CPSC 340:
Machine Learning and Data Mining

Non-Parametric Models

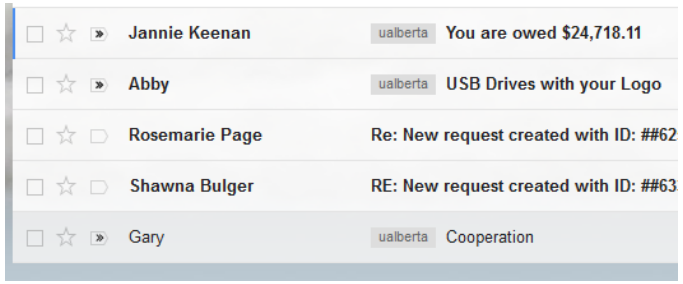
Fall 2016

Admin

- Course add/drop deadline tomorrow.
- Assignment 1 is due Friday.
  - Setup your CS undergrad account ASAP to use Handin:
    - https://www.cs.ubc.ca/getacct
  - Instructions for handin posted to Piazza.
  - Start the assignment ASAP, if you haven't already.
    - The material will be getting much harder and the workload much higher.

Application: E-mail Spam Filtering

- Want a build a system that filters spam e-mails:



- We formulated as supervised learning:
  - (y_i = 1) if e-mail ‘i’ is spam, (y_i = 0) if e-mail is not spam.
  - (xij = 1) if word/phrase ‘j’ is in e-mail ‘i’, (xij = 0) if it is not.

$	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
…	…	…	…	…	…	…	…

Generative Models

- We considered spam filtering methods based on generative models:

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

- What do these terms mean?

ALL E-MAILS

(including duplicates)

Generative Models

- We considered spam filtering methods based on generative models:

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

- $p(x_i)$ is probability that a random e-mail has features x_i.

ALL E-MAILS

(including duplicates)

Generative Models

- We considered spam filtering methods based on generative models:

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

- $p(x_i)$ is probability that a random e-mail has features x_i.

ALL E-MAILS

(including duplicates)

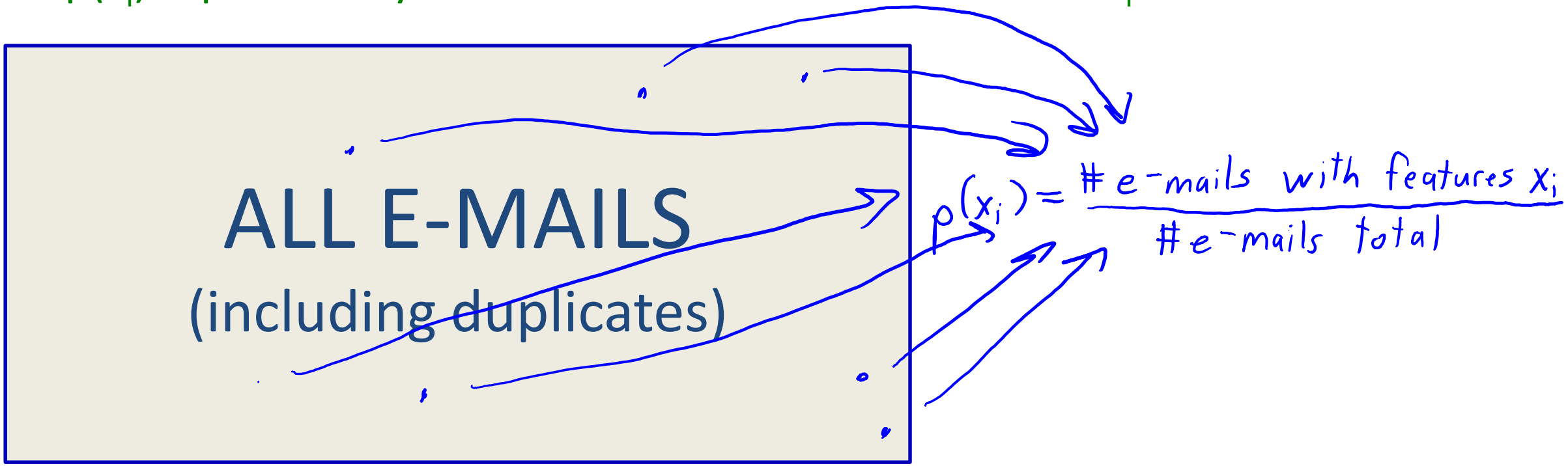
$$p(x_i) = \frac{\#\text{e-mails with features } x_i}{\#\text{e-mails total}}$$

Generative Models

- We considered spam filtering methods based on generative models:

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

- $p(x_i)$ is probability that a random e-mail has features x_i.

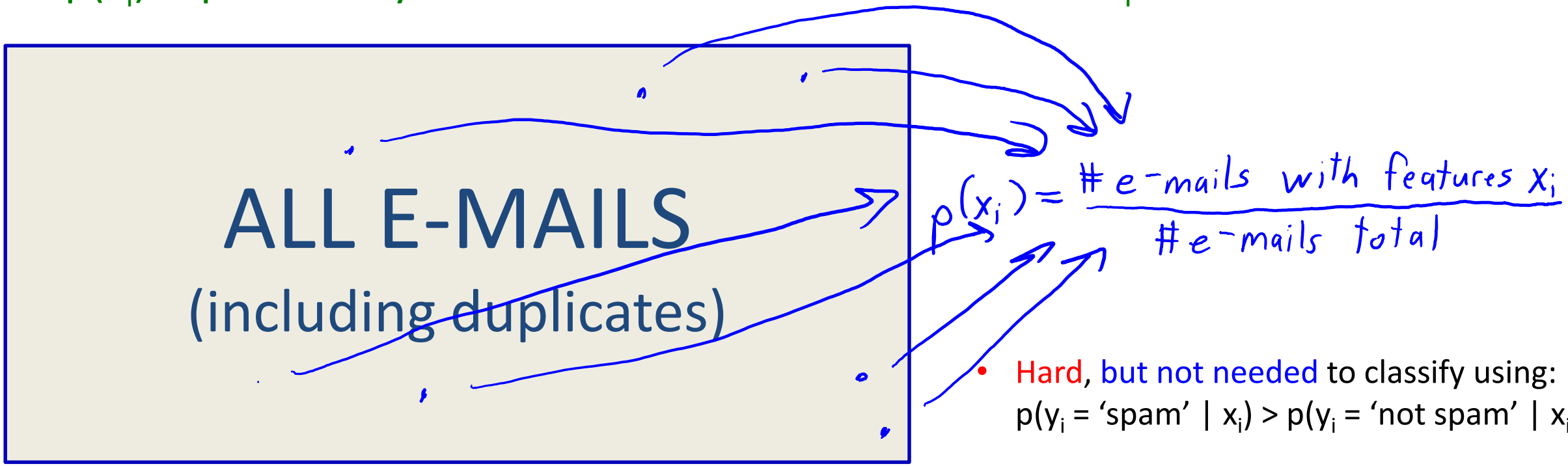


Generative Models

- We considered spam filtering methods based on generative models:

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

- $p(x_i)$ is probability that a random e-mail has features x_i.



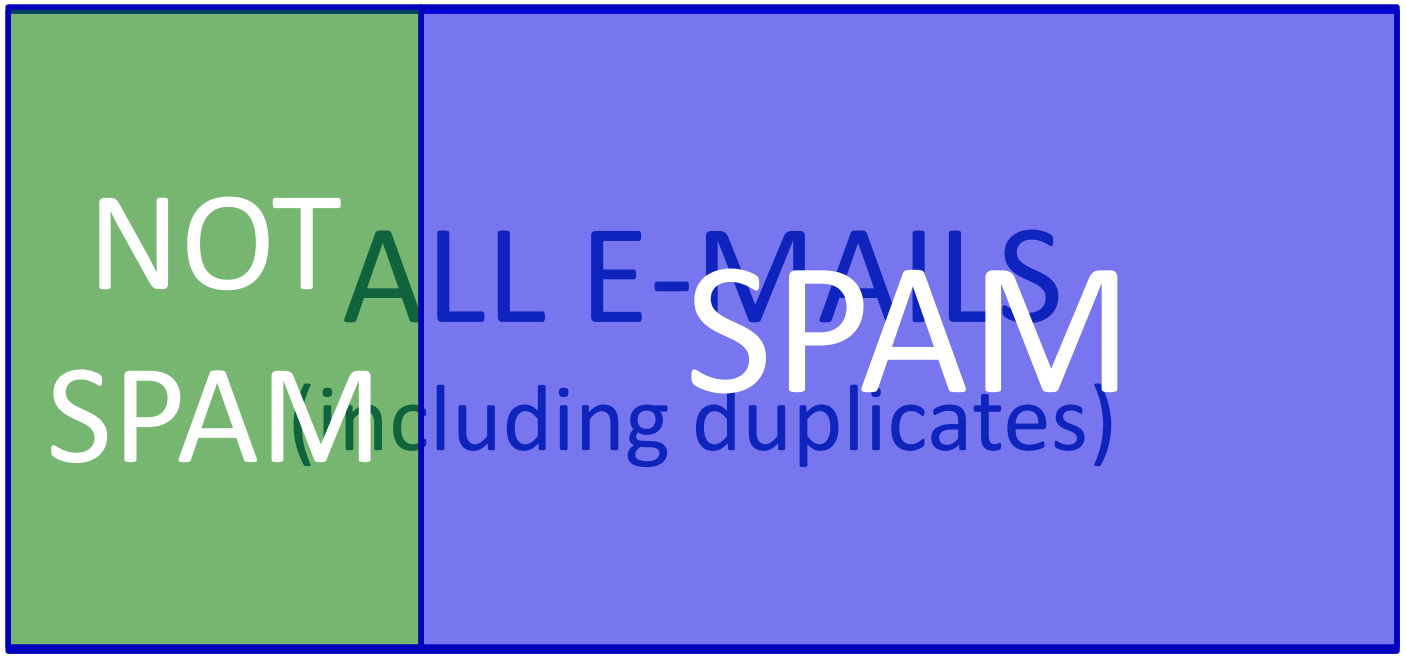
- Hard, but not needed to classify using:
  $p(y_i = \text{'spam'} \mid x_i) > p(y_i = \text{'not spam'} \mid x_i)$

Generative Models

- We considered spam filtering methods based on generative models:

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

- $p(y_i$ = 'spam') is probability that a random e-mail is spam.



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

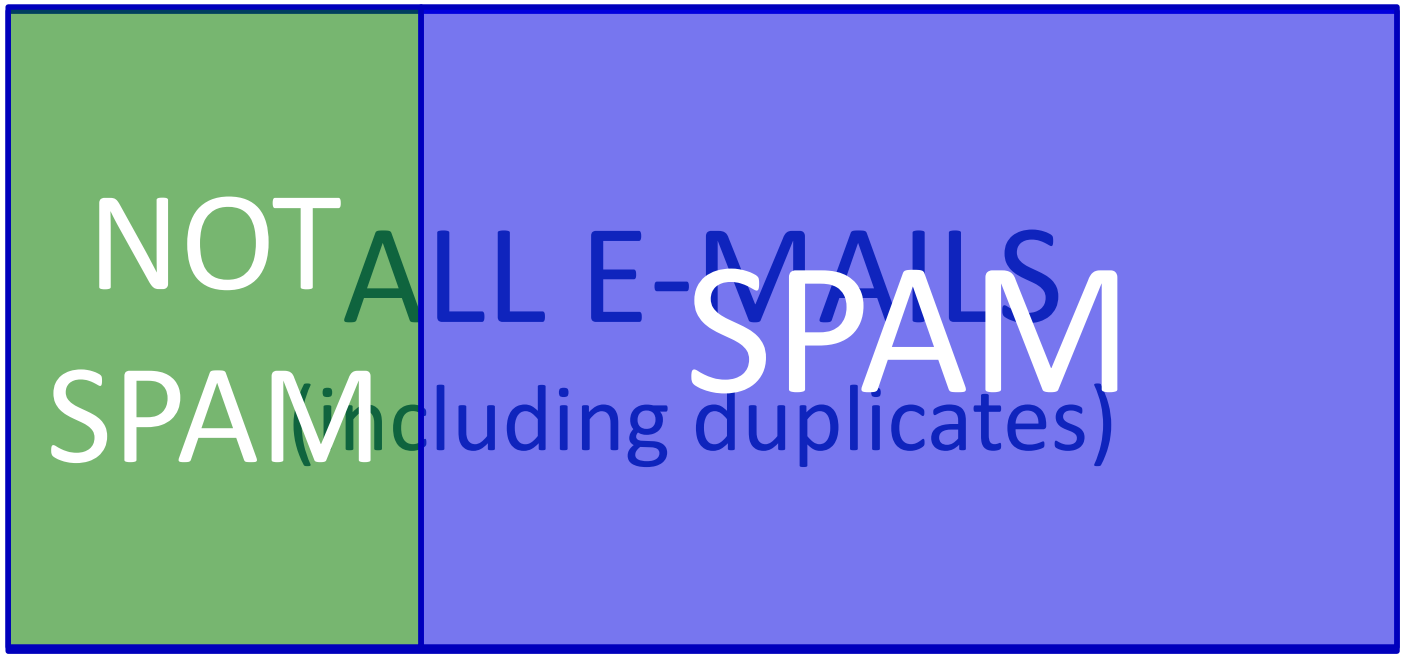
- Hard to compute exactly.
- But is easy to approximate from data:
  - Count (#spam in data)/(#messages)

Generative Models

- We considered spam filtering methods based on generative models:

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

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



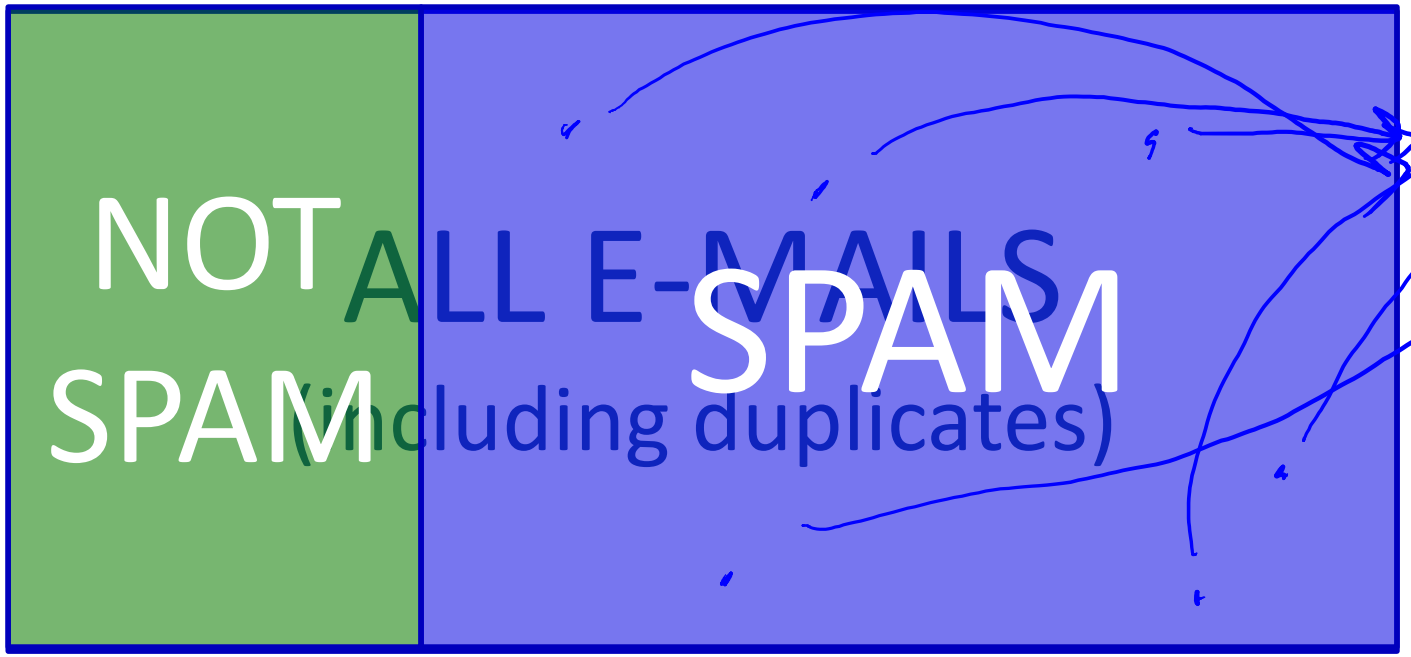
$$p(x_i \mid y_i = \text{"spam"}) = \frac{\#\text{ spam messages with features } x_i}{\#\text{ spam messages}}$$

Generative Models

- We considered spam filtering methods based on generative models:

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

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



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

- Very hard to estimate:
  - Too many possible x_i.

Naïve Bayes

- How the naïve Bayes model deals with the hard terms:

$$p(\text{spam} \mid \text{hello, vicodin, CPSC 340}) = \frac{p(\text{hello, vicodin, CPSC 340} \mid \text{spam})\, p(\text{spam})}{p(\text{hello, vicodin, CPSC 340})} \quad \text{(Bayes rule)}$$

"equal up to constant not depending on spam" $\propto$ $p(\text{hello, vicodin, CPSC 340} \mid \text{spam})\, p(\text{spam})$

HARD: $p(\text{hello, vicodin, CPSC 340} \mid \text{spam})$ — easy: $p(\text{spam})$

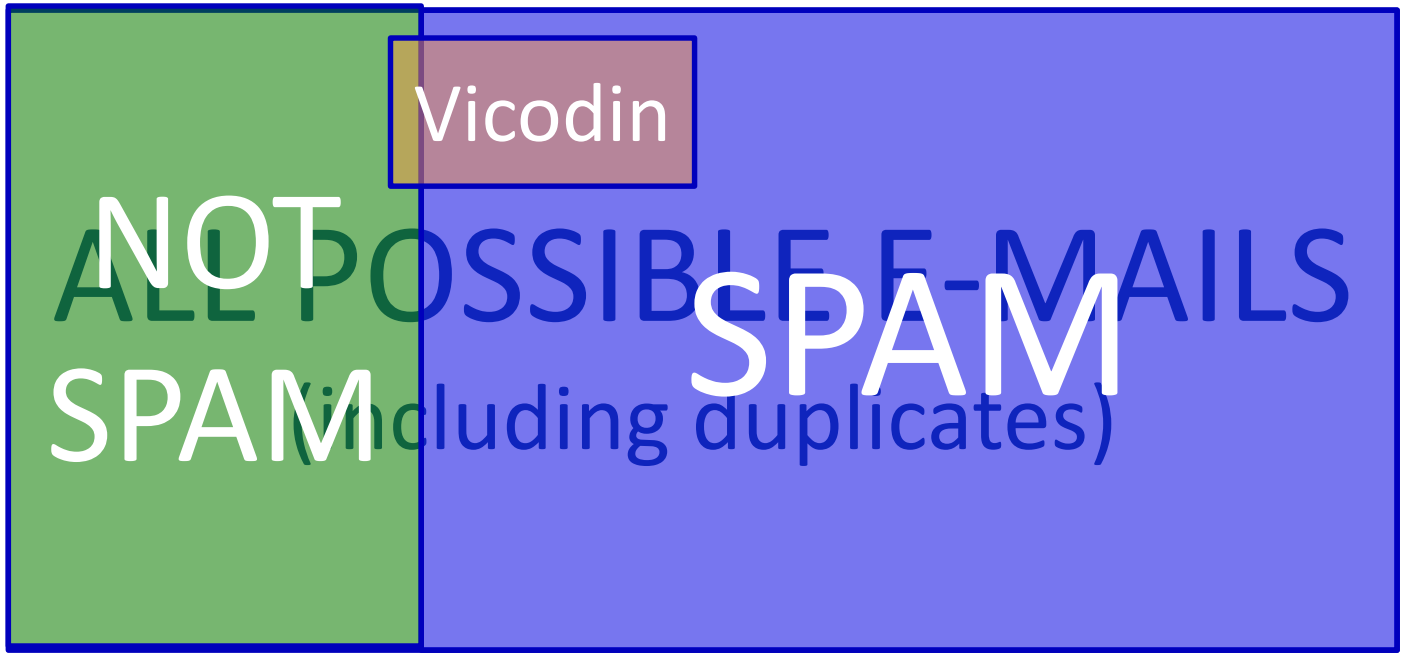
(naive Bayes assumption) $\approx$ $p(\text{hello} \mid \text{spam})\, p(\text{vicodin} \mid \text{spam})\, p(\text{CPSC 340} \mid \text{spam})\, p(\text{spam})$

easy easy easy easy

- Now only need easy quantities like p('vicodin' = 1| y_i = 'spam').

Naïve Bayes Models

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



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

- Easy to estimate:
  - #(spam w/ Vicodin)/#spam
- “Maximum likelihood estimate”

Naïve Bayes

- Naïve Bayes more formally:

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

$$\propto p(x_i \mid y_i)\, p(y_i)$$

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

  - Assumption: all x_i are conditionally independent give y_i.

Independence of Random Variables

- Events A and B are independent if $p(A,B) = p(A)p(B)$.
  - Equivalently: $p(A|B) = p(A)$.
  - "Knowing B happened tells you nothing about A".
  - We use the notation:

$$A \perp B$$

- Random variables are independent if $p(x,y) = p(x)p(y)$ for all x and y.
  - Flipping two coins:

$$p(C_1 = \text{'heads'}, C_2 = \text{'heads'}) = p(C_1 = \text{'heads'})p(C_2 = \text{'heads'}).$$
$$p(C_1 = \text{'tails'}, C_2 = \text{'heads'}) = p(C_1 = \text{'tails'})p(C_2 = \text{'heads'}).$$
$$\ldots$$

Conditional Independence

- A and B are conditionally independent given C if

$$p(A, B \mid C) = p(A \mid C)p(B \mid C).$$

  - Equivalently: $p(A \mid B, C) = p(A \mid C)$.
  - "Knowing C happened, also knowing B happened says nothing about A".
  - Example: $p(\text{Pizza} \mid D_1, \text{Survive}) = p(\text{Pizza} \mid \text{Survive})$.
  - Knowing you survived, dice 1 gives no information about chance of pizza.
  - We use the notation: $A \perp B \mid C$

- Semantics of $p(A, B \mid C, D)$:
  - "probability of A and B happening, if we know that C and D happened".

Naïve Bayes

- In naïve Bayes: assume features are independent given label.
  - "Once you know it's spam, there is no dependency between features."
  - Not true, but sometimes a good approximation.

Training:

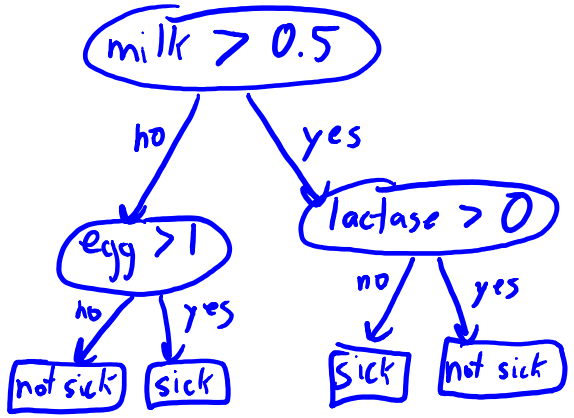
1. Estimate $p(y_i)$ for all y_i
2. Estimate $p(x_{ij} | y_i)$ for all y_i and x_{ij}

Predict:

Maximize $\prod_{j=1}^{d} [p(x_{ij} | y)] p(y)$ over all y.

Decision Trees vs. Naïve Bayes

- Decision trees:



1. Sequence of rules based on 1 feature.
2. Training: 1 pass over data per depth.
3. Hard to find optimal tree.
4. Testing: just look at features in rules.
5. New data: might need to change tree.
6. Accuracy: good if simple rules work.

- Naïve Bayes:

$$p(\text{sick} \mid \text{milk}, \text{egg}, \text{lactase})$$
$$\approx p(\text{milk} \mid \text{sick})\, p(\text{egg} \mid \text{sick})\, p(\text{lactase} \mid \text{sick})\, p(\text{sick})$$

1. Simultaneously combine all features.
2. Training: 1 pass over data to count.
3. Easy to find optimal probabilities.
4. Testing: look at all features.
5. New data: just update counts.
6. Accuracy: good if features almost independent given label.

Naïve Bayes Issues

1. 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.
     - Could use a sparse matrix representation.
2. Problem with maximum likelihood estimate (MLE):
   - MLE of p('lactase' = 1| 'spam') is (#spam messages with 'lactase')/#spam.
   - If you have no spam messages with lactase:
     - p('lactase' | 'spam') = 0, and message automatically gets through filter.
   - Fix: imagine we saw/not-saw each word in spam/not-spam messages:
     - Estimate p(<word> | 'spam') by (1 + count(spam with <word>)/(2 + #spam).
     - "Laplace smoothing" (could replace '1' by 'c' and '2' with '2c').
3. Are we equally concerned about spam vs. not spam?

Decision Theory

- True positives, false positives, false negatives, false negatives:

Predict / True	True ‘spam’	True ‘not spam’
Predict ‘spam’	True Positive	False Positive
Predict ‘not spam’	False Negative	True Negative

- The costs mistakes might be different:
  - Letting a spam message through (false negative) is not a big deal.
  - Filtering a not spam (false positive) message will make users mad.

Decision Theory

- We can give a cost to each scenario, such as:

Predict / True	True 'spam'	True 'not spam'
Predict 'spam'	0	100
Predict 'not spam'	10	0

- Instead of assigning to most likely classify, minimize expected cost:

$$E[C(\hat{y}_i = \text{spam})] = p(y_i = \text{spam} \mid x_i) C(\hat{y}_i = \text{spam}, y_i = \text{spam}) + p(y_i = \text{not spam} \mid x_i) C(\hat{y}_i = \text{spam}, y_i = \text{not spam})$$

"cost of predicting spam when e-mail is not spam"

- Even if p(spam $|x_i$) > p(not spam $| x_i$),
  - Might still classify as "not spam",
    if E[C(yhat$_i$ = spam)] > E[C(yhat$_i$ = not spam)].

Parametric vs. Non-Parametric

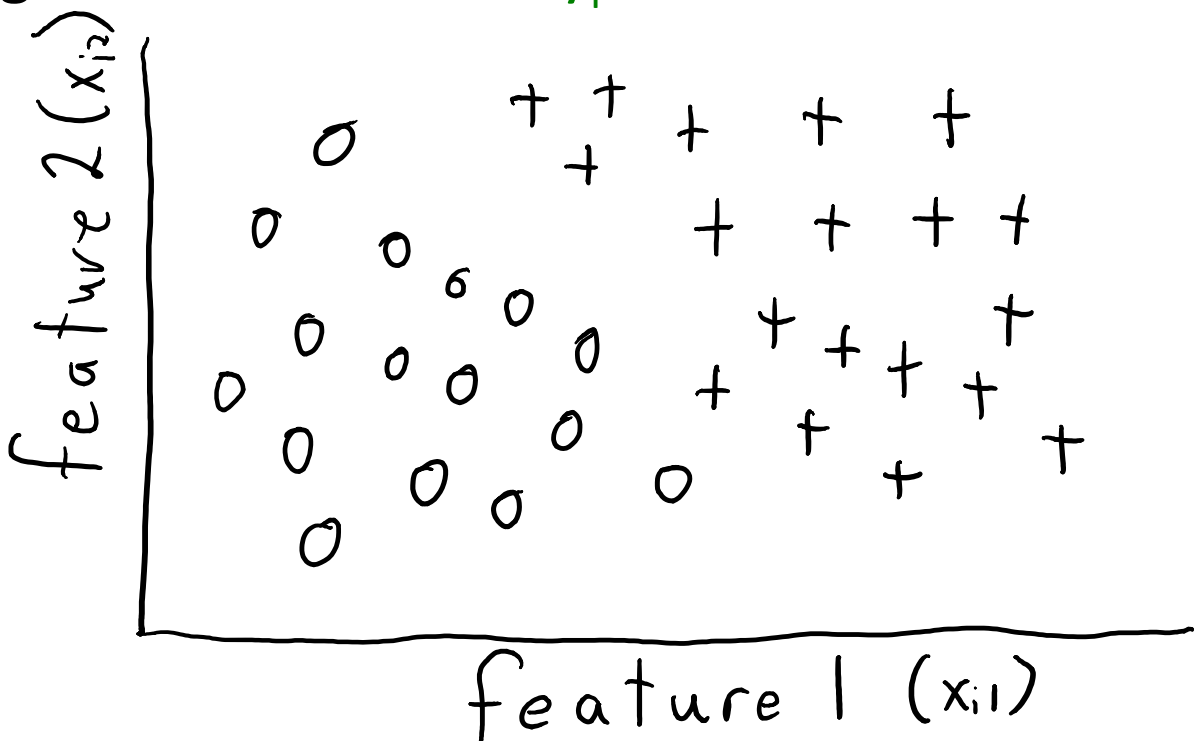
- Decision trees and naïve Bayes are often not very accurate.
  - Greedy rules or conditional independence might be bad assumptions.
  - They are also parametric models.

Parametric vs. Non-Parametric

- Parametric models:
  - Have a fixed number of parameters: size of "model" is O(1) in terms 'n'.
    - E.g., decision tree just stores rules.
    - E.g., naïve Bayes just stores counts.
  - You can estimate the fixed parameters more accurately with more data.
  - But eventually more data doesn't help: model is too simple.
- Non-parametric models:
  - Number of parameters grows with 'n': size of "model" depends on 'n'.
  - Model gets more complicated as you get more data.
  - E.g., decision tree whose depth *grows with the number of examples*.

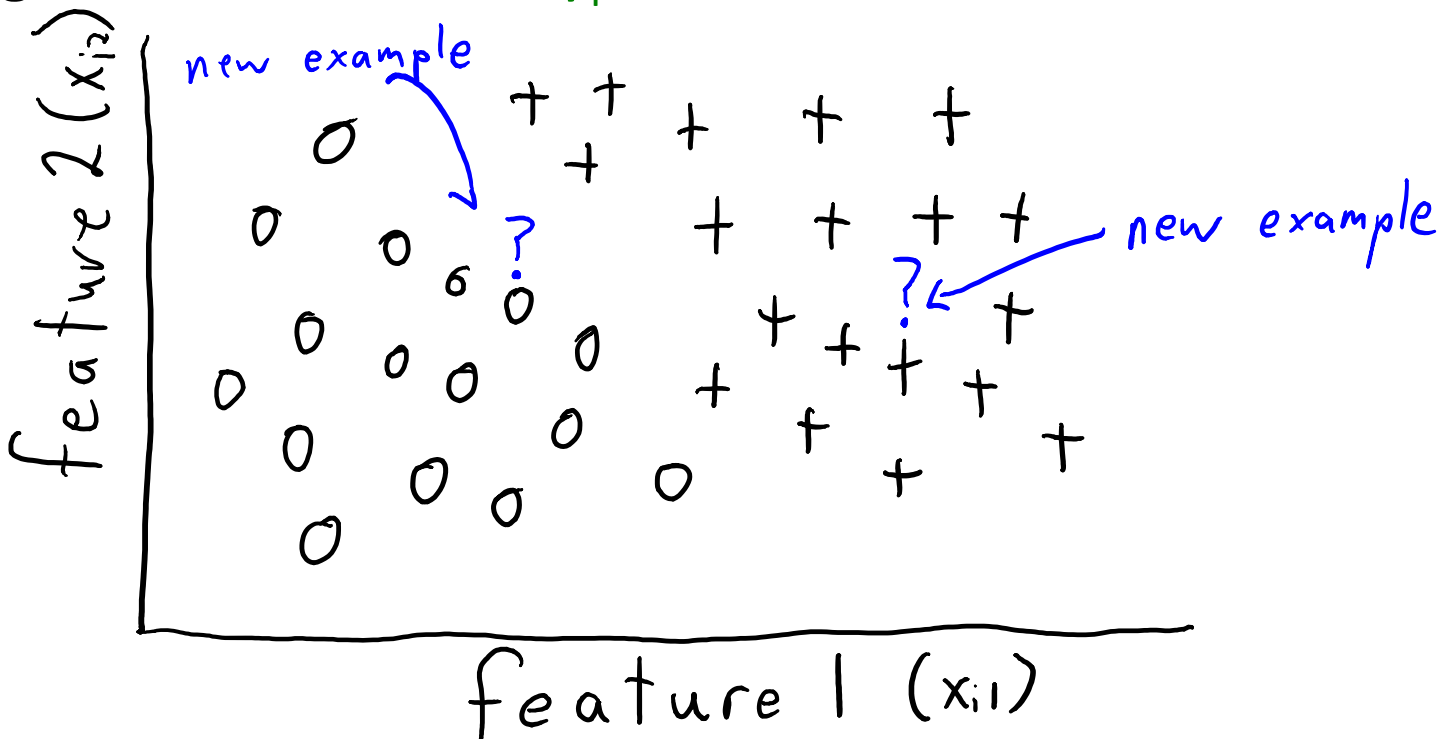
K-Nearest Neighbours (KNN)

- Classical non-parametric classifier is k-nearest neighbours (KNN).
- KNN algorithm for classifying an object ‘x’:
  1. Find ‘k’ training examples x_i that are most “similar” to x.
  2. Classify using the mode of their y_i.



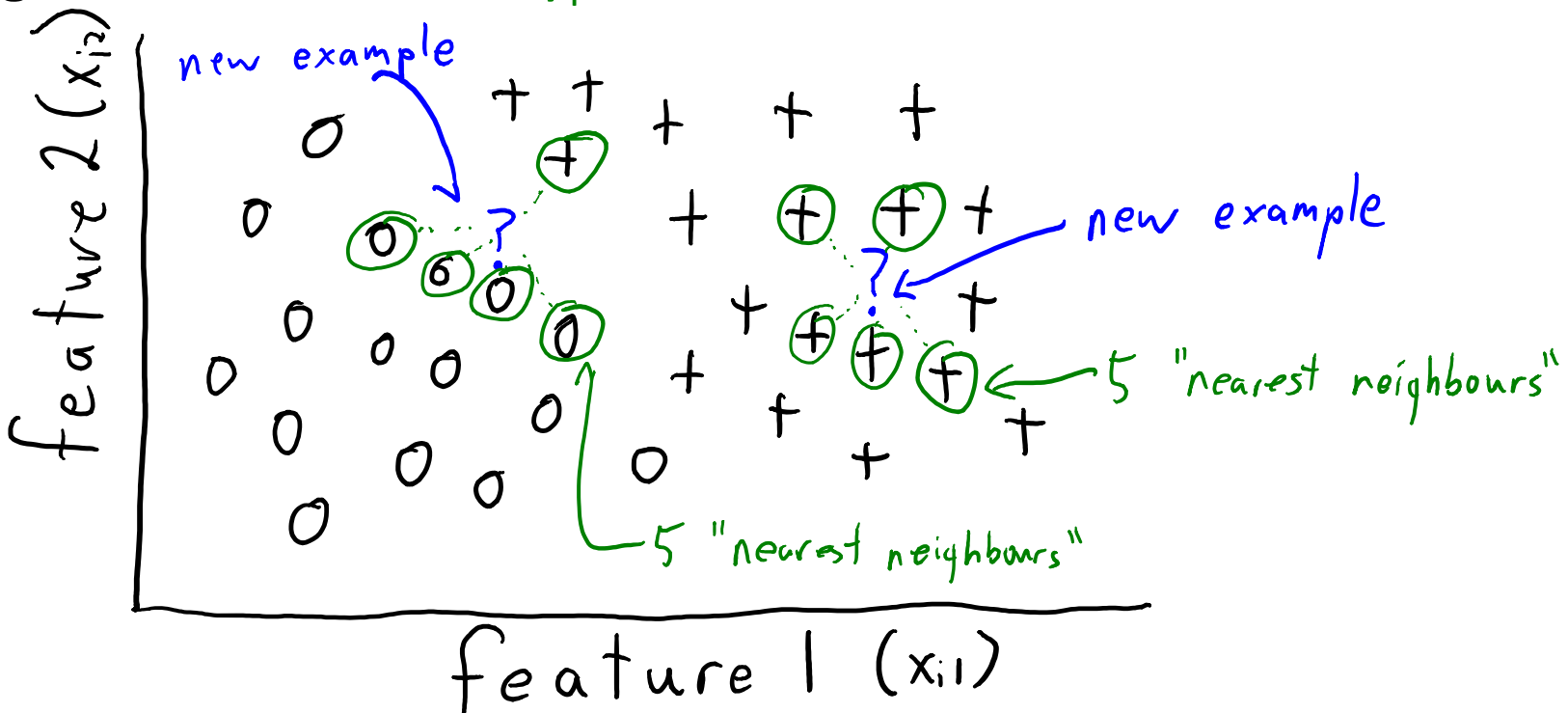
K-Nearest Neighbours (KNN)

- Classical non-parametric classifier is k-nearest neighbours (KNN).
- KNN algorithm for classifying an object ‘x’:
  1. Find ‘k’ training examples x_i that are most “similar” to x.
  2. Classify using the mode of their y_i.



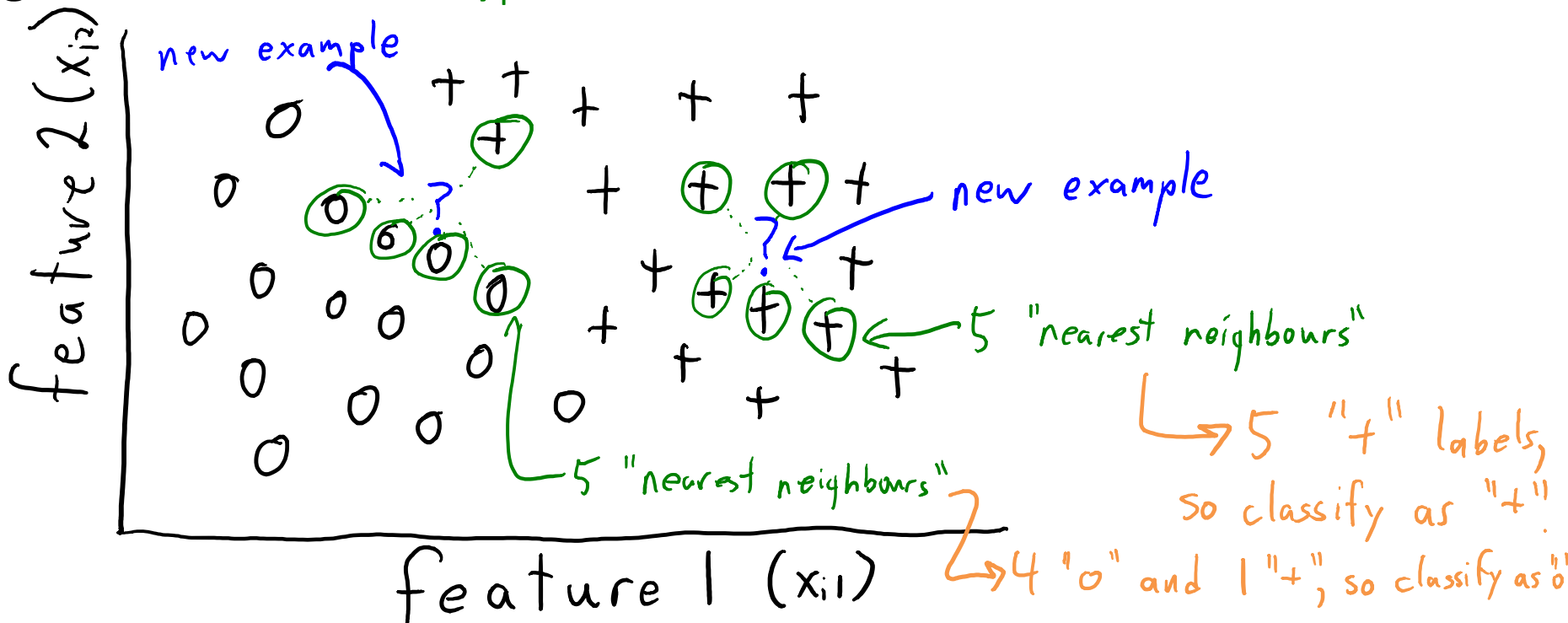
K-Nearest Neighbours (KNN)

- Classical non-parametric classifier is k-nearest neighbours (KNN).
- KNN algorithm for classifying an object ‘x’:
  1. Find ‘k’ training examples x_i that are most “similar” to x.
  2. Classify using the mode of their y_i.



K-Nearest Neighbours (KNN)

- Classical non-parametric classifier is k-nearest neighbours (KNN).
- KNN algorithm for classifying an object 'x':
  1. Find 'k' training examples x_i that are most "similar" to x.
  2. Classify using the mode of their y_i.



K-Nearest Neighbours (KNN)

- Assumption:
  - Objects with similar features likely have similar labels.

- There is no training phase ("lazy" learning).
  - You just store the training data.
  - Non-parametric because the size of the model is O(nd).

- But predictions are expensive: O(nd) to classify 1 test object.
  - Tons of work on reducing this cost.

How to Define 'Nearest'?

- There are many ways to define similarity between x_i and x_j.
- Most common is Euclidean distance:

$$D(x_1, x_2) = \sqrt{\sum_{j=1}^{d} (x_{1j} - x_{2j})^2}$$

- Other possibilities:
  - L_1 distance:

$$D(x_1, x_2) = \sum_{j=1}^{d} |x_{1j} - x_{2j}|$$

  - Jaccard similarity (binary):

$$D(x_1, x_2) = \frac{x_1 \cap x_2}{x_1 \cup x_2}$$

$x_1 \cap x_2$ → # times both are '1'

$x_1 \cup x_2$ → # times either is '1'

  - Cosine similarity.
  - Distance after dimensionality reduction (later in course).
  - Metric learning (*learn* the best distance function).

Consistency of KNN

- With a small dataset, KNN model will be very simple.
- With more data, model gets more complicated:
  - Starts to detect subtle differences between examples.
- With a fixed ‘k’, it has appealing consistency properties:
  - With binary labels and under mild assumptions:
    - As ‘n’ goes to infinity, KNN test error is less than twice irreducible error.
- Stone’s Theorem:
  - If k/n goes to zero and ‘k’ goes to infinity:
    - KNN is ‘universally consistent’: test error converges to the irreducible error.
    - First algorithm shown to have this property.
- Does Stone’s Theorem violate the no free lunch theorem?
  - No, requires assumptions on data and says nothing about finite training sets.

Summary

1. Naïve Bayes:
   - Conditional independence assumptions to make estimation practical.
2. Decision theory allows us to consider costs of predictions.
3. Non-parametric models grow with number of training examples.
4. K-Nearest Neighbours:
   - A simple non-parametric classifier.
   - Appealing consistency properties.

- Next Time:
  - Learning behind Microsoft Kinect.