best" model achieving the best generalization error for every problem.
  - If model A generalizes better to new data than model B on one dataset, there is another dataset where model B works better.
- This question is like asking which is "best" among "rock", "paper", and "scissors".
- Given a dataset, we need to try out multiple models.
- So which ones to study in CPSC 340?
  - We'll usually motivate each method by a specific application.
  - But we're focusing on models that have been effective in many applications.
- Machine learning research:
  - Large focus on models that are useful across many applications.

"State-Of-The-Art" Models



- A subset of ML research is OBSESSED with beating the state-of-the-art performance on benchmark tasks
  - ➢ State-of-the-art (SOTA)
    := test accuracy is best in the world
  - ➢ Benchmark tasks
    := well-known learning tasks
    (e.g. object recognition, machine translation, etc.)

- SOTA models for each task is very specialized.
  - ➢ Models that perform well on task A
    don't necessarily perform well on task B

- Reviewers look carefully for whether your model works well across different datasets for the same task
  - ➢ Otherwise, you are not SOTA.
    You just overfitted to one dataset!

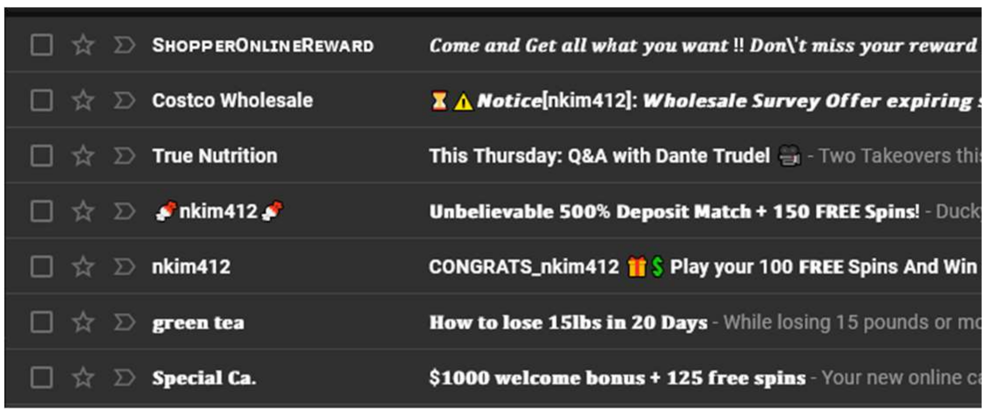
Coming Up Next

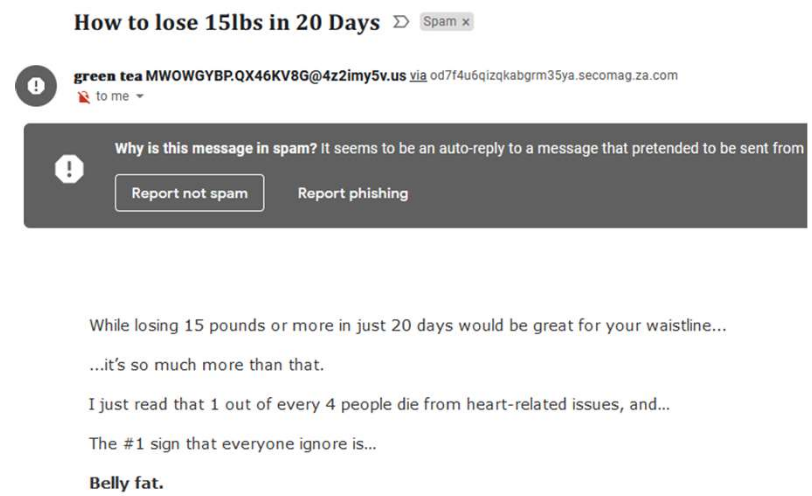
NAÏVE BAYES INTRO

Rev. Thomas Bayes

Application: Email Spam Filtering

- Want a build a system that detects spam emails.
  - Context: spam used to be a big problem.





Q: How do we formulate this as supervised learning?

Representing Emails

- Assumption: spam emails have a predictable pattern
  - Certain words occur more often in spams
    - E.g. “exclusive”, “offer”, “reward”, “Vicodin”, “keto”, etc.
  - Some words occur together more often in spams
    - E.g. “hi there”, “you have been selected”, “too late”, etc.
- We will represent emails with bag-of-words

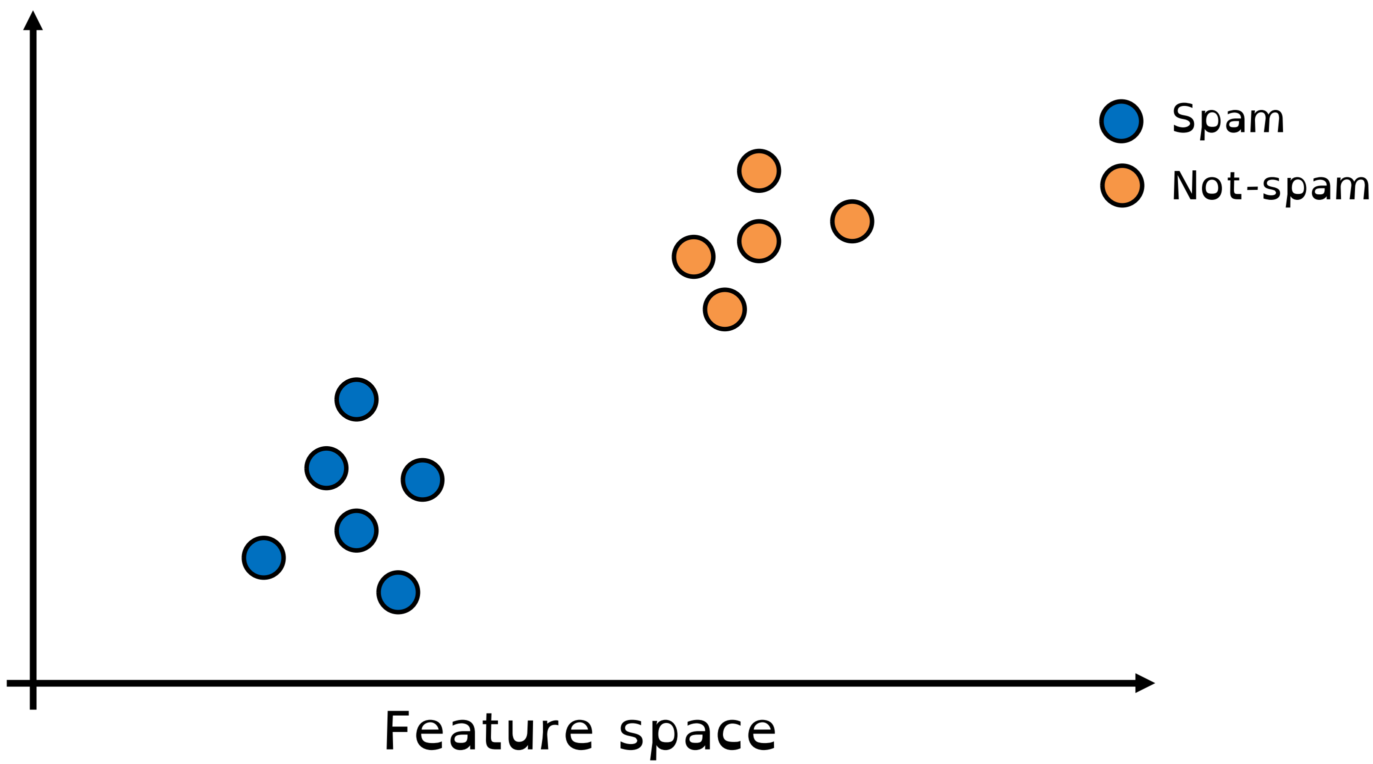
$	Hi	CPSC	340	Vicodin	Offer	…
1	1	0	0	1	0	…
0	0	0	0	1	1	…
0	1	1	1	0	0	…
…	…	…	…	…	…	…

d features: keywords for bag

- x_{ij} = 1 if word/phrase ‘j’ is in email ‘i’, x_{ij} = 0 if it is not.

Space of Emails

Spams have predictable patterns
=> spams and not-spams look different in space of emails



Spam Filtering as Supervised Learning

- Collect a large number of emails, gets user to label them.

$	Hi	CPSC	340	Vicodin	Offer	…
1	1	0	0	1	0	…
0	0	0	0	1	1	…
0	1	1	1	0	0	…
…	…	…	…	…	…	…

Spam?
1
1
0
…

- $y_i = 1$ if email 'i' is spam, $y_i = 0$ if email is not spam.

Probabilistic Classifiers

- For years, best spam filtering methods used naïve Bayes.
  - A probabilistic classifier based on Bayes rule.
  - It tends to work well with bag of words.
  - Recently shown to improve on state of the art for CRISPR "gene editing" (link).

- Probabilistic classifiers: use probability for generating predictions
  - Model the conditional probability, $p(y_i \mid x_i)$.
  - "If a message has words x_i, what is probability that message is spam?"

- Classify it as spam if probability of spam is higher than not spam:
  - If $p(y_i = \text{"spam"} \mid x_i) > p(y_i = \text{"not spam"} \mid x_i)$
    - return "spam".
  - Else
    - return "not spam".

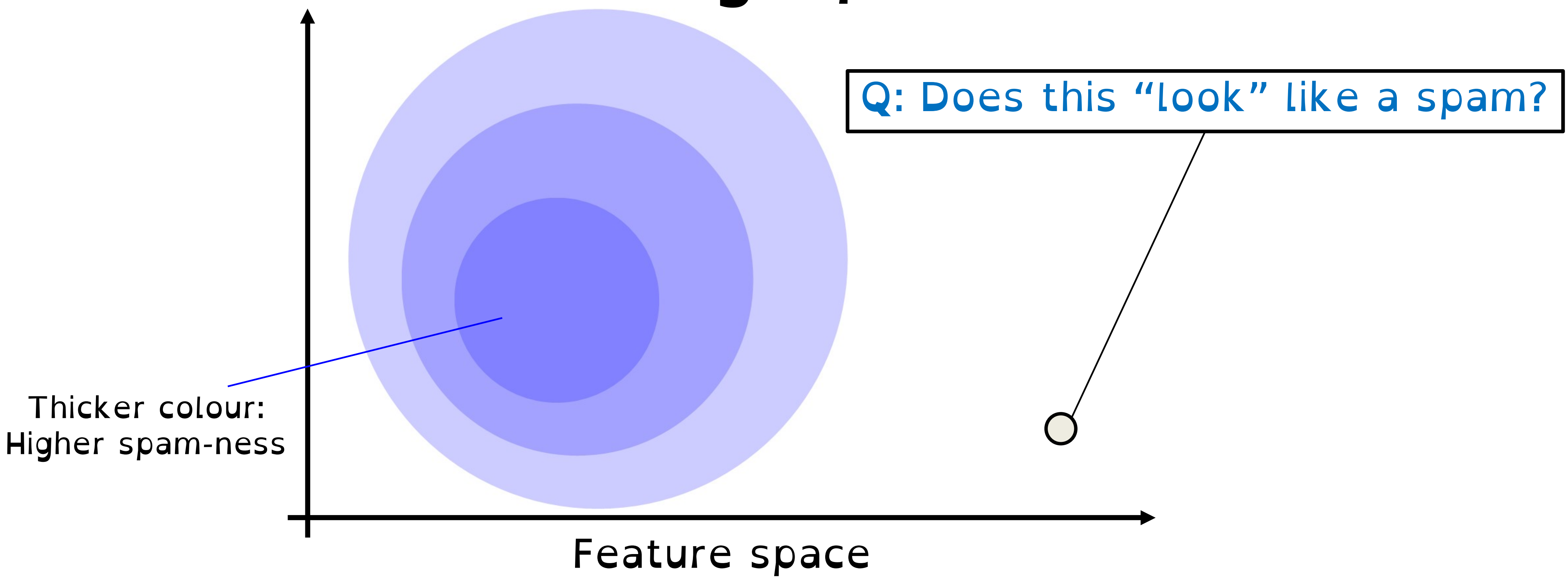
Note on Learned Probability

- $p(y_i = \text{"spam"} \mid x_i)$ reads:
  "probability that message is spam given these features"

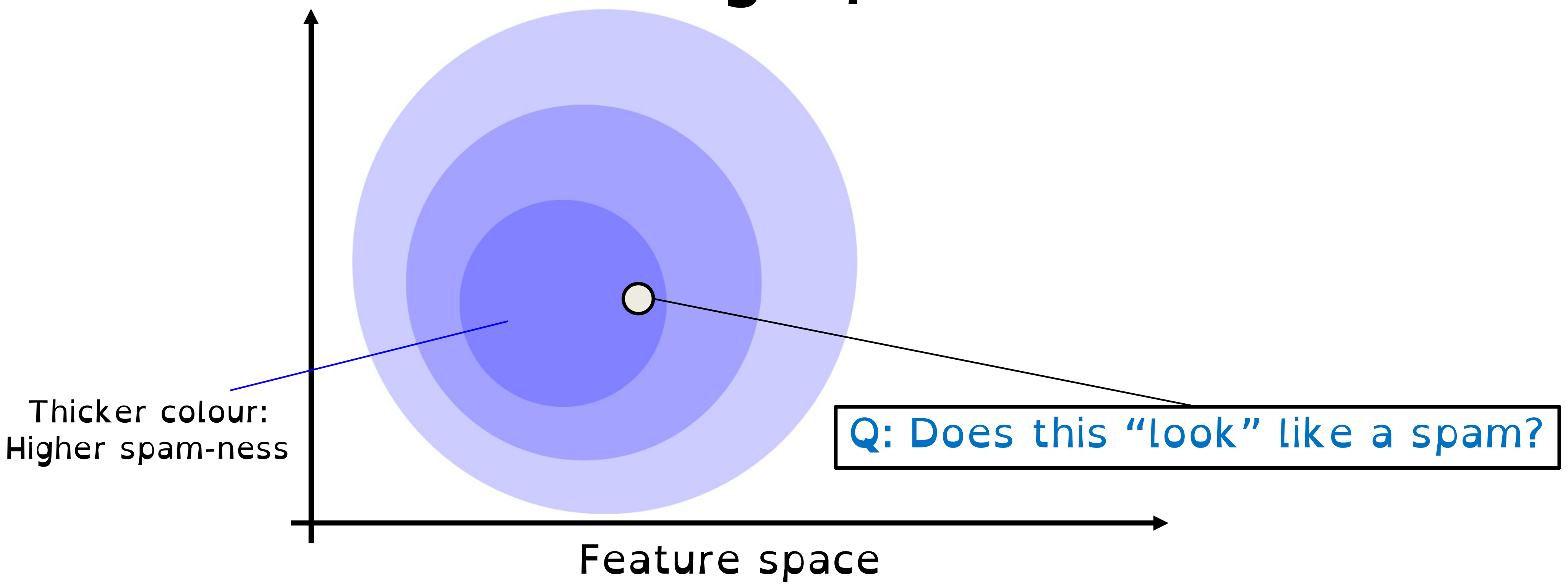
- In practice, we treat it more like a score:
  "the spam-ness of the input message"

- Our goal is to build a model that can compute the spam-ness, based on the examples of spam messages

Visualizing Spam-ness



Visualizing Spam-ness



Coming Up Next

NAÏVE BAYES DETAILS

Computing Spam-ness

$$p(y_i = \text{“spam”} \mid x_i)$$

- Naïve Bayes uses Bayes rule:

$$p(y_i = \text{"spam"} \mid x_i) = \frac{p(x_i \mid y_i = \text{"spam"})\, p(y_i = \text{"spam"})}{p(x_i)}$$

- On the right we have three terms:
  - Marginal probability $p(y_i)$ that an email is spam.
  - Marginal probability $p(x_i)$ that an email has the set of words x_i.
  - Conditional probability $p(x_i \mid y_i)$ that a spam e-mail has the words x_i.
    - And the same for non-spam e-mails.

What is $p(y_i)$?

$$p(y_i = \text{"spam"} \mid x_i) = \frac{p(x_i \mid y_i = \text{"spam"})\, p(y_i = \text{"spam"})}{p(x_i)}$$

- $p(y_i$ = "spam") is the "baseline spam-ness"
  - Probability that an email is a spam, without even looking at features.

Q: How do I learn this quantity?

Step 1: Look at all emails ~~in existence~~ in dataset
Step 2: Count the number of spams

What is $p(x_i)$?

$$p(y_i = \text{"spam"} \mid x_i) = \frac{p(x_i \mid y_i = \text{"spam"})\, p(y_i = \text{"spam"})}{p(x_i)}$$

- $p(x_i)$ is the is probability that a random email looks like x_i

Q: How do I learn this quantity?

Step 1: Look at all emails ~~in existence~~ in dataset

Step 2: Count the number of times x_i occurs

What is $p(x_i \mid y_i)$?

$$p(y_i = \text{"spam"} \mid x_i) = \frac{\boxed{p(x_i \mid y_i = \text{"spam"})}\, p(y_i = \text{"spam"})}{p(x_i)}$$

- $p(x_i \mid y_i=\text{“spam”})$ is the is probability that a random spam looks like x_i

Q: How do I learn this quantity?

Step 1: Look at all **spams** ~~in existence~~ in dataset

Step 2: Count the number of times x_i occurs

IID Assumption

ALL EMAILS
IN EXISTENCE

Too big to analyze

EMAILS
IN DATASET

n is smaller but decently large

- IID assumption lets us treat the dataset as a snapshot of truth
  - i.e. emails in dataset (somewhat) accurately reflect the patterns in all emails in existence.
- Then probabilities can be estimated by frequencies in dataset

Counting for $p(x_i)$ and $p(x_i \mid y_i)$

- Seeing all possible examples at least once is extremely unlikely!

$	Hi	CPSC	340	Vicodin	Offer	…
1	0	0	0	0	0	…
0	1	0	0	0	0	…
0	0	1	0	0	0	…
…	…	…	…	…	…	…

d features: keywords for bag

- I need to have O(_) examples in order to see all possible examples.
- If I had fewer examples than that,
  I'll end up setting $p(x_i)$ and $p(x_i \mid y_i)$ to 0 all the time

Q: What should we do about that?

Getting Rid of $p(x_i)$

$$p(y_i = \text{"spam"} \mid x_i) = \frac{p(x_i \mid y_i = \text{"spam"})\, p(y_i = \text{"spam"})}{p(x_i)}$$

Naive Bayes returns "spam" if $p(y_i = \text{"spam"} \mid x_i) > p(y_i = \text{"not spam"} \mid x_i)$.

By Bayes rule this means $\dfrac{p(x_i \mid y_i = \text{"spam"})\, p(y_i = \text{"spam"})}{p(x_i)} > \dfrac{p(x_i \mid y_i = \text{"not spam"})\, p(y_i = \text{"not spam"})}{p(x_i)}$

Multiply both sides by $p(x_i)$:

$$p(x_i \mid y_i = \text{"spam"})\, p(y_i = \text{"spam"}) > p(x_i \mid y_i = \text{"not spam"})\, p(y_i = \text{"not spam"})$$

Naïve Bayes

- Naïve Bayes makes a big assumption to make things easier:

$$\underbrace{p(\text{hello}=1, \text{vicodin}=0, 340=1 \mid \text{spam})}_{\text{HARD}} \approx \underbrace{p(\text{hello}=1 \mid \text{spam})}_{\text{easy}}\, \underbrace{p(\text{vicodin}=0 \mid \text{spam})}_{\text{easy}}\, \underbrace{p(340=1 \mid \text{spam})}_{\text{easy}}$$

- We assume *all* features x_i are conditionally independent give label y_i.
  - Once you know it's spam, probability of "vicodin" doesn't depend on "340".
  - Definitely not true, but sometimes a good approximation.

- And now we only need easy quantities like p("vicodin" = 0| y_i = "spam").

What is p("Vicodin" = 0| y_i = "spam")?

$$p(hello=1, vicodin=0, 340=1 \mid spam) \approx p(hello=1 \mid spam)\, p(vicodin=0 \mid spam)\, p(340=1 \mid spam)$$

- p("vicodin" = 0| y_i = "spam") is the
  is probability that a spam does not contain the word "Vicodin"

Q: How do I learn this quantity?

Step 1: Look at all spams ~~in existence~~ in dataset

Step 2: Count the number of times "Vicodin" doesn't occur

Summary

- Optimization bias: using a validation set too much overfits.
- Cross-validation: allows better use of data to estimate test error.
- No free lunch theorem: there is no "best" ML model.
- Probabilistic classifiers: try to estimate $p(y_i \mid x_i)$.
- Naïve Bayes: simple probabilistic classifier based on counting.
  - Uses conditional independence assumptions to make training practical.

- Next time:
  - A "best" machine learning model as 'n' goes to ∞.

Review Questions

- Q1: Is having a super small search space always a good idea for hyper-parameter tuning?
- Q2: In practice, people rarely use cross-validation for very large datasets. Why?
- Q3: If we're using Naïve Bayes for spam filtering,
  why can a non-binary bag-of-words be problematic?
- Q4: What is so naïve about Naïve Bayes?

Cross-Validation Pseudo-Code

To choose depth

```
for depth in 1:20
    compute cross-validation score
return depth with highest score
```

To compute 5-fold cross-validation score:

```
for fold in 1:5
    train 80% that doesn't include fold
    test on fold
return average test error
```

Notes:

- This fits 100 models! (20 depths times 5 folds)
- We get one (average) score for each of the 20 depths.
- Use this score to pick depth

Feature Representation for Spam

- Are there better features than bag of words?
  - We add bigrams (sets of two words):
    - "CPSC 340", "wait list", "special deal".
  - Or trigrams (sets of three words):
    - "Limited time offer", "course registration deadline", "you're a winner".
  - We might include the sender domain:
    - <sender domain == "mail.com">.
  - We might include regular expressions:
    - <your first and last name>.

Back to Decision Trees

- Instead of validation set, you can use CV to select tree depth.

- But you can also use these to decide whether to split:
  - Don't split if validation/CV error doesn't improve.
  - Different parts of the tree will have different depths.
- Or fit deep decision tree and use [cross-]validation to prune:
  - Remove leaf nodes that don't improve CV error.

- Popular implementations that have these tricks and others.

Random Subsamples

- Instead of splitting into k-folds, consider "random subsample" method:
  - At each "round", choose a random set of size 'm'.
    - Train on all examples except these 'm' examples.
    - Compute validation error on these 'm' examples.

- Advantages:
  - Still an unbiased estimator of error.
  - Number of "rounds" does not need to be related to "n".
- Disadvantage:
  - Examples that are sampled more often get more "weight".

Handling Data Sparsity

- Do we need to store the full bag of words 0/1 variables?
  - No: only need list of non-zero features for each e-mail.

$	Hi	CPSC	340	Vicodin	Offer	...
1	1	0	0	1	0	...
0	0	0	0	1	1	...
0	1	1	1	0	0	...
1	1	0	0	0	1	...

vs.

Non-Zeroes
{1,2,5,…}
{5,6,…}
{2,3,4,…}
{1,2,6,…}

  - Math/model doesn't change, but more efficient storage.

Generalization Error

- An alternative measure of performance is the generalization error:
  - Average error over the set of x^i values that are not seen in the training set.
  - "How well we expect to do for a *completely unseen* feature vector".
- Test error vs. generalization error when labels are deterministic:

$$E_{test} = \mathbb{E}\left[|\hat{y}^i - \tilde{y}^i| \right] \qquad E_{generalization} = \frac{1}{t} \sum_{\tilde{x}^i \notin \{\text{train set}\}} |\hat{y}_i - \tilde{y}_i|$$

Labels are deterministic, but we still take expectation over data distribution

number of x^i values not in training set.

average error over unseen x^i values.

“Best” and the “Good” Machine Learning Models

- Question 1: what is the “best” machine learning model?
  - The model that gets lower generalization error than all other models.
- Question 2: which models always do better than random guessing?
  - Models with lower generalization error than “predict 0” for all problems.

- No free lunch theorem:
  - There is **no** “best” model achieving the best generalization error for every problem.
  - If model A generalizes better to new data than model B on one dataset,
    there is another dataset where model B works better.

No Free Lunch Theorem

- Let's show the "no free lunch" theorem in a simple setting:
  - The x^i and y^i are binary, and y^i being a deterministic function of x^i.
- With 'd' features, each "learning problem" is a map from $\{0,1\}^d$ -> $\{0,1\}$.
  - Assigning a binary label to each of the 2^d feature combinations.

Feature 1	Feature 2	Feature 3
0	0	0
0	0	1
0	1	0
…	…	…

y (map 1)	y (map 2)	y (map 3)	…
0	1	0	…
0	0	1	…
0	0	0	…
…	…	…	…

- Let's pick one of these 'y' vectors ("maps" or "learning problems") and:
  - Generate a set training set of 'n' IID samples.
  - Fit model A (convolutional neural network) and model B (naïve Bayes).

No Free Lunch Theorem

- Define the "unseen" examples as the $(2^d - n)$ not seen in training.
  - Assuming no repetitions of x^i values, and $n < 2^d$.
  - Generalization error is the average error on these "unseen" examples.
- Suppose that model A got 1% error and model B got 60% error.
  - We want to show model B beats model A on another "learning problem".

- Among our set of "learning problems" find the one where:
  - The labels y^i agree on all training examples.
  - The labels y^i disagree on all "unseen" examples.
- On this other "learning problem":
  - Model A gets 99% error and model B gets 40% error.

Proof of No Free Lunch Theorem

- Let's show the "no free lunch" theorem in a simple setting:
  - The x^i and y^i are binary, and y^i being a deterministic function of x^i.
- With 'd' features, each "learning problem" is a map from each of the 2^d feature combinations to 0 or 1: $\{0,1\}^d \rightarrow \{0,1\}$

Feature 1	Feature 2	Feature 3
0	0	0
0	0	1
0	1	0
…	…	…

Map 1	Map 2	Map 3	…
0	1	0	…
0	0	1	…
0	0	0	…
…	…	…	…

- Let's pick one of these maps ("learning problems") and:
  - Generate a set training set of 'n' IID samples.
  - Fit model A (convolutional neural network) and model B (naïve Bayes).

Proof of No Free Lunch Theorem

- Define the "unseen" examples as the $(2^d - n)$ not seen in training.
  - Assuming no repetitions of x^i values, and $n < 2^d$.
  - Generalization error is the average error on these "unseen" examples.
- Suppose that model A got 1% error and model B got 60% error.
  - We want to show model B beats model A on another "learning problem".

- Among our set of "learning problems" find the one where:
  - The labels y^i agree on all training examples.
  - The labels y_i disagree on all "unseen" examples.
- On this other "learning problem":
  - Model A gets 99% error and model B gets 40% error.

Proof of No Free Lunch Theorem

- Further, across all “learning problems” with these ‘n’ examples:
  - Average generalization error of **every** model is 50% on unseen examples.
    - It’s right on each unseen example in exactly half the learning problems.
  - With ‘k’ classes, the average error is (k-1)/k (random guessing).

- This is kind of depressing:
  - For general problems, no “machine learning” is better than “predict 0”.

- But the proof also reveals the problem with the NFL theorem:
  - Assumes every “learning problem” is equally likely.
  - World encourages patterns like “similar features implies similar labels”.