CPSC 340:
Machine Learning and Data Mining

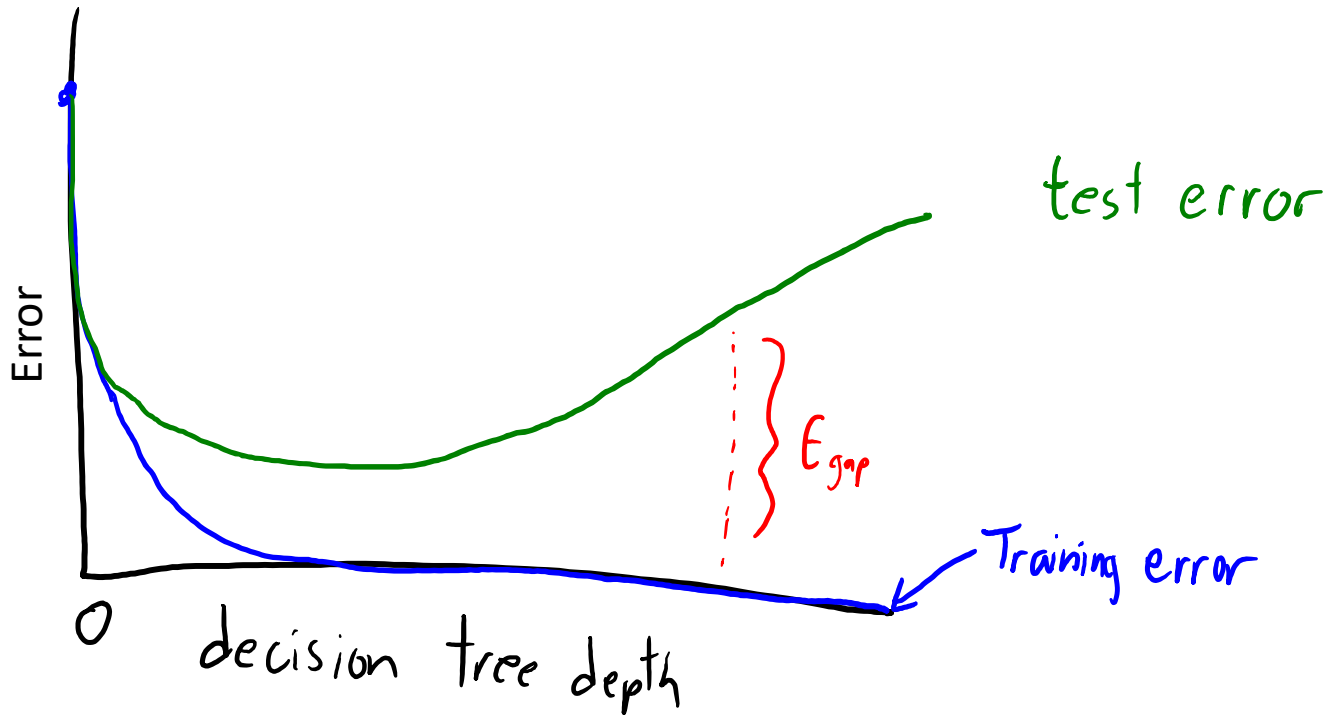
Probabilistic Classification

Admin

- Assignment 1 is due Friday: you should be almost done.
  - You can use 1 late days to submit Monday, 2 for Wednesday.
- Waiting list people: everyone should be in.
- Auditors:
  - Bring your forms at the end of class.

Fundamental Trade-Off

- Note “fundamental trade-off” is UBC terminology
- Also tends to be true (not always true)
- And only starts once overfitting starts



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.

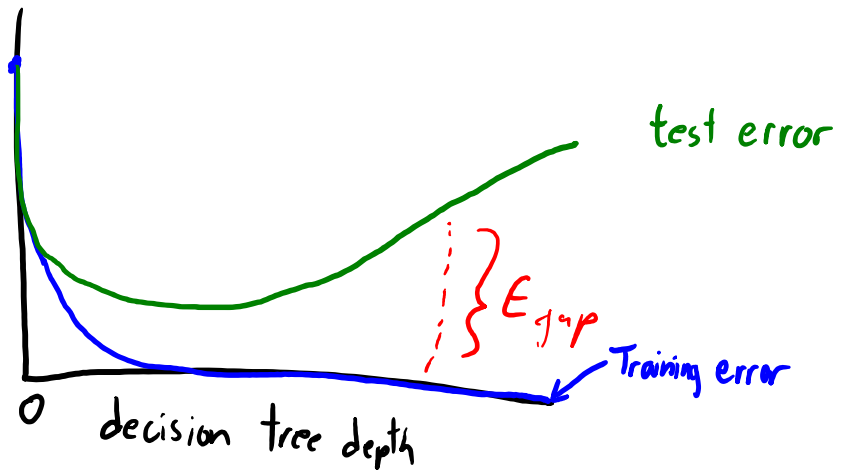
Last Time: Fundamental Trade-Off

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

$$E_{test} = E_{gap} + E_{train}$$

"test error" "generalization gap" "training error"

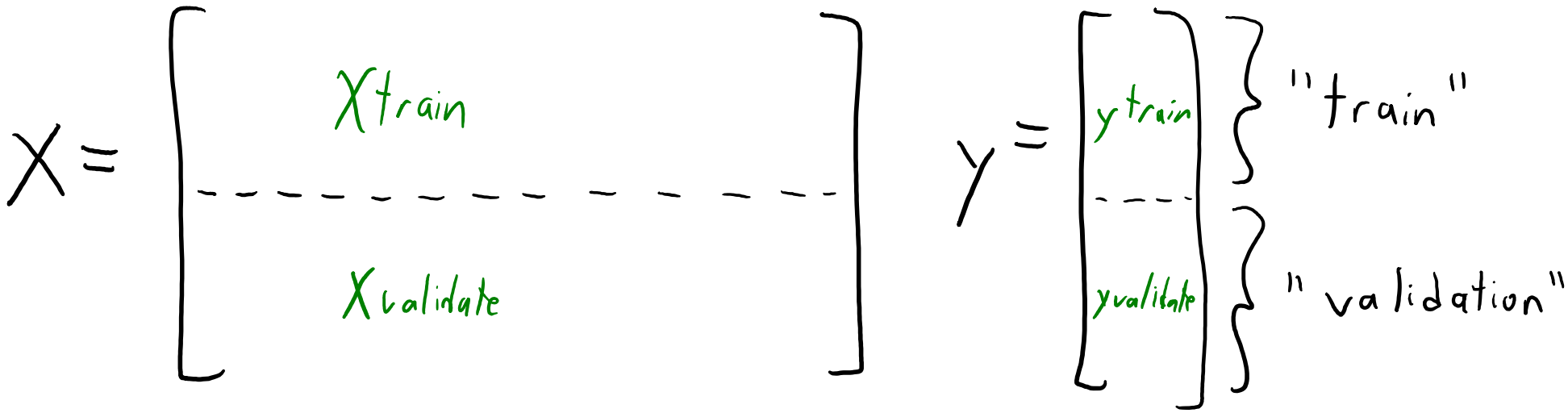
  - Where $E_{gap} = (E_{test} - E_{train})$.



- E_{train} goes down as model gets complicated:
  - Training error goes down as a decision tree gets deeper.
- But E_{gap} goes up as model gets complicated:
  - Training error becomes a worse approximation of test error.

Last Time: Validation Error

- Key principle: we cannot let test data influence 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.

Overfitting to the Validation Set?

- We can overfit to the validation set (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.

- But validation error usually has lower optimization bias than train error.
  - Might optimize over 20 values of "depth", instead of millions+ of possible trees.
    - Amount of overfitting to validation set is low if we only try 10 things.

- Optimization bias is larger 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).

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 second half of data did not influence training.

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.
  - Probably know key principle of not letting test data influence training.

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 last third is not influencing training.
  - And assuming you are using IID data in the first place.

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

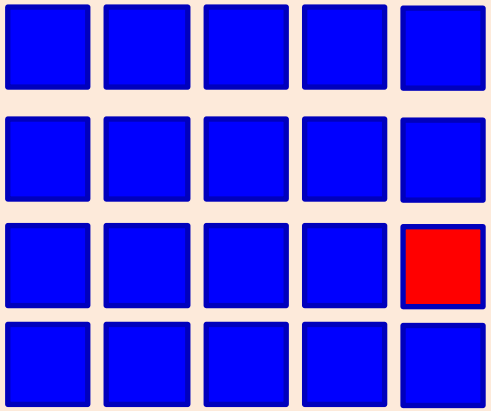
- Optimization bias is small if you only compare a few models:
  - Best decision tree on the training set among depths 1, 2, 3,…, 10.
  - Risk of overfitting to validation set is low if we try 10 things.

- 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.

Aside: Optimization Bias leads to Publication Bias

- Suppose that 20 researchers perform the exact same experiment:

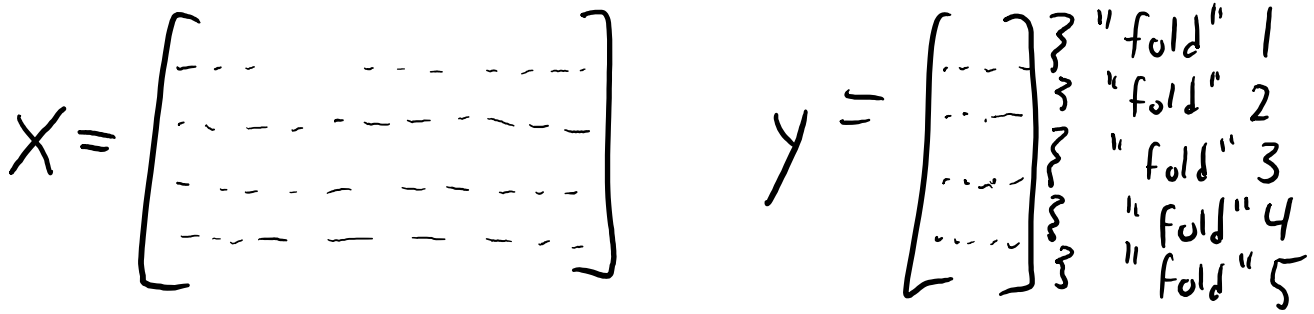


- They each test whether their effect is "significant" ($p < 0.05$).
  - 19/20 find that it is not significant.
  - But the 1 group finding it's significant publishes a paper about the effect.

- This is again optimization bias, contributing to publication bias.
  - A contributing factor to many reported effects being wrong.

Cross-Validation (CV)

- Isn't it wasteful to only use part of your data?
- 5-fold cross-validation:
  - Train on 80% of the data, validate on the other 20%.
  - Repeat this 5 times with the different splits, and average the score.

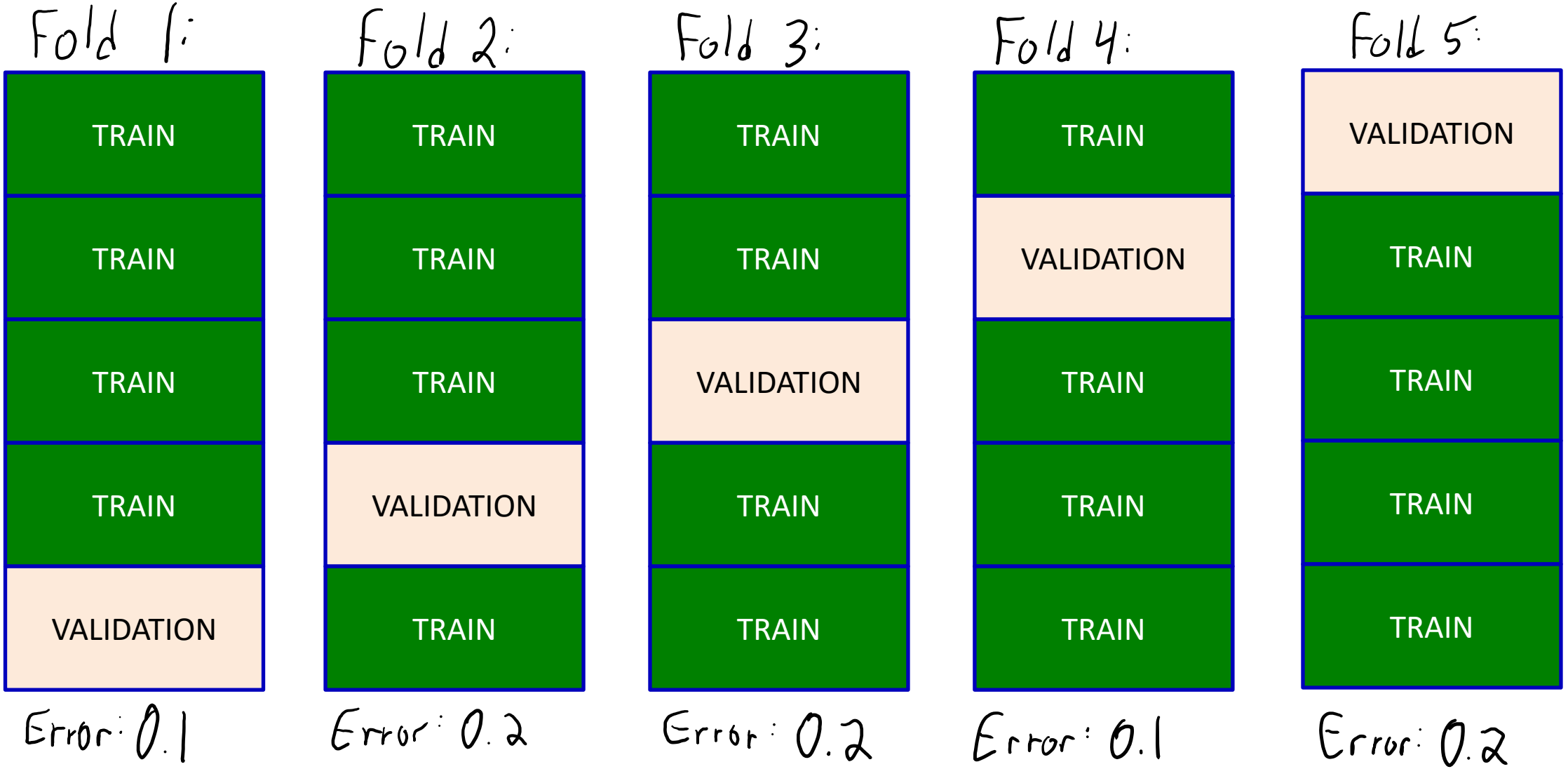


1. Train on folds $\{1,2,3,4\}$, compute error on fold 5.
2. Train on folds $\{1,2,3,5\}$, compute error on fold 4.
3. Train on folds $\{1,2,4,5\}$, compute error on fold 3.

⋮

6. Take average of the 5 errors as approximation of test error

Cross-Validation (CV)



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

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.
- This procedure only picks the depth.
  - ↳ You might then train on the whole dataset with the chosen depth.

Cross-Validation (CV)

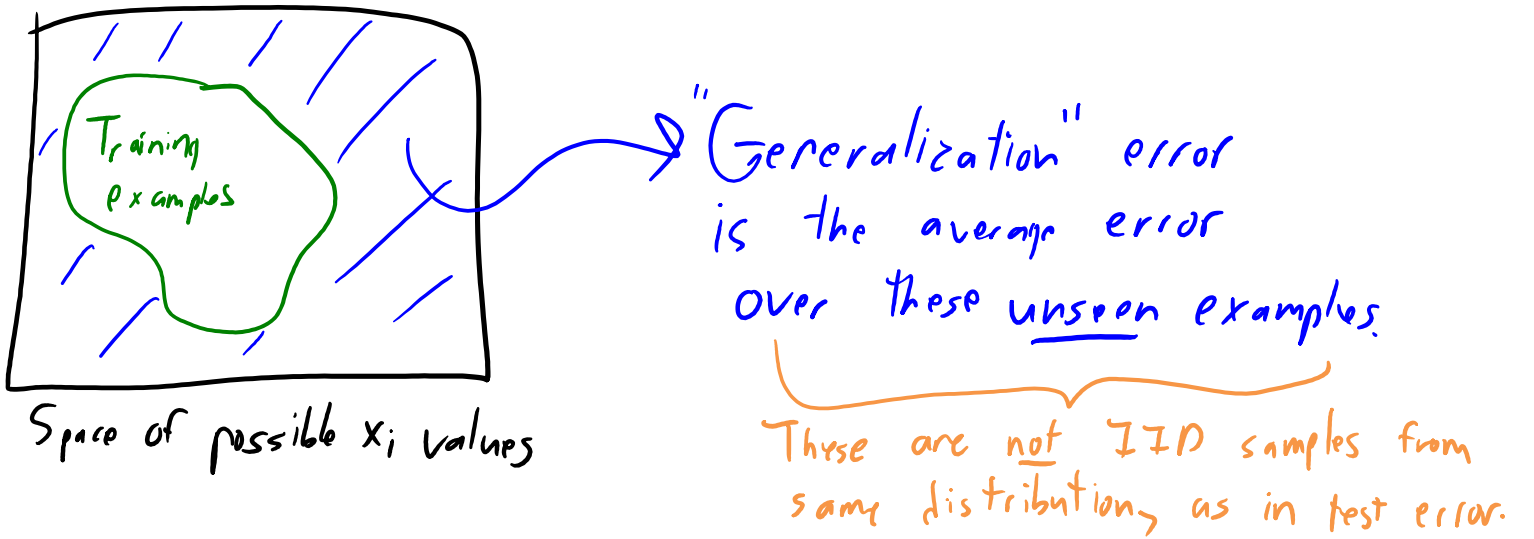
- You can take this idea further ("k-fold cross-validation"):
  - 10-fold cross-validation: train on 90% of data and validate on 10%.
    - Repeat 10 times and average (test on fold 1, then fold 2,…, then fold 10),
  - Leave-one-out cross-validation: train on all but one training example.
    - Repeat 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.

Next Topic: Probabilistic Classifiers

Generalization Error

- An alternative to test error is the generalization error:
  - Average error over all x_i vectors that are not seen in the training set.
    - Assuming each unseen x_i is equally probable.
    - "Error averaged over all *completely unseen* feature vectors".
  - Different than test error, which assumes IID data from same distribution.
    - Test error allows some x_i to be more probable, and does not exclude training x_i.



The “Best” Machine Learning Model

- Decision trees are not always most accurate on test error.
- What is 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”.

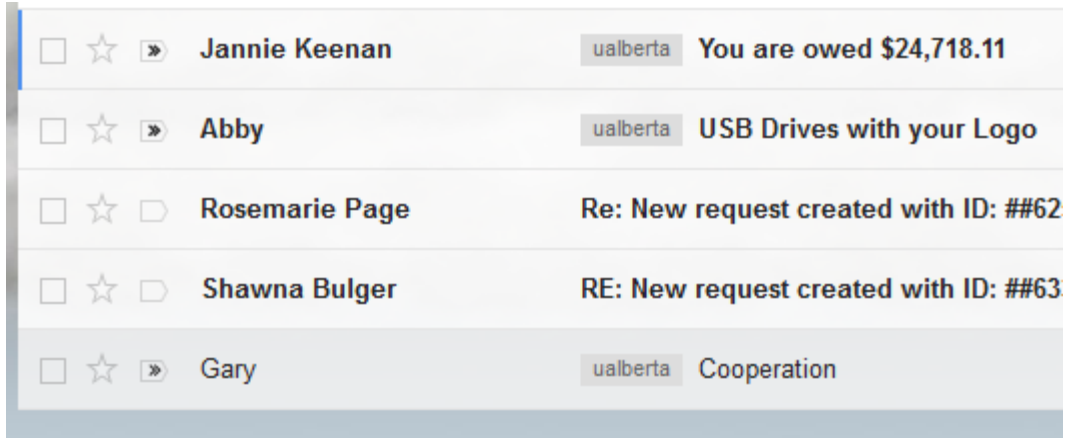
The “Best” Machine Learning Model

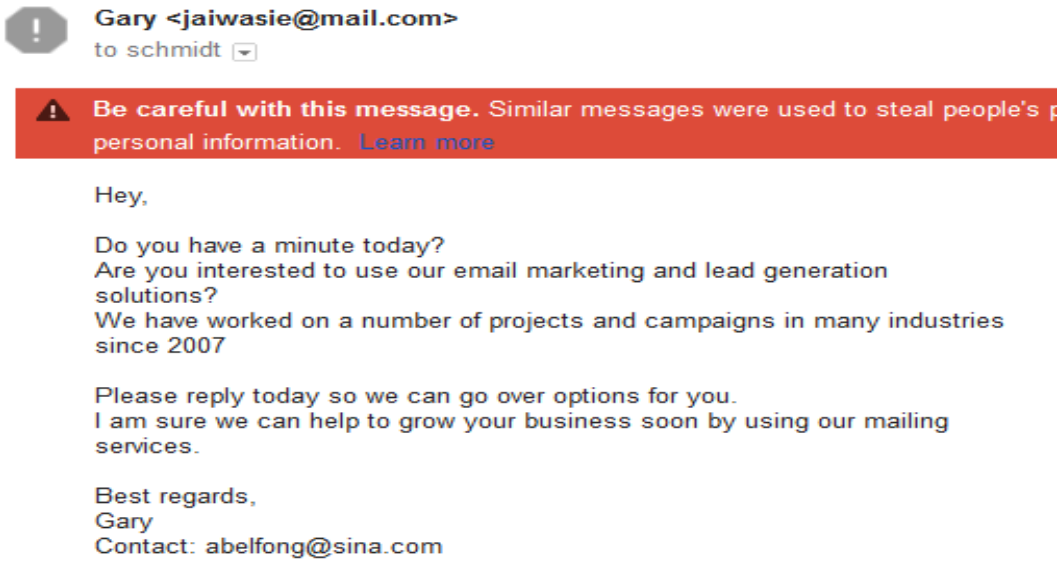
- Implications of the lack of a “best” model:
  - We need to learn about and try out multiple models.
- So which ones to study in CPSC 340?
  - We’ll usually motivate each method by a specific application.
  - But we’re focusing on models that have been effective in many applications.

- Caveat of no free lunch (NFL) theorem:
  - The world is very structured.
    - But proof of the no-free-lunch theorem assumes any map from x_i to y_i is equally likely.
  - Some datasets are more likely than others.
  - Model A really could be better than model B on every real dataset in practice.
- Machine learning research:
  - Large focus on models that are useful across many applications.

Application: E-mail Spam Filtering

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



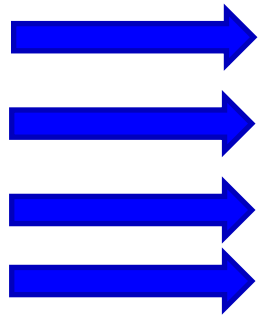


- Can we formulate as supervised learning?

Spam Filtering as Supervised Learning

- Collect a large number of e-mails, gets users to label them.

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



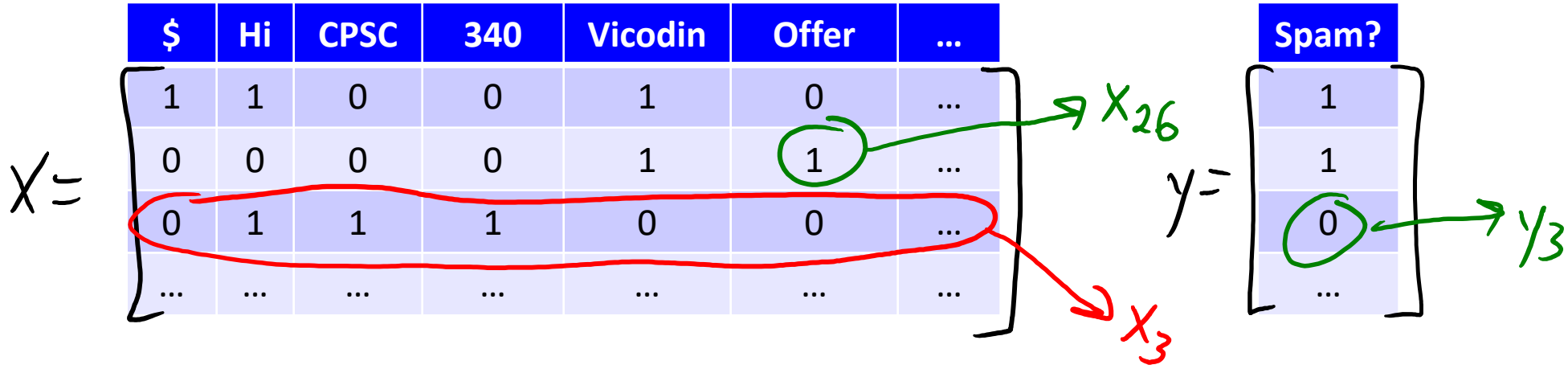
- We can use ($y_i = 1$) if e-mail 'i' is spam, ($y_i = 0$) if e-mail is not spam.
- Extract features of each e-mail (like bag of words).
  - ($x_{ij} = 1$) if word/phrase 'j' is in e-mail 'i', ($x_{ij} = 0$) if it is not.

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>.

Review: Supervised Learning Notation (MEMORIZE)

- We have been using the notation 'X' and 'y' for supervised learning:



- X is matrix of all features, y is vector of all labels.
  - We use y_i for the label of example 'i' (element 'i' of 'y').
  - We use x_{ij} for feature 'j' of example 'i'.
  - We use x_i as the list of features of example 'i' (row 'i' of 'X').
    - So in the above x_3 = [0 1 1 1 0 0 …].
    - In practice, only store list of non-zero features for each x_i (small memory requirement).

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 estimate conditional probability $p(y_i \mid x_{i1}, x_{i2}, \ldots, x_{id})$.
  - "If a message has words $x_i = [x_{i1}, x_{i2}, \ldots, x_{id}]$, what is probability that message is spam?"
    - When we use commas (',') in probability statements, it should be read as an "AND".

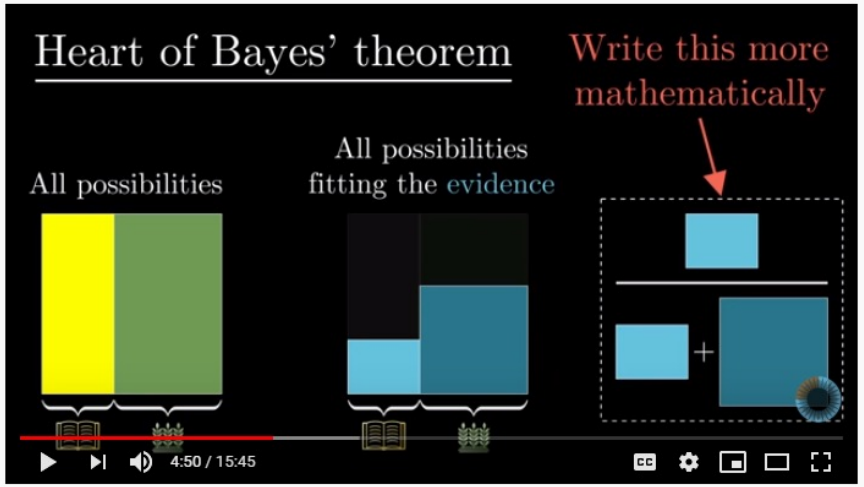
- Classify it as spam if probability of spam is higher than not spam:
  - If $p(y_i = \text{"spam"} \mid x_{i1}, x_{i2}, \ldots, x_{id}) > p(y_i = \text{"not spam"} \mid x_{i1}, x_{i2}, \ldots, x_{id})$
    - return "spam".
  - Else
    - return "not spam".

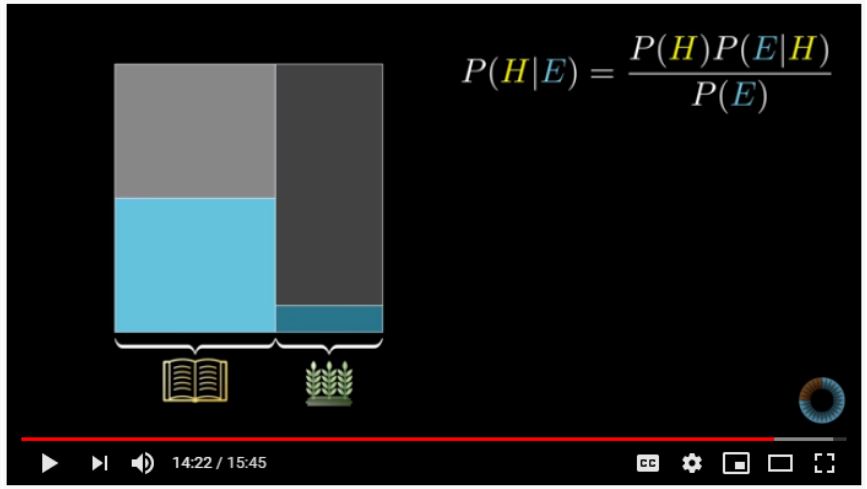
Spam Filtering with Bayes Rule

- To model conditional probability, naïve Bayes uses Bayes rule:

$$p(y_i = \text{"spam"} \mid x_{i1}, x_{i2}, \ldots, x_{id}) = \frac{p(x_{i1}, x_{i2}, \ldots, x_{id} \mid y_i = \text{"spam"})\, p(y_i = \text{"spam"})}{p(x_{i1}, x_{i2}, \ldots, x_{id})}$$

- Nice video giving visual intuition for Bayes rule here:





Spam Filtering with Bayes Rule

- To model conditional probability, naïve Bayes uses Bayes rule:

$$p(y_i = \text{"spam"} \mid x_{i1}, x_{i2}, \dots, x_{id}) = \frac{p(x_{i1}, x_{i2}, \dots, x_{id} \mid y_i = \text{"spam"})\, p(y_i = \text{"spam"})}{p(x_{i1}, x_{i2}, \dots, x_{id})}$$

- On the right we have three terms:
  - Probability $p(y_i)$ that an e-mail is spam.
  - Probability $p(x_{i1}, x_{i2}, \dots, x_{id})$ that an e-mail has the set of words x_i.
  - Conditional $p(x_{i1}, x_{i2}, \dots, x_{id} \mid y_i = \text{“spam”})$ that spam e-mail has the words x_i.
    - And you need the same for non-spam e-mails.
- We do not know any of these probabilities.
  - We train the model by estimating these probabilities from training data.

Spam Filtering with Bayes Rule

$$p(y_i = \text{"spam"} \mid x_{i1}, x_{i2}, \ldots, x_{id}) = \frac{p(x_{i1}, x_{i2}, \ldots, x_{id} \mid y_i = \text{"spam"})\, p(y_i = \text{"spam"})}{p(x_{i1}, x_{i2}, \ldots, x_{id})}$$

- What do these terms mean?

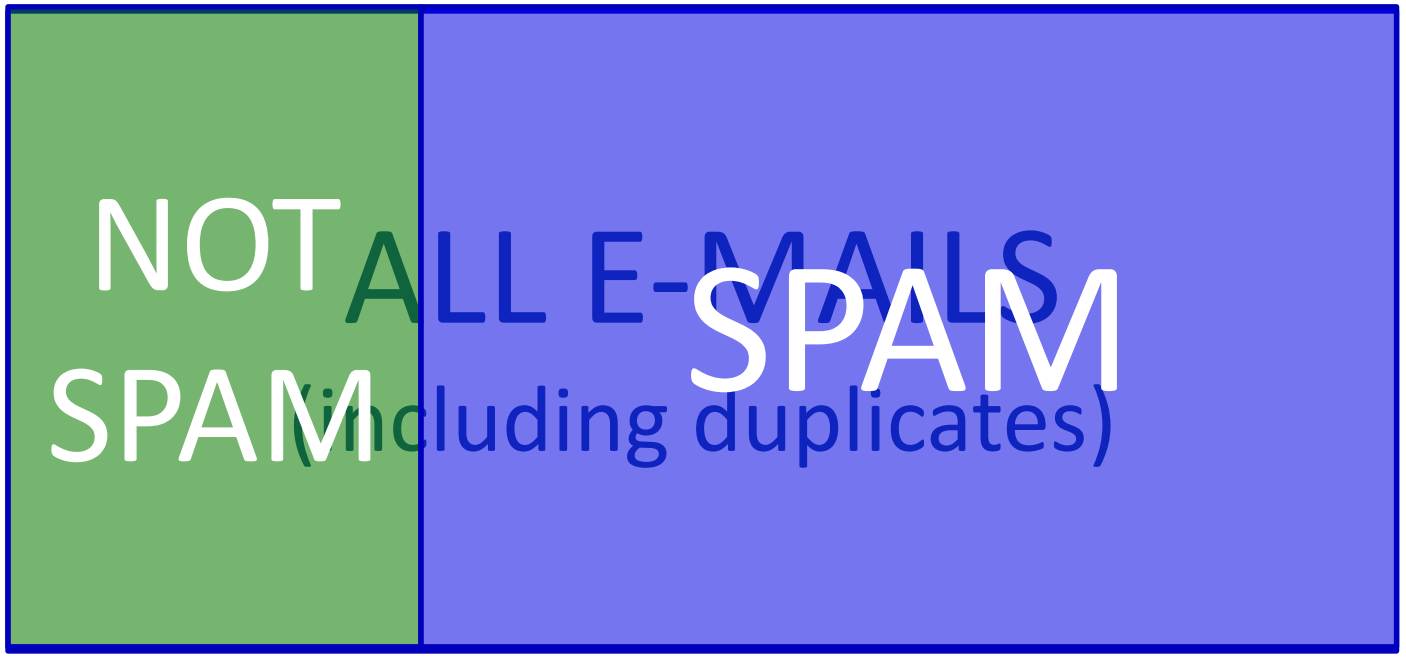
ALL E-MAILS

(including duplicates)

Spam Filtering with Bayes Rule

$$p(y_i = \text{"spam"} \mid x_{i1}, x_{i2}, \ldots, x_{id}) = \frac{p(x_{i1}, x_{i2}, \ldots, x_{id} \mid y_i = \text{"spam"})\, p(y_i = \text{"spam"})}{p(x_{i1}, x_{i2}, \ldots, x_{id})}$$

- $p(y_i$ = "spam") is probability that a random e-mail is spam.
  - This is easy to approximate from data: use the proportion in your data.



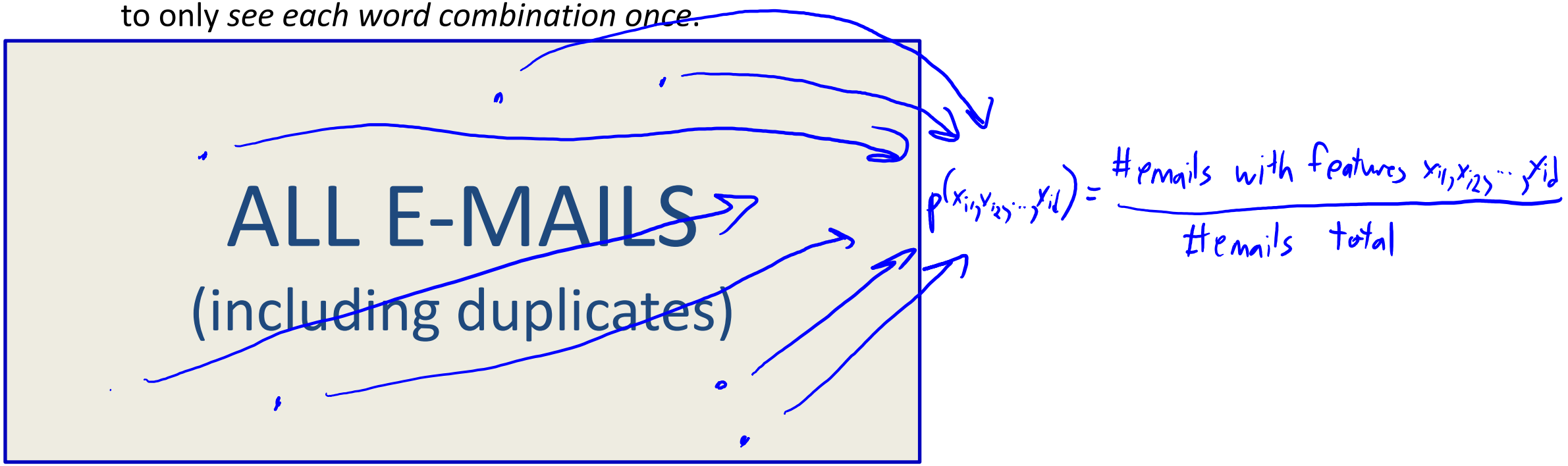
$$p(y_i = \text{"spam"}) = \frac{\#\text{ spam messages}}{\#\text{ total messages}}$$

This is an "estimate" of the true probability. In particular, this formula is a "maximum likelihood estimate" (MLE). We will cover likelihoods and MLEs later in the course.

Spam Filtering with Bayes Rule

$$p(y_i = \text{"spam"} \mid x_{i1}, x_{i2}, \ldots, x_{id}) = \frac{p(x_{i1}, x_{i2}, \ldots, x_{id} \mid y_i = \text{"spam"})\, p(y_i = \text{"spam"})}{p(x_{i1}, x_{i2}, \ldots, x_{id})}$$

- $p(x_{i1}, x_{i2}, \ldots, x_{id})$ is joint probability that a random e-mail has features x_i:
  - Hard to approximate: with 'd' words we need to collect 2^d "coupons",
    to only *see each word combination once*.



Spam Filtering with Bayes Rule

$$p(y_i = \text{"spam"} \mid x_{i1}, x_{i2}, \ldots, x_{id}) = \frac{p(x_{i1}, x_{i2}, \ldots, x_{id} \mid y_i = \text{"spam"})\, p(y_i = \text{"spam"})}{p(x_{i1}, x_{i2}, \ldots, x_{id})}$$

- $p(x_{i1}, x_{i2}, \ldots, x_{id})$ is joint probability that a random e-mail has features x_i :
  - Hard to approximate: with 'd' words we need to collect 2^d "coupons",
    but it turns out we can ignore it:

Naive Bayes returns "spam" if $p(y_i = \text{"spam"} \mid x_{i1}, x_{i2}, \ldots, x_{id}) > p(y_i = \text{"not spam"} \mid x_{i1}, x_{i2}, \ldots, x_{id})$

By Bayes rule this means $\frac{p(x_{i1}, x_{i2}, \ldots, x_{id} \mid y_i = \text{"spam"})\, p(y_i = \text{"spam"})}{p(x_{i1}, x_{i2}, \ldots, x_{id})} > \frac{p(x_{i1}, x_{i2}, \ldots, x_{id} \mid y_i = \text{"not spam"})\, p(y_i = \text{"not spam"})}{p(x_{i1}, x_{i2}, \ldots, x_{id})}$

Multiply both sides by $p(x_{i1}, x_{i2}, \ldots, x_{id})$:

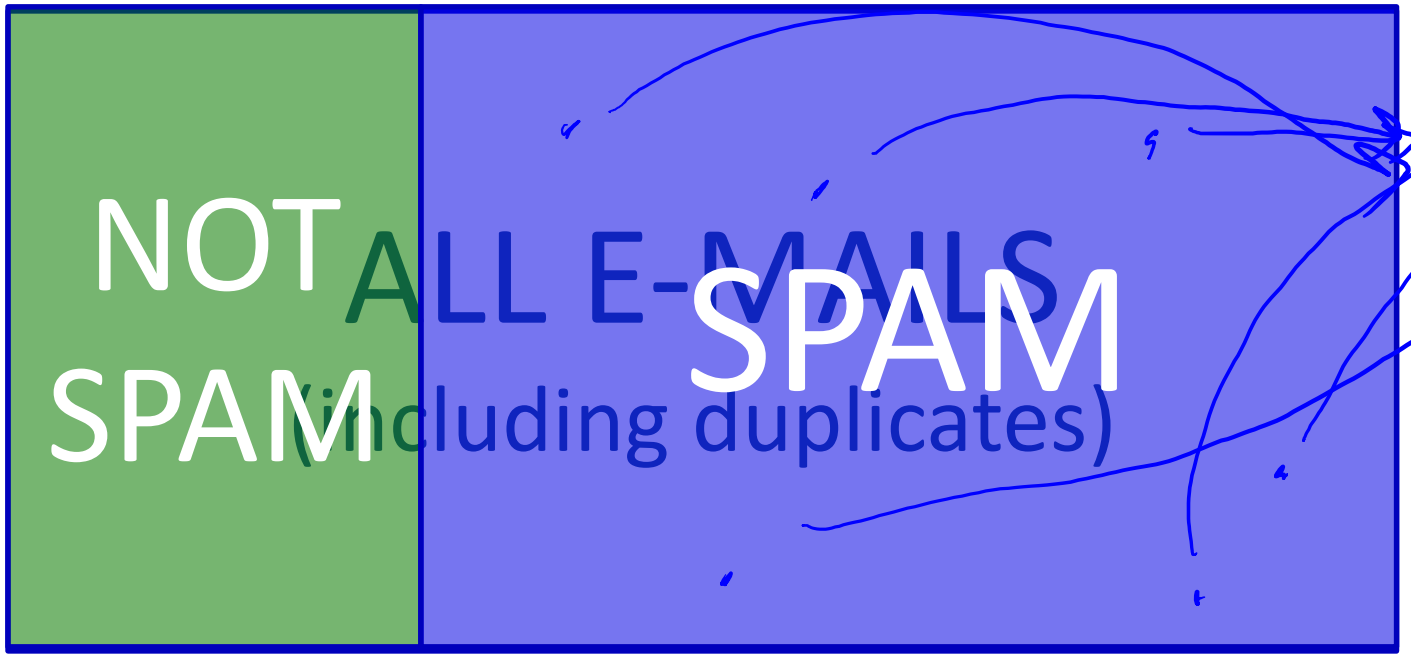
$$p(x_{i1}, x_{i2}, \ldots, x_{id} \mid y_i = \text{"spam"})\, p(y_i = \text{"spam"}) > p(x_{i1}, x_{i2}, \ldots, x_{id} \mid y_i = \text{"not spam"})\, p(y_i = \text{"not spam"})$$

No need to know $p(x_{i1}, x_{i2}, \ldots, x_{id})$.

Spam Filtering with Bayes Rule

$$p(y_i = \text{"spam"} \mid x_{i1}, x_{i2}, \ldots, x_{id}) = \frac{p(x_{i1}, x_{i2}, \ldots, x_{id} \mid y_i = \text{"spam"})\, p(y_i = \text{"spam"})}{p(x_{i1}, x_{i2}, \ldots, x_{id})}$$

- $p(x_{i1}, x_{i2}, \ldots, x_{id} \mid y_i = \text{“spam”})$ is probability that spam has features x_i.



$$p(x_{i1}, x_{i2}, \ldots, x_{id} \mid y_i = \text{"spam"}) = \frac{\#\text{ spam messages with features } x_i}{\#\text{ spam messages}}$$

- Also hard to approximate.
  - And we need it.

Naïve Bayes

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

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

HARD

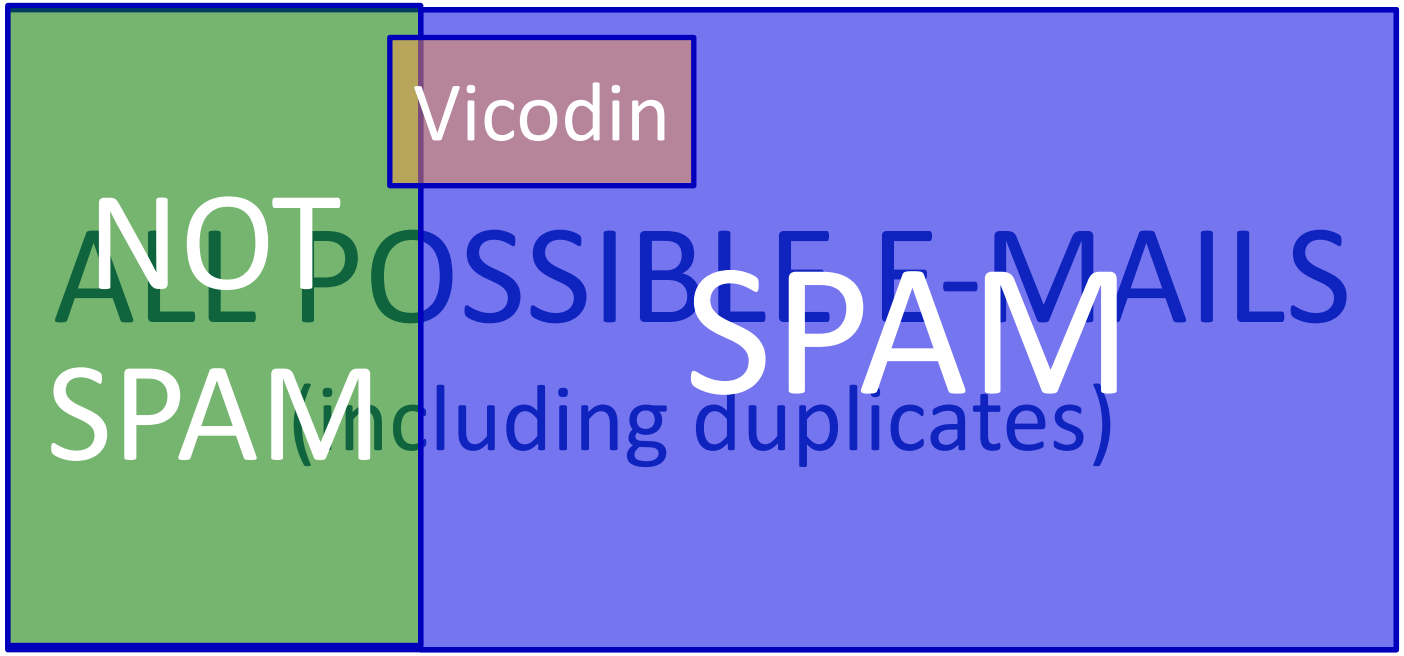
easy easy easy

- 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").

Naïve Bayes

- p("vicodin" = 1 | "spam" = 1) is probability of seeing "vicodin" in spam.



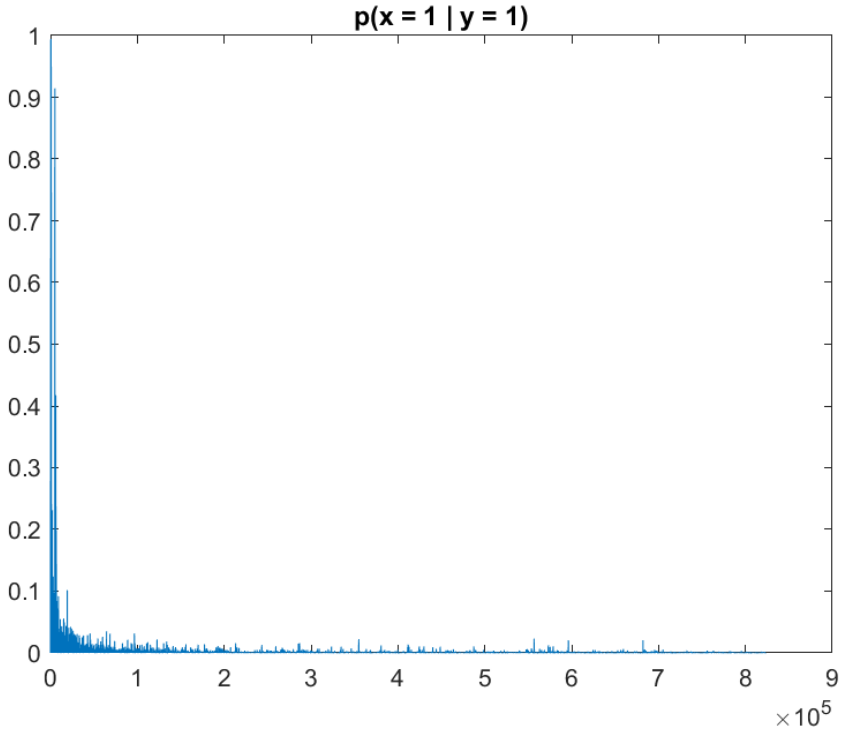
- Easy to estimate:

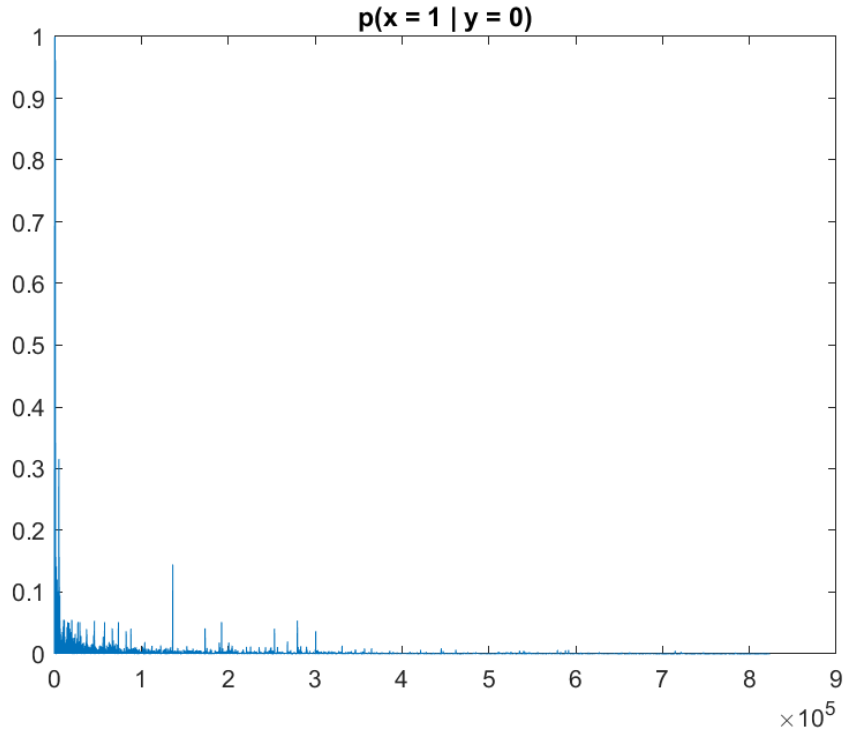
$$p(\text{vicodin}=1 \mid \text{spam}=1) = \frac{\#\text{ spam messages w/ vicodin}}{\#\text{ spam messages}}$$

Again, this is a "maximum likelihood estimate" (MLE). We will cover how to derive this later.

Naïve Bayes

- Comparing p(x | y = c) for “spam” and “not spam”:





- Even though independence is not true,
  these values may be enough to distinguish the classes.

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 $p(y_i \mid x_{i1}, x_{i2},\ldots,x_{id})$.
- 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 ∞.

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".

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.

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:

$$E_{test} = \mathbb{E}\left[|\hat{y}^i - \tilde{y}^i|\right]$$

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

$$E_{generalize} = \frac{1}{t}\sum_{x^i \notin \{train\ set\}} |\hat{y}_i - \tilde{y}_i|$$

number of x^i values not in training set.

average error over unseen x^i values.

Generalization Error

Let's do an example. Suppose you have two binary features and deterministic labels, and these features appear with the following probabilities:
Case 1: $x_i = [0, 0], y_i = 1$ (25% of the time).
Case 2: $x_i = [0, 1], y_i = 1$ (25% of the time).
Case 3: $x_i = [1, 0], y_i = 0$ (10% of the time).
Case 4: $x_i = [1, 1], y_i = 1$ (40% of the time).

In this case, no matter what your training dataset is, the test error is given by:
0.25*(wrong prediction for case 1?) + 0.25*(wrong prediction in case 2?) + 0.10*(wrong prediction in case 3?) + 0.40*(wrong prediction in case 4?),

where "wrong prediction in case x?" is 1 if you predicted wrong and 0 if you predicted correctly.

On the other hand, the generalization error ignores the distribution's probabilities and takes the average weighted error over the feature sets not seen during training. So if your training set just consists of Case 2, the generalization error would be:
0.33*(wrong prediction for case 1?) + 0.33*(wrong prediction in case 3?) + 0.33*(wrong prediction in case 4?).

If you training set contains Case 2 and Case 3 (even with repeats), the generalization error would be:
0.50*(wrong prediction for case 1?) + 0.50*(wrong prediction in case 4?).

And if you see all four cases then the generalization error is not defined.

Generalization error is different than test error because it depends on what data you saw during training (so it is hard to compare generalization error of models trained on different datasets). Generalization error also does not take into account that some feature vectors and more likely than others, which could be good or bad depending on the application (bad in cases like images where a random setting of pixels will produce a non-sense image, but good in cases like security where you want to protect against weird inputs that might exist).

“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

- 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 'ℓ' classes, the average error is (ℓ-1)/ℓ (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".