CPSC 340:
Machine Learning and Data Mining

Probabilistic Classification

Fall 2018

Admin

- Course webpage: www.ugrad.cs.ubc.ca/~cs340
- Assignment 1 is due tonight.
  - You can use 1 of your 2 late days to submit it up to 48 hours late.

- Waiting list people: you are registered.
  - The other section of 340 has space.
- Graduate students who don't need 500-level credit:
  - You should now be able to sign up for 340 (no project)?
- Auditing: message me on Piazza if you want to audit.
  - Bring your forms to me in class Friday/Monday.

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 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.

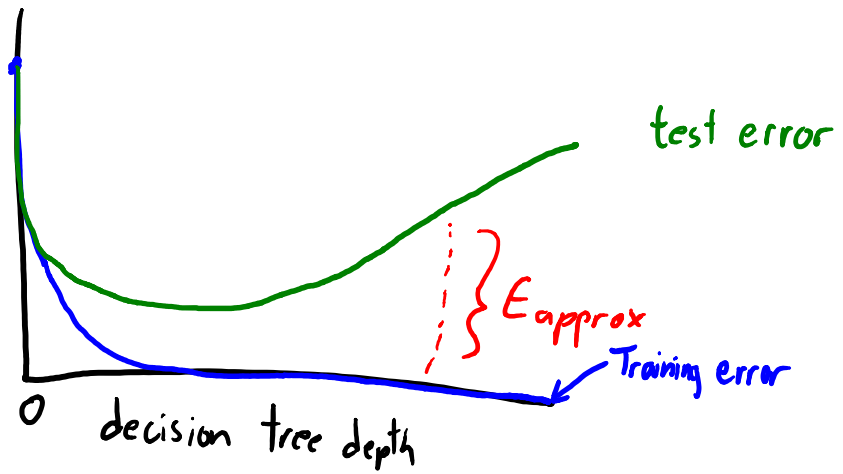
Last Time: Fundamental Trade-Off

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

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

"test error" "approximation error" "training error"

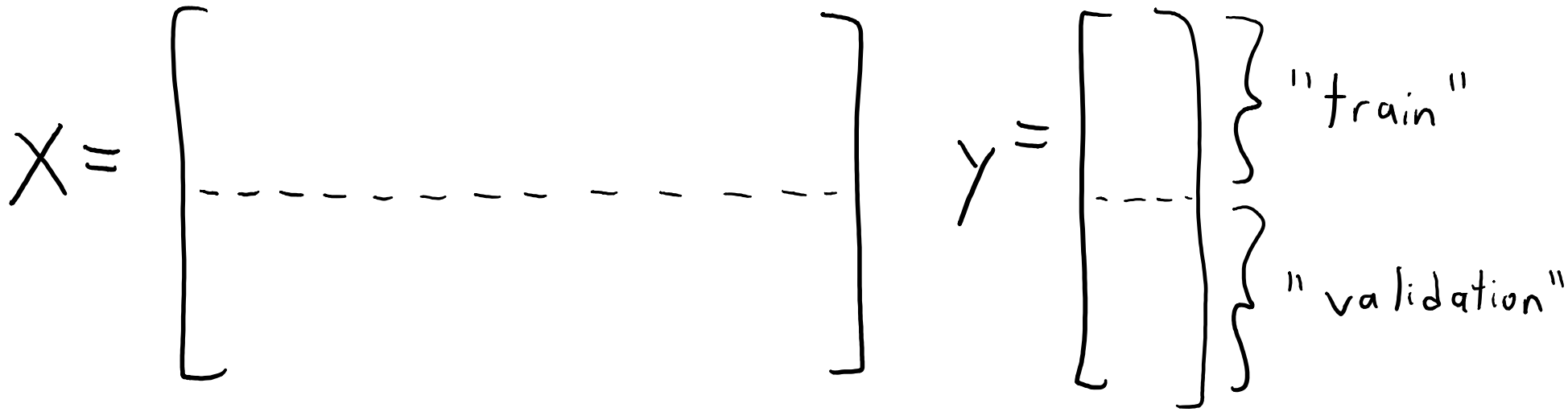
  - Where $E_{approx} = (E_{test} - E_{train})$.



- 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 Errorr

- 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.

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).

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.
  - Maximizing over these errors is a biased approximation of test error.
  - 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.

Validation Error and Optimization Bias

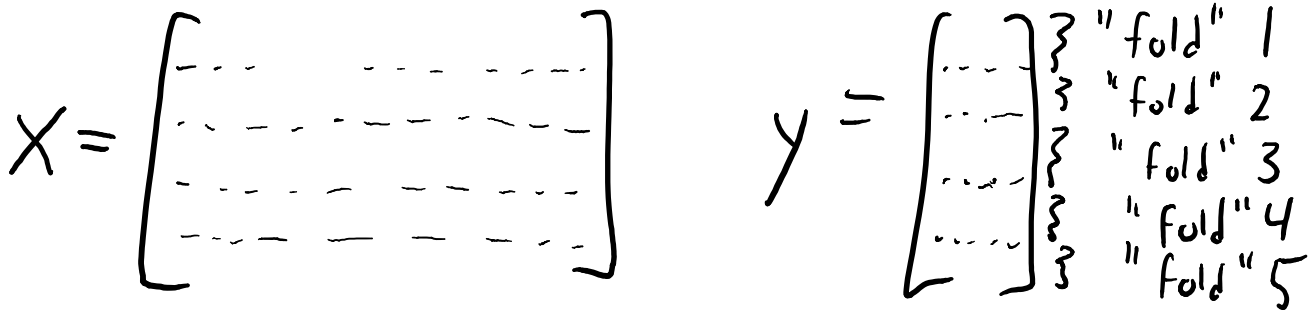
- Optimization bias is small if you only compare a few models:
  - Best decision tree on the training set among depths, 1, 2, 3,…, 10.
  - Risk of overfitting to validation set is low if we try 10 things.

- 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.

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 more times with 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)

- You can take this idea further:
  - 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 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

(pause)

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 vectos that are not seen in the training set.
  - “How well we expect to do for a *completely unseen* feature vector”.
- 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”.

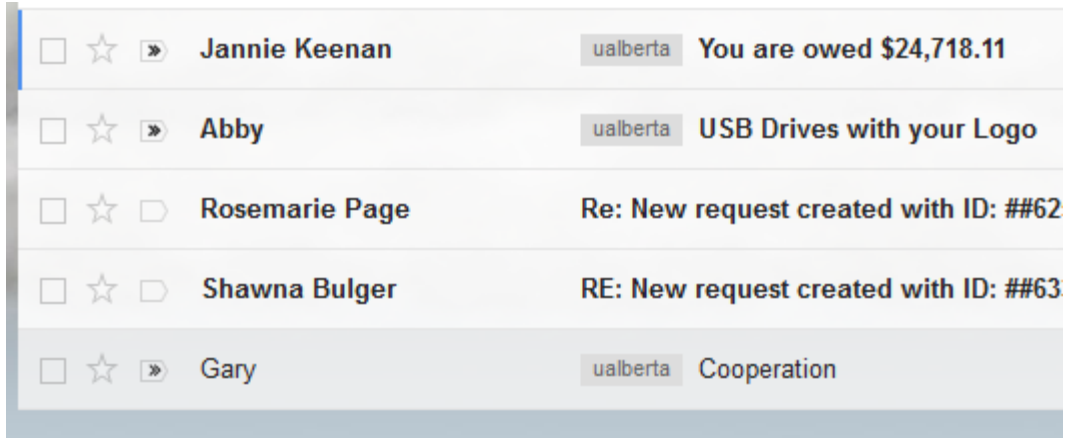
The “Best” Machine Learning Model

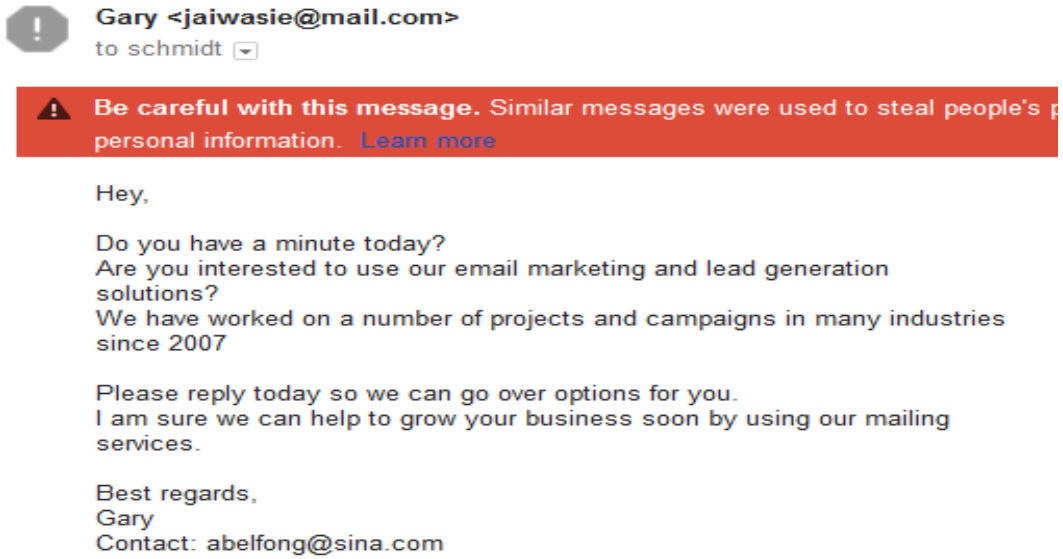
- Implications of the lack of a “best” model:
  - We need to learn about and try out multiple models.
- So which ones to study in CPSC 340?
  - We’ll usually motivate each method by a specific application.
  - But we’re focusing on models that have been effective in many applications.

- Caveat of no free lunch (NFL) theorem:
  - The world is very structured.
  - 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.



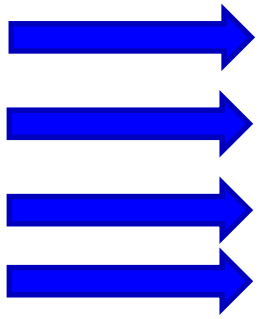


- Can we formulate as supervised learning?

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.

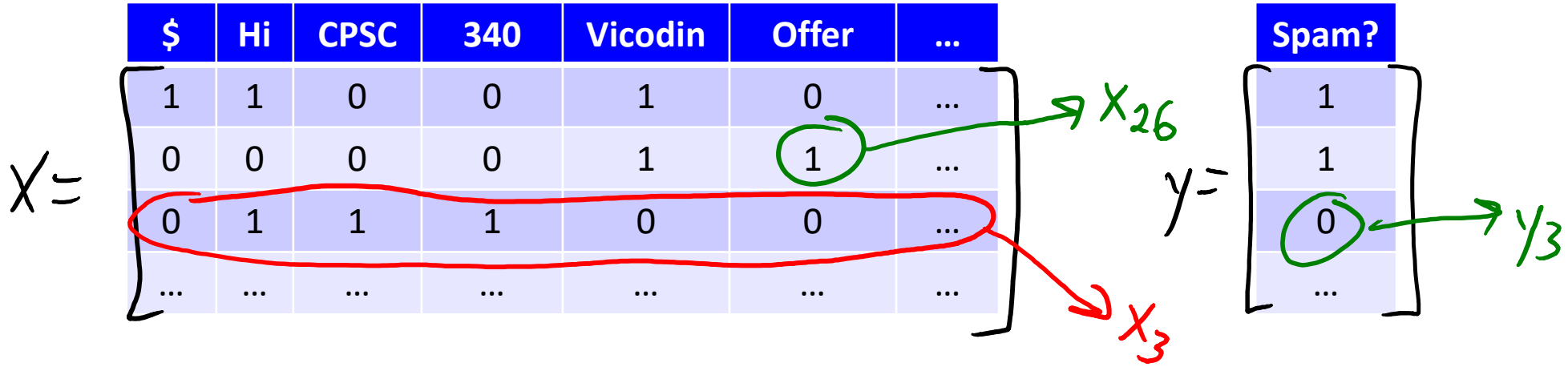
Feature Representation for Spam

- 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>.

- Also, note that we only need list of non-zero features for each x_i.

Review of Supervised Learning Notation

- 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 …].

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.
  - Last year shown to improve on state of the art for CRISPR "gene editing" (link).

- Probabilistic classifiers 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 has 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".

Spam Filtering with Bayes Rule

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

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

- So we need to figure out three types of terms:
  - Marginal probabilities $p(y_i)$ that an e-mail is spam.
  - Marginal probability $p(x_i)$ that an e-mail 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.

Spam Filtering with Bayes Rule

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

- What do these terms mean?

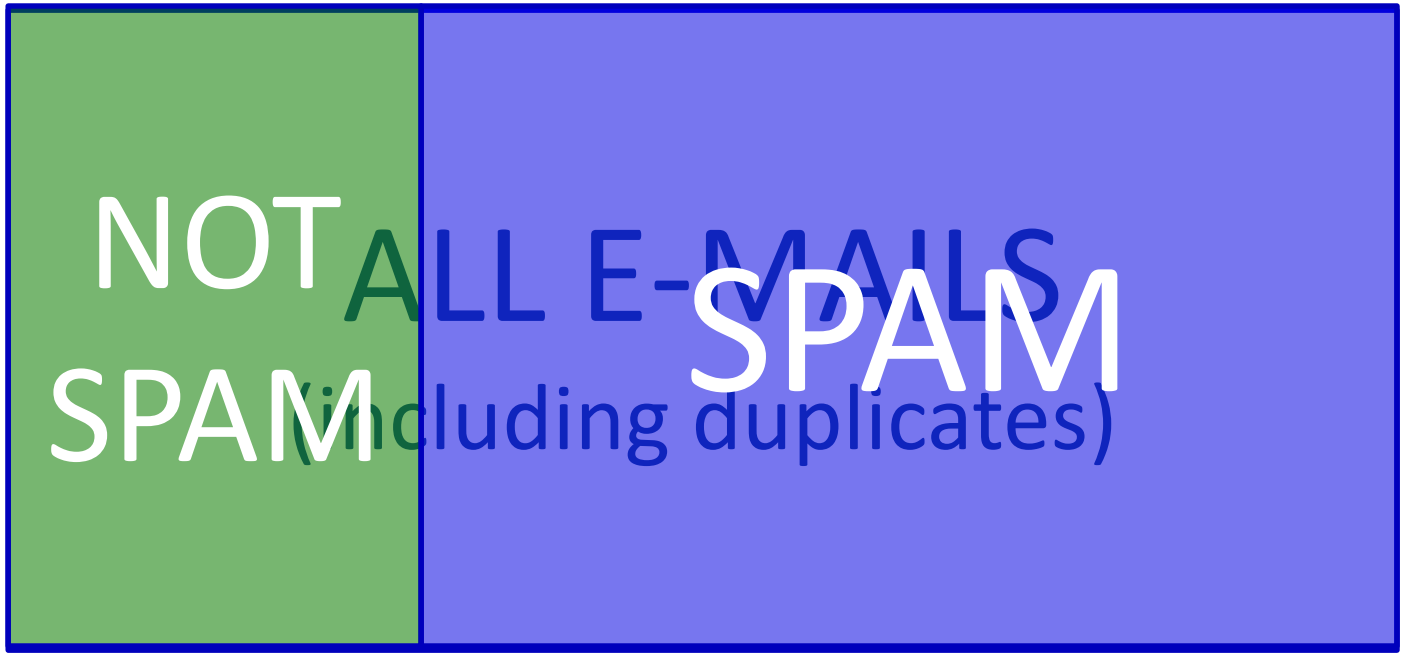
ALL E-MAILS

(including duplicates)

Spam Filtering with Bayes Rule

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

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



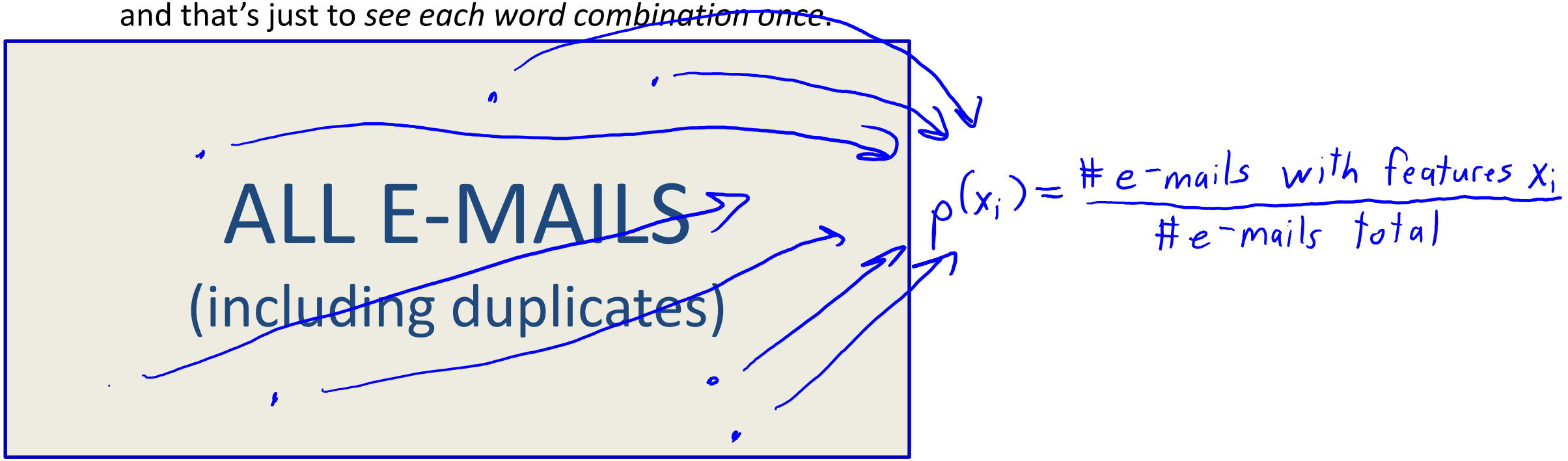
$$p(y_i = \text{"spam"}) = \frac{\#\text{ spam messages}}{\#\text{ total messages}}$$

This is a "maximum likelihood estimate", a concept we'll discuss in detail later. If you're interested in a proof, see here.

Spam Filtering with Bayes Rule

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

- $p(x_i)$ is probability that a random e-mail has features x_i:
  - Hard to approximate: with 'd' words we need to collect 2^d "coupons",
    and that's just to *see each word combination once*.



Spam Filtering with Bayes Rule

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

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

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

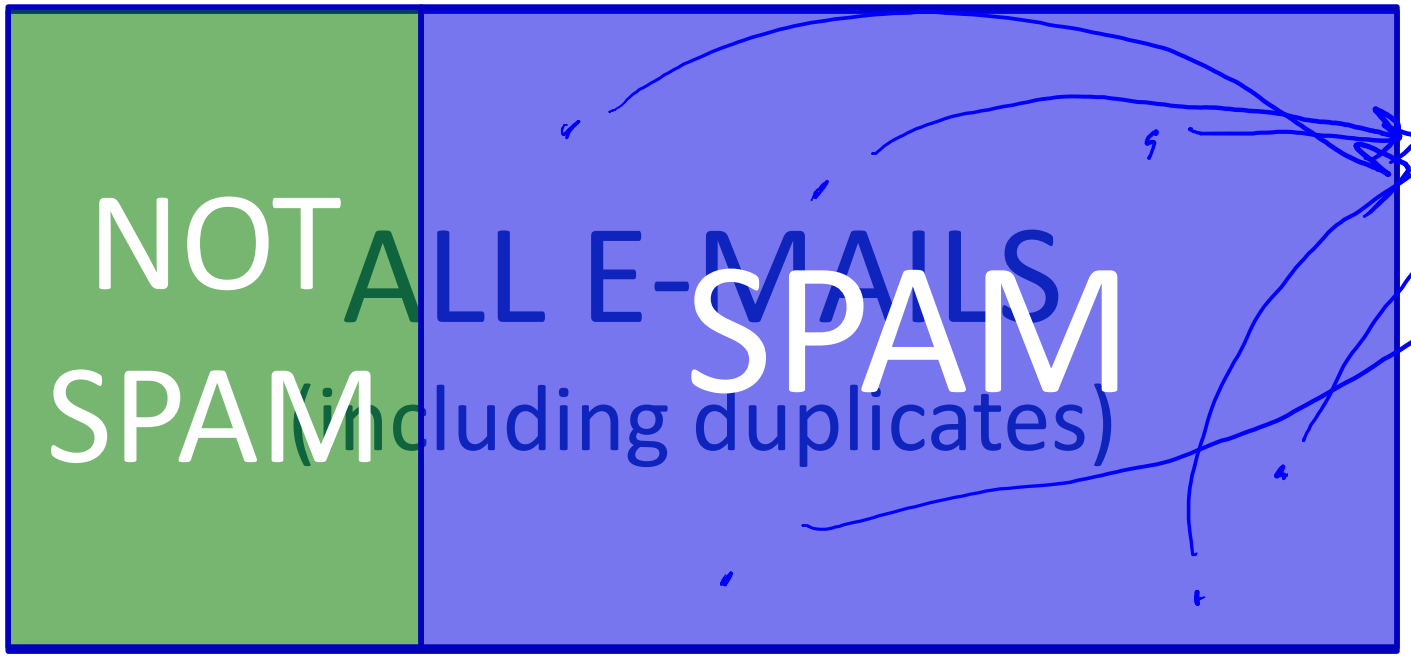
Multiply both sides by $p(x_i)$:

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

Spam Filtering with Bayes Rule

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

- $p(x_i \mid y_i = \text{“spam”})$ is probability that spam has features x_i.



$$p(x_i \mid y_i = \text{"spam"}) = \frac{\#\text{ spam messages with features } x_i}{\#\text{ spam messages}}$$

- 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

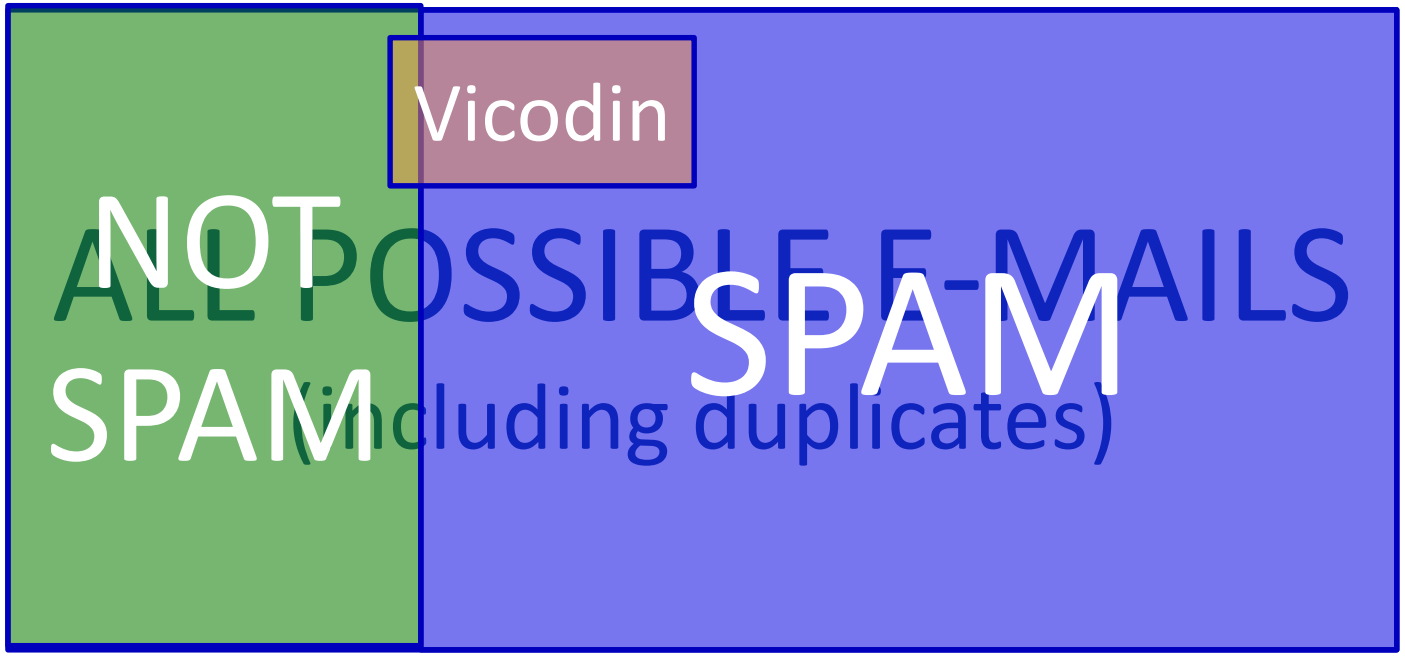
easy easy easy

- 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:

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

Naïve Bayes

- Naïve Bayes more formally:

$$p(y_i \mid x_i) = \frac{p(x_i \mid y_i)p(y_i)}{p(x_i)} \quad \text{(first use Bayes rule)}$$

$$\propto p(x_i \mid y_i)\, p(y_i) \quad \text{("denominator doesn't matter")}$$

$$\approx \prod_{j=1}^{d} \left[p(x_{ij} \mid y_i) \right] p(y_i) \quad \text{(conditional independence assumption)}$$

Only needs easy probabilities.

- Post-lecture slides: how to train/test by hand on a simple example.

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 ∞.

Naïve Bayes Training Phase

- Training a naïve Bayes model:

$$X = \begin{bmatrix} 0 & 1 \\ 1 & 1 \\ 0 & 0 \\ 1 & 1 \\ 1 & 1 \\ 0 & 0 \\ 1 & 0 \\ 1 & 0 \\ 1 & 1 \\ 1 & 0 \end{bmatrix}, \quad y = \begin{bmatrix} 1 \\ 1 \\ 1 \\ 1 \\ 1 \\ 1 \\ 0 \\ 0 \\ 0 \\ 0 \end{bmatrix}$$

Naïve Bayes Training Phase

- Training a naïve Bayes model:

1. Set n_c to the number of times $(y_i = c)$.

$$X = \begin{bmatrix} 0 & 1 \\ 1 & 1 \\ 0 & 0 \\ 1 & 1 \\ 1 & 1 \\ 0 & 0 \\ 1 & 0 \\ 1 & 0 \\ 1 & 1 \\ 1 & 0 \end{bmatrix}, \quad y = \begin{bmatrix} 1 \\ 1 \\ 1 \\ 1 \\ 1 \\ 1 \\ 0 \\ 0 \\ 0 \\ 0 \end{bmatrix}$$

$n_1 = 6$

$n_0 = 4$

Naïve Bayes Training Phase

- Training a naïve Bayes model:

1. Set n_c to the number of times $(y_i = c)$.
2. Estimate $p(y_i = c)$ as $\frac{n_c}{n}$.

$$X = \begin{bmatrix} 0 & 1 \\ 1 & 1 \\ 0 & 0 \\ 1 & 1 \\ 1 & 1 \\ 0 & 0 \\ 1 & 0 \\ 1 & 0 \\ 1 & 1 \\ 1 & 0 \end{bmatrix}, \quad y = \begin{bmatrix} 1 \\ 1 \\ 1 \\ 1 \\ 1 \\ 1 \\ 0 \\ 0 \\ 0 \\ 0 \end{bmatrix}$$

$p(y_i = 1) = \frac{6}{10} \leftarrow n_1 = 6$

$p(y_i = 0) = \frac{4}{10} \leftarrow n_0 = 4$

Naïve Bayes Training Phase

- Training a naïve Bayes model:

1. Set n_c to the number of times $(y_i = c)$.
2. Estimate $p(y_i = c)$ as $\frac{n_c}{n}$.
3. Set n_{cjk} as the number of times $(y_i = c, x_{ij} = k)$

$$X = \begin{bmatrix} 0 & 1 \\ 1 & 1 \\ 0 & 0 \\ 1 & 1 \\ 1 & 1 \\ 0 & 0 \\ 1 & 0 \\ 1 & 0 \\ 1 & 1 \\ 1 & 0 \end{bmatrix}, \quad y = \begin{bmatrix} 1 \\ 1 \\ 1 \\ 1 \\ 1 \\ 1 \\ 0 \\ 0 \\ 0 \\ 0 \end{bmatrix}$$

$p(y_i = 1) = \frac{6}{10}$ ← $n_1 = 6$

$n_{121} = 4$

$n_0 = 4$

$p(y_i = 0) = \frac{4}{10}$

Naïve Bayes Training Phase

- Training a naïve Bayes model:

1. Set n_c to the number of times $(y_i = c)$.
2. Estimate $p(y_i = c)$ as $\frac{n_c}{n}$.
3. Set n_{cjk} as the number of times $(y_i = c, x_{ij} = k)$
4. Estimate $p(x_{ij} = k, y_i = c)$ as $\frac{n_{cjk}}{n}$.

$$X = \begin{bmatrix} 0 & 1 \\ 1 & 1 \\ 0 & 0 \\ 1 & 1 \\ 1 & 1 \\ 0 & 0 \\ 1 & 0 \\ 1 & 0 \\ 1 & 1 \\ 1 & 0 \end{bmatrix}, \quad y = \begin{bmatrix} 1 \\ 1 \\ 1 \\ 1 \\ 1 \\ 1 \\ 0 \\ 0 \\ 0 \\ 0 \end{bmatrix}$$

$p(y_i = 1) = \frac{6}{10} \leftarrow n_1 = 6$

$n_{121} = 4$

$p(x_{ij} = 1, y_i = 1) = \frac{4}{10}$

$n_0 = 4$

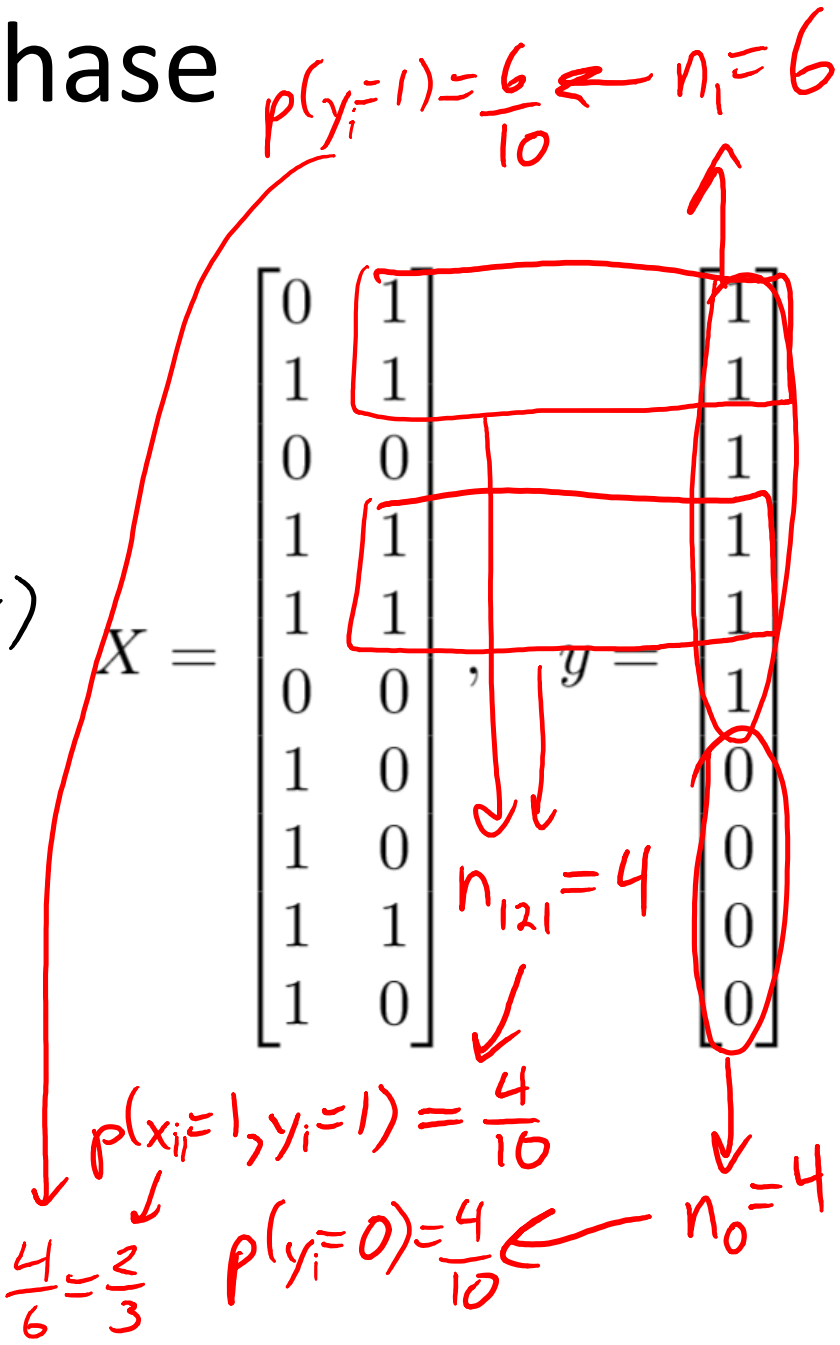
$p(y_i = 0) = \frac{4}{10}$

Naïve Bayes Training Phase

- Training a naïve Bayes model:

1. Set n_c to the number of times $(y_i = c)$.
2. Estimate $p(y_i = c)$ as $\frac{n_c}{n}$.
3. Set n_{cjk} as the number of times $(y_i = c, x_{ij} = k)$
4. Estimate $p(x_{ij} = k, y_i = c)$ as $\frac{n_{cjk}}{n}$.
5. Use that $p(x_{ij} = k \mid y_i = c) = \frac{p(x_{ij} = k, y_i = c)}{p(y_i = c)}$

$$= \frac{n_{cjk}/n}{n_c/n} = \frac{n_{cjk}}{n_c}$$



Naïve Bayes Prediction Phase

- Prediction in a naïve Bayes model:

Given a test example $\tilde{x}_i$ we set prediction $\hat{y}_i$ to the 'c' maximizing $p(\tilde{x}_i \mid \tilde{y}_i = c)$

Under the naive Bayes assumption we can maximize:

$$p(\tilde{y}_i = c \mid \tilde{x}_i) \propto \prod_{j=1}^{d} \left[p(\tilde{x}_{ij} \mid \tilde{y}_i = c) \right] p(\tilde{y}_i = c)$$

Naïve Bayes Prediction Phase

- Prediction in a naïve Bayes model:

Consider $\tilde{x}_i = [1\ 1]$ in this data set ⟶

$$X = \begin{bmatrix} 0 & 1 \\ 1 & 1 \\ 0 & 0 \\ 1 & 1 \\ 1 & 1 \\ 0 & 0 \\ 1 & 0 \\ 1 & 0 \\ 1 & 1 \\ 1 & 0 \end{bmatrix}, \quad y = \begin{bmatrix} 1 \\ 1 \\ 1 \\ 1 \\ 1 \\ 1 \\ 0 \\ 0 \\ 0 \\ 0 \end{bmatrix}$$

Naïve Bayes Prediction Phase

- Prediction in a naïve Bayes model:

Consider $\tilde{x}_i = [1\ 1]$ in this data set $\longrightarrow$

$$p(\tilde{y}_i = 0 \mid \tilde{x}_i) \propto p(\tilde{x}_{i1} = 1 \mid \tilde{y}_i = 0)\, p(\tilde{x}_{i2} = 1 \mid \tilde{y}_i = 0)\, p(\tilde{y}_i = 0)$$

$$= \quad (1) \qquad (0.25) \qquad (0.4) = 0.1$$

$$X = \begin{bmatrix} 0 & 1 \\ 1 & 1 \\ 0 & 0 \\ 1 & 1 \\ 1 & 1 \\ 0 & 0 \\ 1 & 0 \\ 1 & 0 \\ 1 & 1 \\ 1 & 0 \end{bmatrix}, \quad y = \begin{bmatrix} 1 \\ 1 \\ 1 \\ 1 \\ 1 \\ 1 \\ 0 \\ 0 \\ 0 \\ 0 \end{bmatrix}$$

Naïve Bayes Prediction Phase

- Prediction in a naïve Bayes model:

Consider $\tilde{x}_i = [1\ 1]$ in this data set $\longrightarrow$

$$p(\tilde{y}_i = 0 \mid \tilde{x}_i) \propto p(\tilde{x}_{i1} = 1 \mid \tilde{y}_i = 0)\, p(\tilde{x}_{i2} = 1 \mid \tilde{y}_i = 0)\, p(\tilde{y}_i = 0)$$
$$= (1)\quad (0.25)\quad (0.4) = 0.1$$

$$p(\tilde{y}_i = 1 \mid \tilde{x}_i) \propto p(\tilde{x}_{i1} = 1 \mid \tilde{y}_i = 1)\, p(\tilde{x}_{i2} = 1 \mid \tilde{y}_i = 1)\, p(\tilde{y}_i = 1)$$
$$= (0.5)\quad (0.666\ldots)\quad (0.6) = 0.2$$

$$X = \begin{bmatrix} 0 & 1 \\ 1 & 1 \\ 0 & 0 \\ 1 & 1 \\ 1 & 1 \\ 0 & 0 \\ 1 & 0 \\ 1 & 0 \\ 1 & 1 \\ 1 & 0 \end{bmatrix}, \quad y = \begin{bmatrix} 1 \\ 1 \\ 1 \\ 1 \\ 1 \\ 1 \\ 0 \\ 0 \\ 0 \\ 0 \end{bmatrix}$$

Naïve Bayes Prediction Phase

- Prediction in a naïve Bayes model:

Consider $\tilde{x}_i = [1\ 1]$ in this data set ⟶

$$p(\tilde{y}_i = 0 \mid \tilde{x}_i) \propto p(\tilde{x}_{i1} = 1 \mid \tilde{y}_i = 0)\, p(\tilde{x}_{i2} = 1 \mid \tilde{y}_i = 0)\, p(\tilde{y}_i = 0)$$
$$= (1)\quad (0.25)\quad (0.4) = 0.1$$

$$p(\tilde{y}_i = 1 \mid \tilde{x}_i) \propto p(\tilde{x}_{i1} = 1 \mid \tilde{y}_i = 1)\, p(\tilde{x}_{i2} = 1 \mid \tilde{y}_i = 1)\, p(\tilde{y}_i = 1)$$
$$= (0.5)\quad (0.666\ldots)\quad (0.6) = 0.2$$

$$X = \begin{bmatrix} 0 & 1 \\ 1 & 1 \\ 0 & 0 \\ 1 & 1 \\ 1 & 1 \\ 0 & 0 \\ 1 & 0 \\ 1 & 0 \\ 1 & 0 \\ 1 & 1 \\ 1 & 0 \end{bmatrix}, \quad y = \begin{bmatrix} 1 \\ 1 \\ 1 \\ 1 \\ 1 \\ 1 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \end{bmatrix}$$

Since $p(\tilde{y}_i = 1 \mid \tilde{x}_i)$ is bigger than $p(\tilde{y}_i = 0 \mid \tilde{x}_i)$, naive Bayes predicts $\hat{y}_i = 1$.

(Don't sum to 1 because we're ignoring $p(\tilde{x}_i)$)

Probability of Paying Back a Loan and Ethics

- Article discussing predicting "whether someone will pay back a loan":
  - https://www.thecut.com/2017/05/what-the-words-you-use-in-a-loan-application-reveal.html

- Words that increase probability of paying back the most:
  - *debt-free, lower interest rate, after-tax, minimum payment, graduate.*

- Words that decrease probability of paying back the most:
  - *God, promise, will pay, thank you, hospital.*

- Article also discusses an important issue: are all these features ethical?
  - Should you deny a loan because of religion or a family member in the hospital?
  - ICBC is limited in the features it is allowed to use for prediction.

Avoiding Underflow

- During the prediction, the probability can underflow:

$$p(y_i = c \mid x_i) \propto \prod_{j=1}^{d} \left[p(x_{ij} \mid y_i = c) \right] p(y_i = c)$$

All these are < 1 so the product gets very small!

  - Standard fix is to (equivalently) maximize the logarithm of the probability:

Rember that $\log(ab) = \log(a) + \log(b)$ so $\log(\prod a_i) = \sum_i \log(a_i)$

Since log is monotonic the 'c' maximizing $p(y_i = c \mid x_i)$ also maximizes $\log p(y_i = c \mid x_i)$,

so maximize $\log\left(\prod_{j=1}^{d} \left[p(x_{ij} \mid y_i = c) \right] p(y_i = c)\right) = \sum_{j=1}^{d} \log(p(x_{ij} \mid y_i = c)) + \log(p(y_i = c))$

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 CV to prune:
  - Remove leaf nodes that don't improve CV error.

- Popular implementations that have these tricks and others.

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, using 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.

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.

Less-Naïve Bayes

- Given features {x1,x2,x3,…,xd}, naïve Bayes approximates p(y|x) as:

$$p(y \mid x_1, x_2, \dots, x_d) \propto p(y)\, p(x_1, x_2, \dots, x_d \mid y)$$

product rule applied repeatedly

$$= p(y)\, p(x_1 \mid y)\, p(x_2 \mid x_1, y)\, p(x_3 \mid x_2, x_1, y) \cdots p(x_d \mid x_1, x_2, \dots, x_{d-1}, y)$$

$$\approx p(y)\, p(x_1 \mid y)\, p(x_2 \mid y)\, p(x_3 \mid y) \cdots p(x_d \mid y) \quad \text{(naive Bayes assumption)}$$

- The assumption is very strong, and there are "less naïve" versions:
  - Assume independence of all variables except up to 'k' largest 'j' where j < i.
    - E.g., naïve Bayes has k=0 and with k=2 we would have:

$$\approx p(y)\, p(x_1 \mid y)\, p(x_2 \mid x_1, y)\, p(x_3 \mid x_2, x_1, y)\, p(x_4 \mid x_3, x_2, y) \cdots p(x_d \mid x_{d-2}, x_{d-1}, y)$$

    - Fewer independence assumptions so more flexible, but hard to estimate for large 'k'.
  - Another practical variation is "tree-augmented" naïve Bayes.

Gaussian Discriminant Analysis

- Classifiers based on Bayes rule are called generative classifier:
  - They often work well when you have tons of features.
  - But they need to know $p(x_i \mid y_i)$, probability of features given the class.
    - How to "generate" features, based on the class label.

- To fit generative models, usually make BIG assumptions:
  - Naïve Bayes (NB) for discrete x_i:
    - Assume that each variables in x_i is independent of the others in x_i given y_i.
  - Gaussian discriminant analysis (GDA) for continuous x_i.
    - Assume that $p(x_i \mid y_i)$ follows a multivariate normal distribution.
    - If all classes have same covariance, it's called "linear discriminant analysis".

Computing $p(x_i)$ under naïve Bayes

- Generative models don't need $p(x_i)$ to make decisions.
- However, it's easy to calculate under the naïve Bayes assumption:

$$p(x_i) = \sum_{c=1}^{K} p(x_i, y=c) \qquad \text{(marginalization rule)}$$

$$= \sum_{c=1}^{K} p(x_i \mid y=c)\, p(y=c) \qquad \text{(product rule)}$$

$$= \sum_{c=1}^{K} \left[\prod_{j=1}^{d} p(x_{ij} \mid y=c) \right] p(y=c) \qquad \text{(naive Bayes assumption)}$$

These are the quantities we compute during training.

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: $\{0,1\}^d$ -> $\{0,1\}$

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".