CPSC 340: Machine Learning and Data Mining

Non-Parametric Models

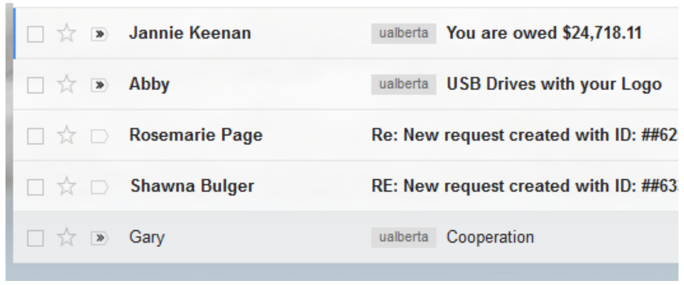
Fall 2016

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:



Jannie Keenan	ualberta	You are owed $24,718.11
Abby	ualberta	USB Drives with your Logo
Rosemarie Page		Re: New request created with ID: ##62
Shawna Bulger		RE: New request created with ID: ##63
Gary	ualberta	Cooperation

- 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	…
1	1	0	0	1	0	…
0	0	0	0	1	1	…
0	1	1	1	0	0	…
…	…	…	…	…	…	…

Spam?
1
1
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)

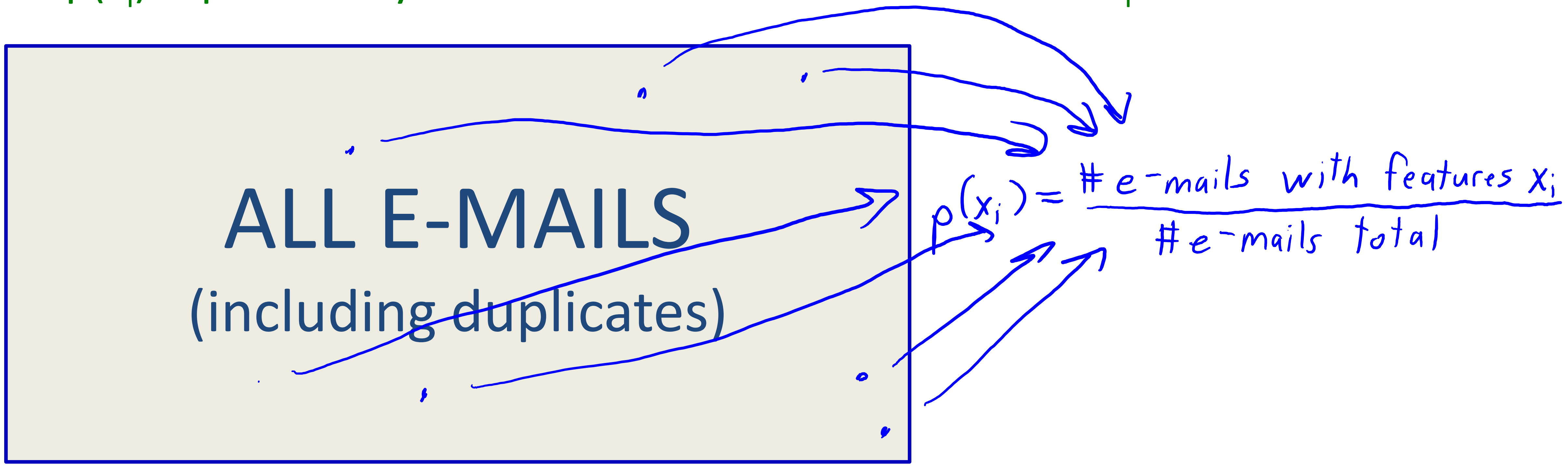
$$p(x_i) = \frac{\#\text{ e-mails with features } x_i}{\#\text{ e-mails total}}$$

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.



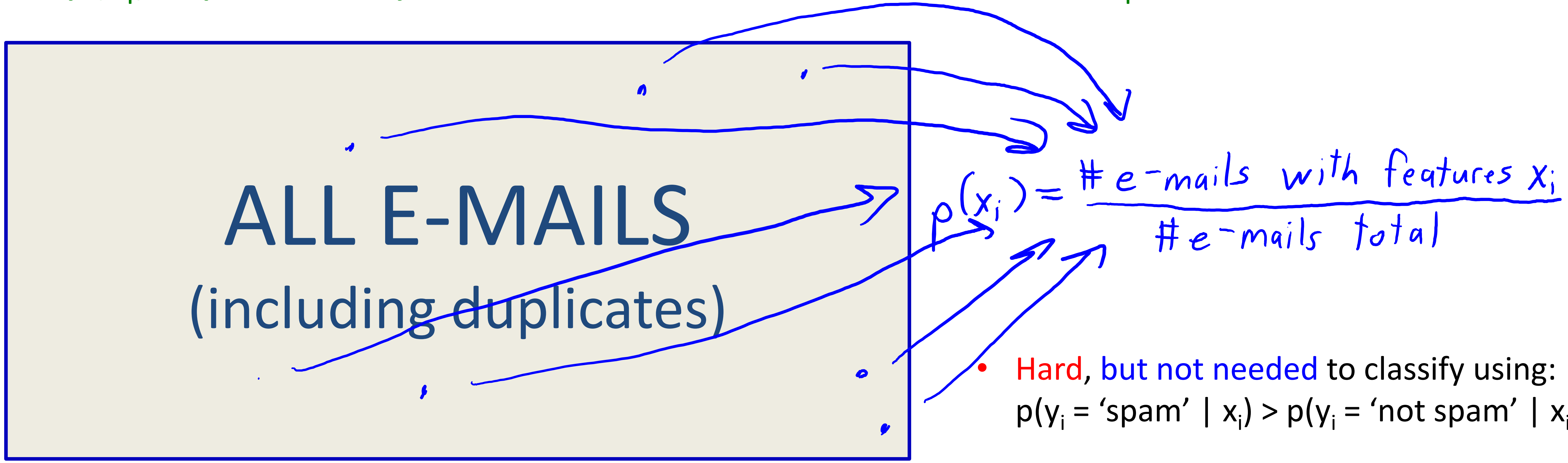
$$p(x_i) = \frac{\#\text{ e-mails with features } x_i}{\#\text{ e-mails total}}$$

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.



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

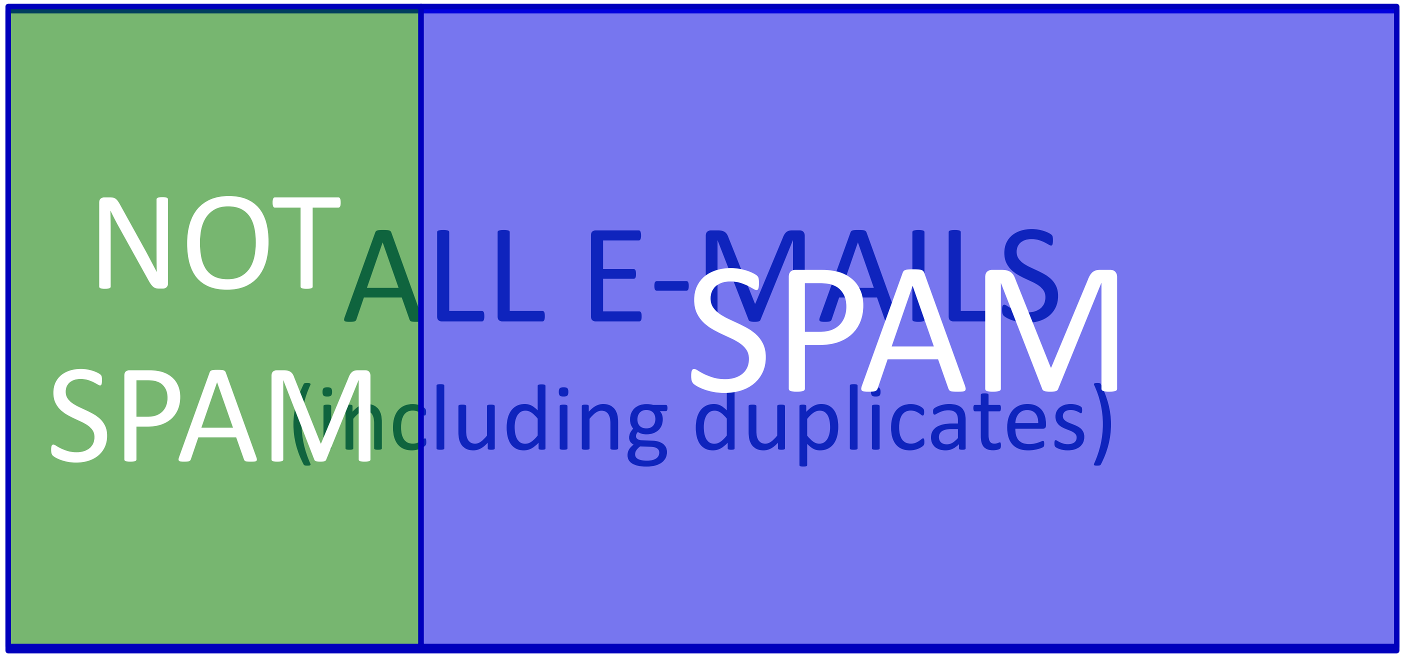
- Hard, but not needed to classify using:
  $p(y_i = \text{'spam'} \mid x_i) > p(y_i = \text{'not spam'} \mid x_i)$

Generative Models

- We considered spam filtering methods based on generative models:

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

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



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

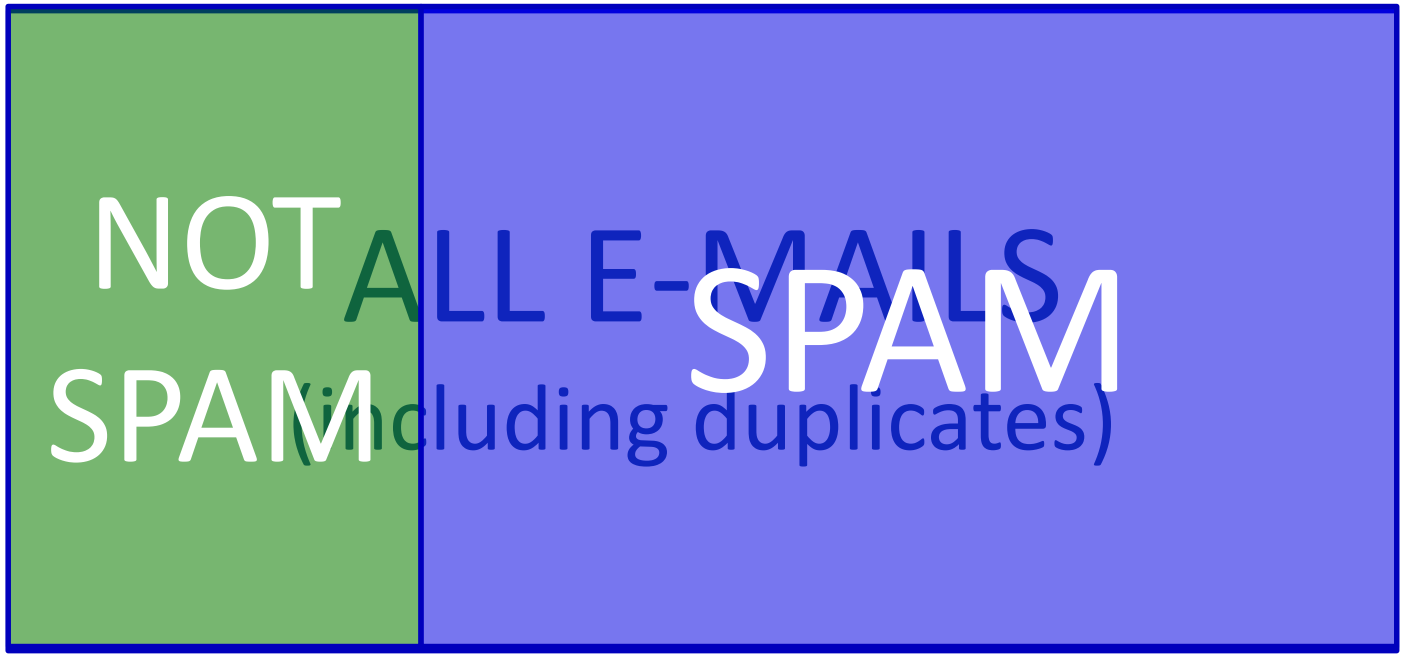
- Hard to compute exactly.
- But is easy to approximate from data:
  - Count (#spam in data)/(#messages)

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.



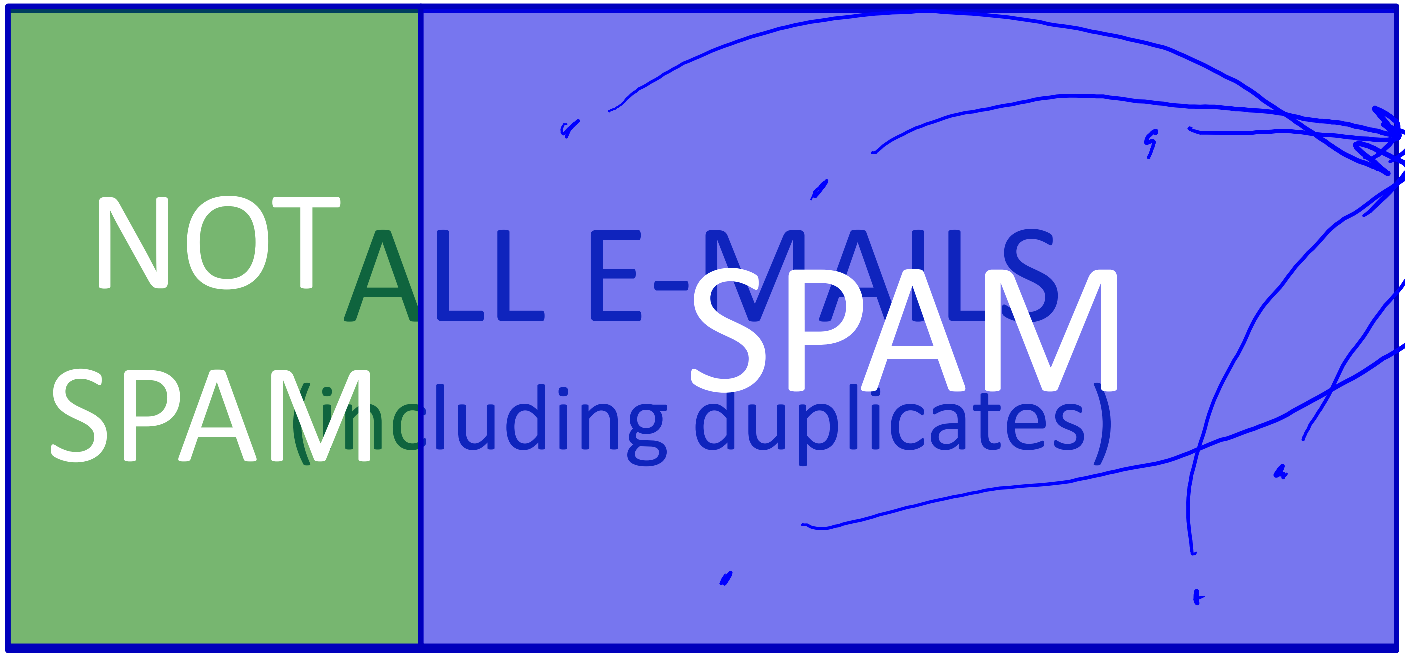
$$p(x_i \mid y_i = \text{"spam"}) = \frac{\text{\# \underline{spam} messages with \underline{features} } x_i}{\text{\# spam messages}}$$

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.



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

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

Naïve Bayes

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

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

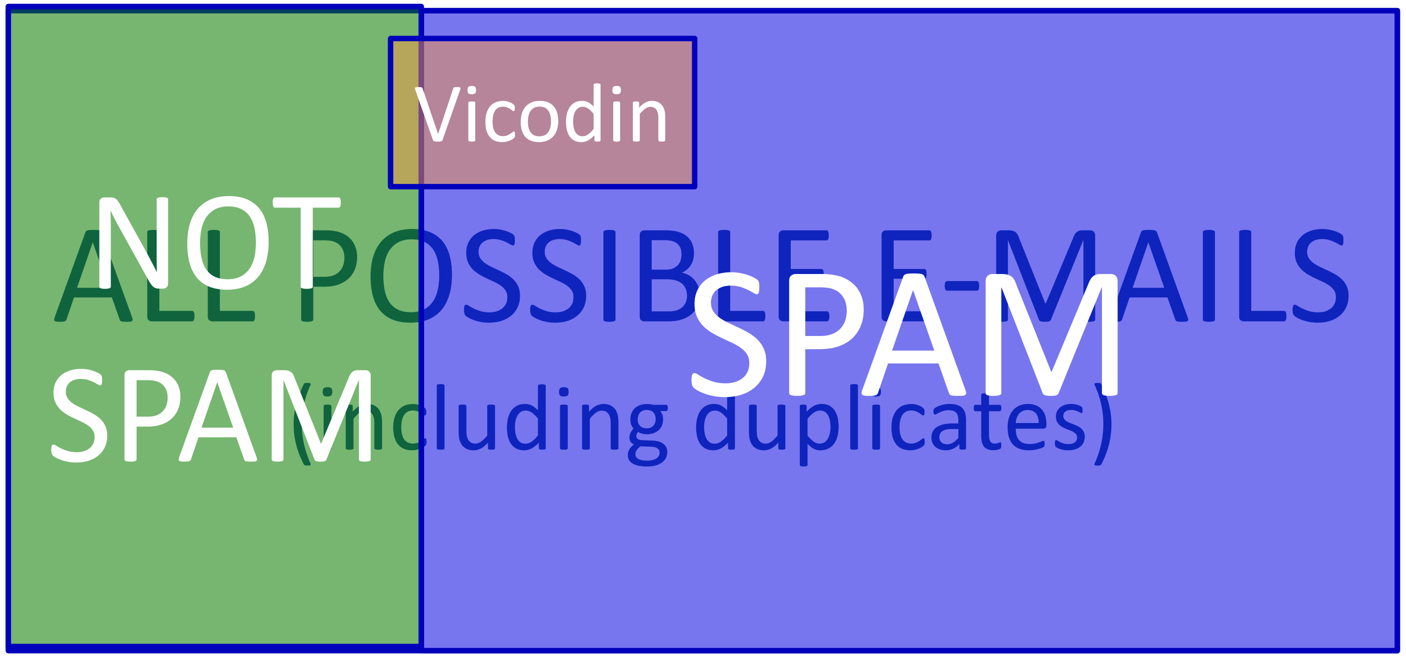
$$\text{"equal up to constant \underline{not} depending on spam"} \propto \underbrace{p(\text{hello, vicodin, CPSC 340} \mid \text{spam})}_{\text{HARD}} \underbrace{p(\text{spam})}_{\text{easy}}$$

$$\text{(naive Bayes \underline{assumption})} \quad \approx \underbrace{p(\text{hello} \mid \text{spam})}_{\text{easy}} \underbrace{p(\text{vicodin} \mid \text{spam})}_{\text{easy}} \underbrace{p(\text{CPSC 340} \mid \text{spam})}_{\text{easy}} \underbrace{p(\text{spam})}_{\text{easy}}$$

- 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:

$$p(y_i | x_i) = \frac{p(x_i | y_i) p(y_i)}{p(x_i)}$$

$$\propto p(x_i | y_i) p(y_i)$$

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

  - 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:

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

…

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

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.

Prediction:

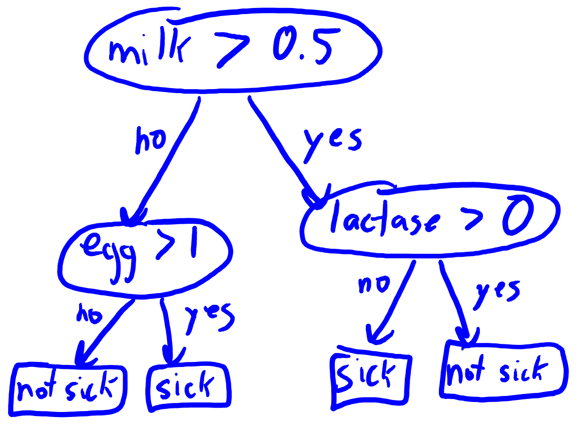
Given a new example x_i we want to find the 'c' maximizing $p(x_i | y_i)$.

Under the naive Bayes assumption we thus maximize

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

Decision Trees vs. Naïve Bayes

- Decision trees:



- Naïve Bayes:

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

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:

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:
     - "Laplace smoothing" for binary features: replace n_{cjk}/n_c with $(n_{cjk} + 1)/(n_c + 2)$.
     - A generalization is (ncjk + β)/(nc + 2β) for some constant "β".
     - If X_{ij} can take 'm' values, you would (ncjk + β)/(nc + mβ).

Naïve Bayes Issues

3. During the prediction, the probability can underflow:

$$p(y=c \mid x_i) \propto \underbrace{\prod_{j=1}^{d}\left[p(x_{ij} \mid y=c)\right] p(y=c)}_{\text{All these are } <1 \text{ so the product gets very small}}$$

All these are <1 so the product gets very small

- Standard fix is to (equivalently) maximize the logarithm of the probability:
  - Logarithm turns multiplication of small numbers into addition of small numbers.
  - Logarithm is monotonic, so it doesn't change location of the maximum.

4. 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:

$$E[C(\hat{y}_i = \text{spam})] = p(y_i = \text{spam} \mid x_i)\,C(\hat{y}_i = \text{spam}, y_i = \text{spam}) + p(y_i = \text{not spam} \mid x_i)\,\underbrace{C(\hat{y}_i = \text{spam}, y_i = \text{not spam})}_{\text{"cost of predicting spam when e-mail is not spam"}}$$

- 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)].

Decision Theory and Darts

- Post on decision theory in “darts”:
  - http://www.datagenetics.com/blog/january12012/index.html

- If you are very accurate, obviously aim for the high-scoring regions.
- If you are very inaccurate, obviously aim for the middle.
- Decision theory gives you the best strategy for other accuracies.

Parametric vs. Non-Parametric

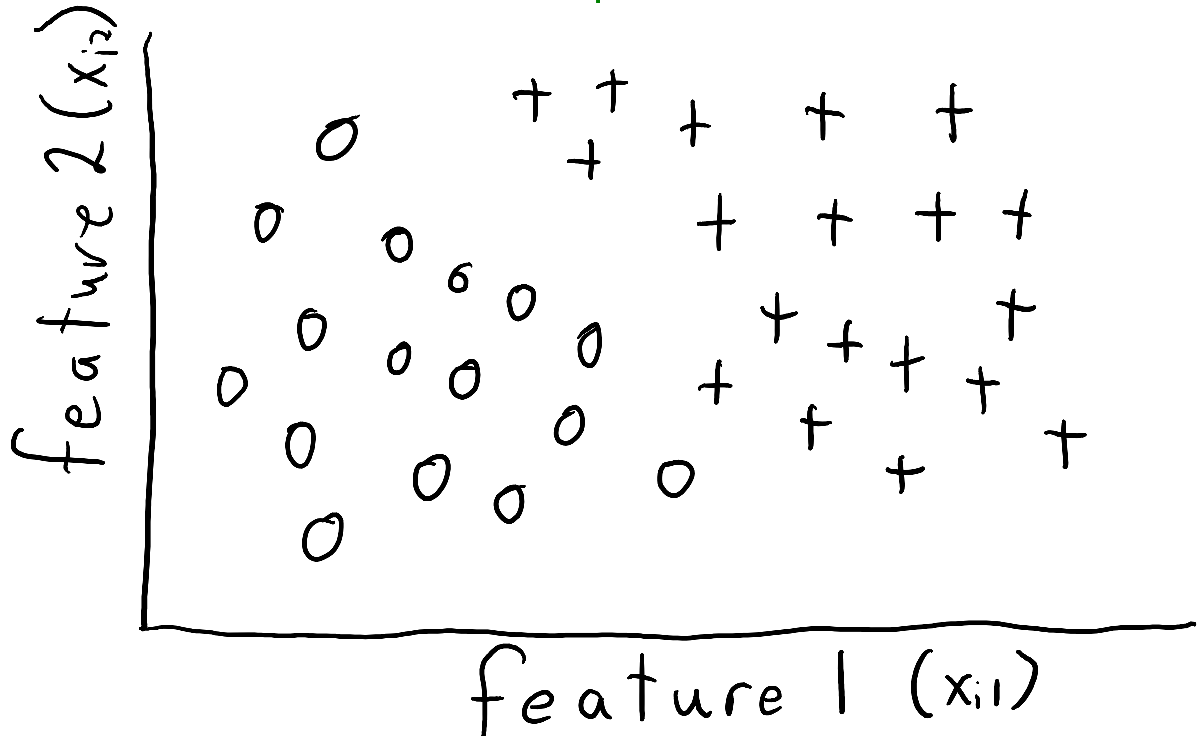
- Decision trees and naïve Bayes are often not very accurate.
  - Greedy rules or conditional independence might be bad assumptions.
  - They are also parametric models.

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*.

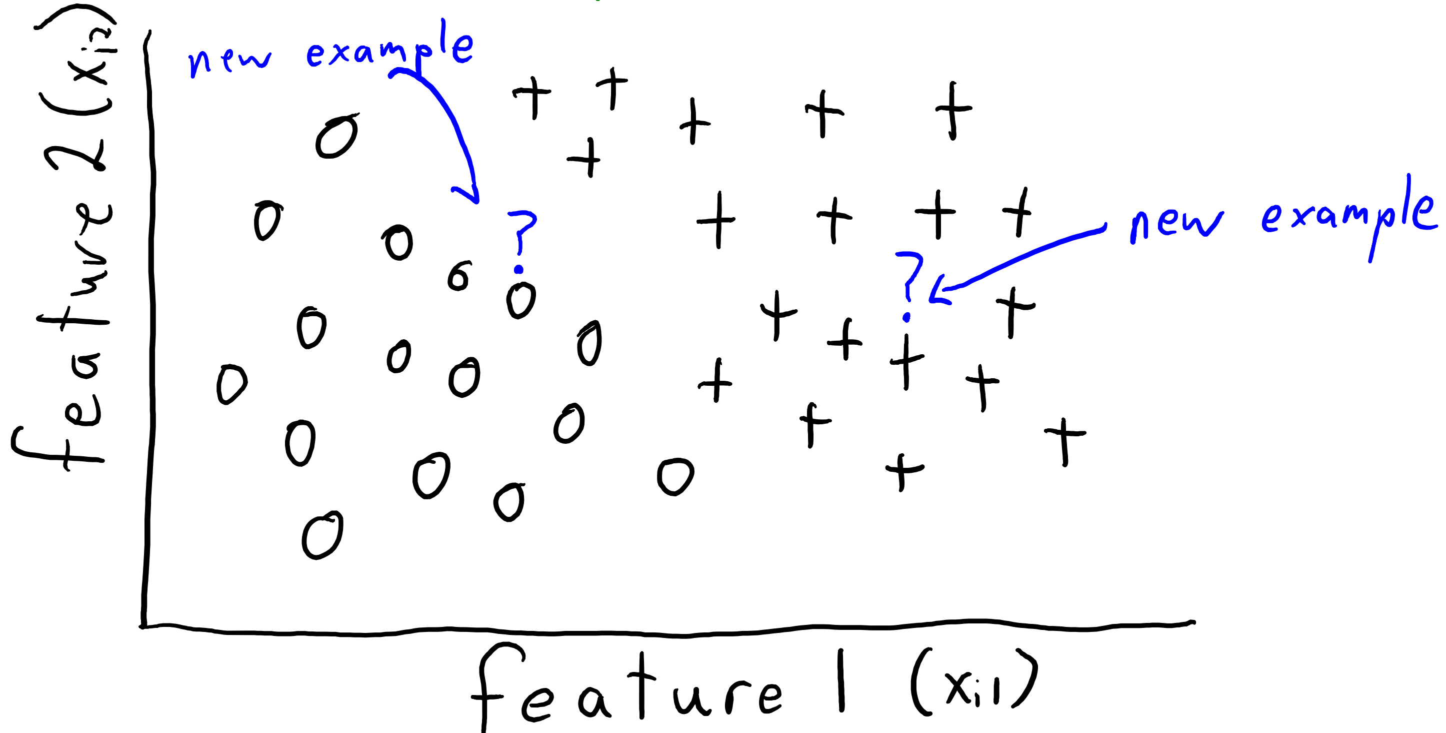
K-Nearest Neighbours (KNN)

- 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.



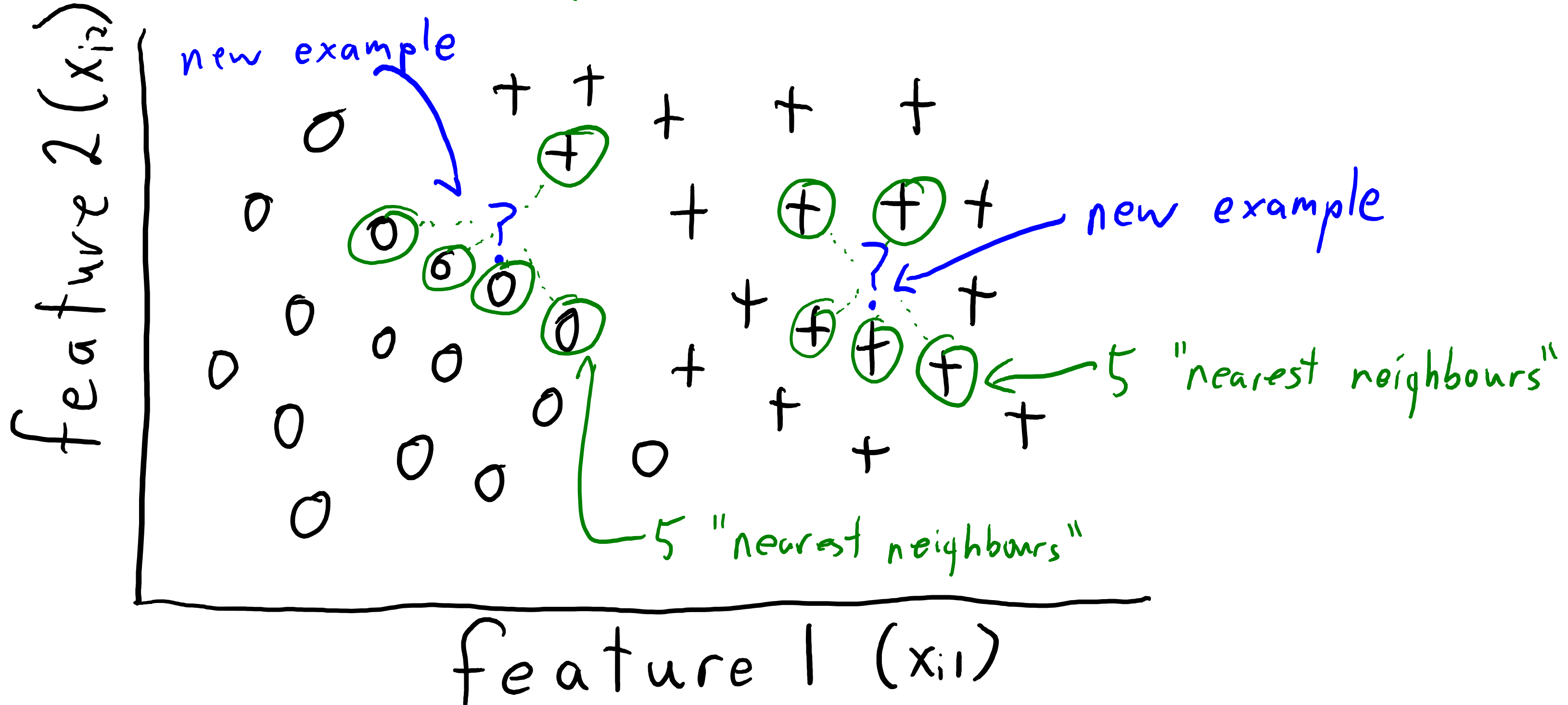
K-Nearest Neighbours (KNN)

- 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.



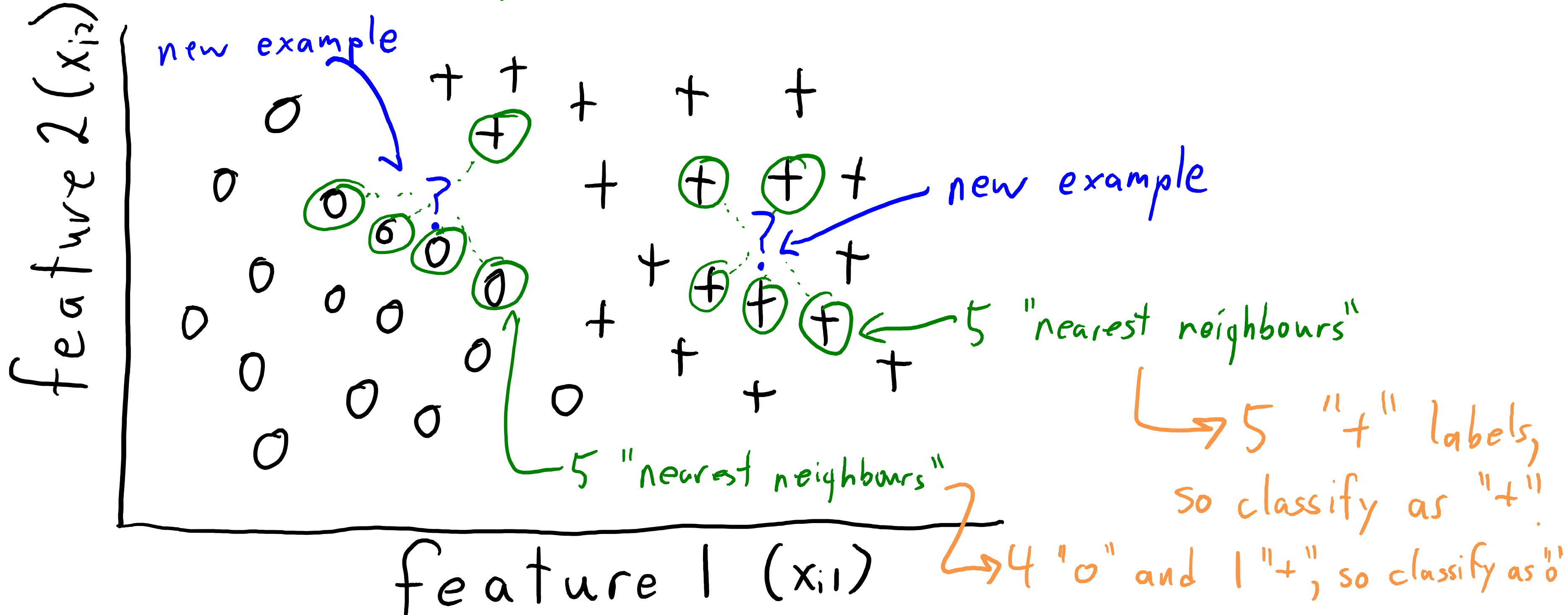
K-Nearest Neighbours (KNN)

- 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 (we'll discuss these later).

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): $$D(x_1, x_2) = \frac{x_1 \cap x_2 \xrightarrow{} \text{\# times both are '1'}}{x_1 \cup x_2 \xrightarrow{} \text{\# times either is '1'}}$$
  - 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.

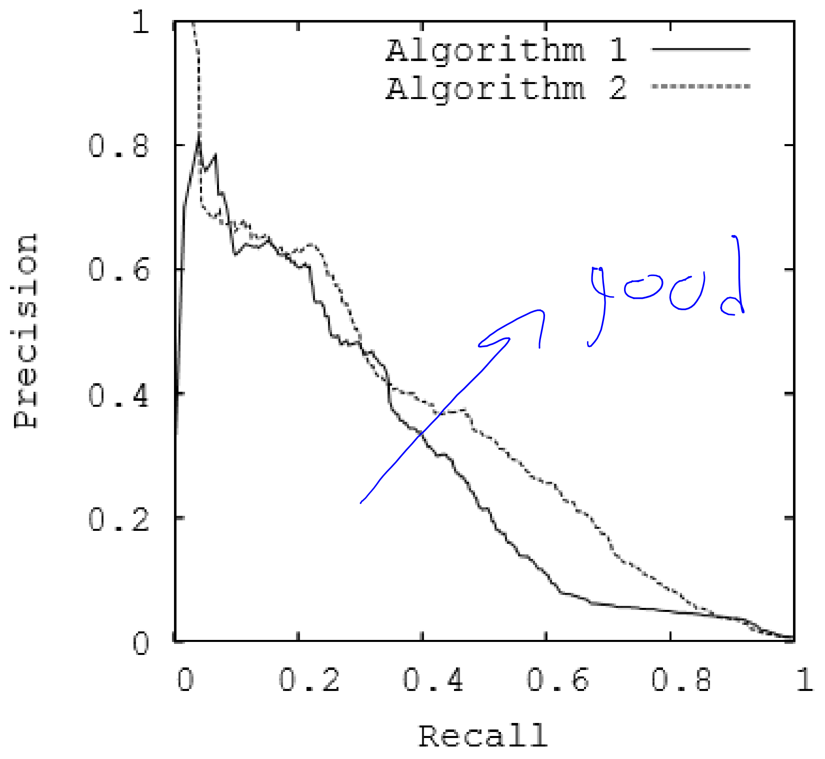
- Next Time:
    - Learning behind Microsoft Kinect.

Bonus Slide: Other Performance Measures

- Classification error might be wrong measure:
  - Use weighted classification error if have different costs.
  - Might want to use things like Jaccard measure: TP/(TP + FP + FN).
- Often, we report precision and recall (want both to be high):
  - Precision: “if I classify as spam, what is the probability it actually is spam?”
    - Precision = TP/(TP + FP).
    - High precision means the filtered messages are likely to really be spam.
  - Recall: “if a message is spam, what is probability it is classified as spam?”
    - Recall = TP/(TP + FN)
    - High recall means that most spam messages are filtered.

Bonus Slide: Precision-Recall Curve

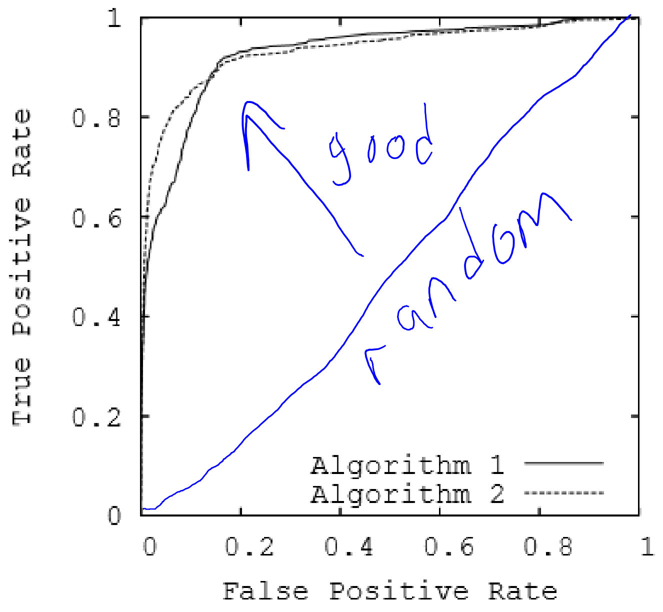
- Consider the rule $p(y_i = \text{'spam'} \mid x_i) > t$, for threshold 't'.
- Precision-recall (PR) curve plots precision vs. recall as 't' varies.



Bonus Slide: ROC Curve

- Receiver operating characteristic (ROC) curve:
  - Plot true positive rate (recall) vs. false positive rate (FP/FP+TN).

    (negative examples classified as positive)



  - Diagonal is random, perfect classifier would be in upper left.
  - Sometimes papers report area under curve (AUC).

http://pages.cs.wisc.edu/~jdavis/davisgoadrichcamera2.pdf

Bonus Slide: Avoiding Underflow

- During the prediction, the probability can underflow:

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

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 $p(y=c \mid x_i)$ also maximizes

$$\log p(y_i=c \mid x_i) = \sum_{j=1}^{d} \left[p(x_{ij} \mid y=c) \right] + p(y=c) + \underbrace{\text{constant}}_{\text{which is the same for all 'c'}}$$

Bonus Slide: 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:

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

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

$$= \sum_{c=1}^{K} \underbrace{\left[\prod_{j=1}^{d} p(x_{ij} \mid y=c)\right] p(y=c)}_{\text{These are the quantities we compute during training.}} \quad \text{(naive Bayes assumption)}$$

Bonus Slide: 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

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

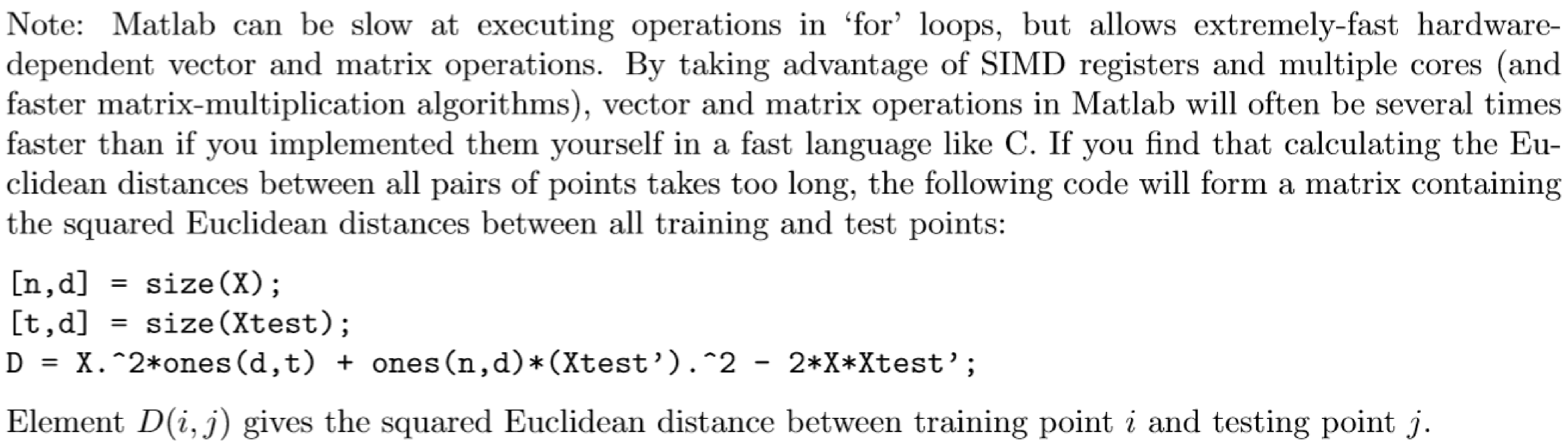
$$\approx p(y)\, p(x_1 \mid y)\, p(x_2 \mid y)\, p(x_3 \mid y) \cdots p(x_d \mid y) \quad \text{(naive Bayes assumption)}$$

- 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.

Bonus Slide: Computing all distances in Matlab

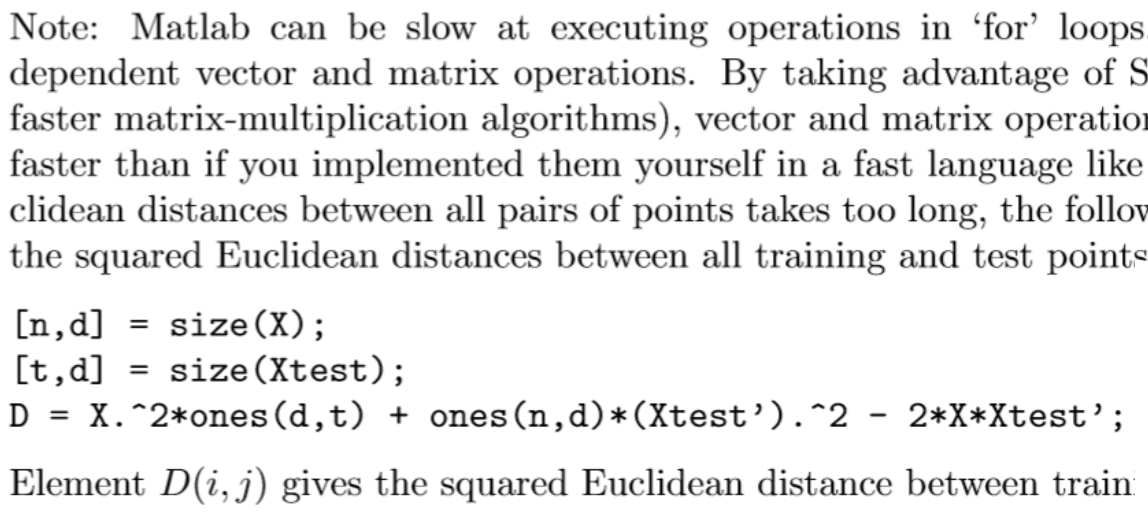


Note: Matlab can be slow at executing operations in ‘for’ loops, but allows extremely-fast hardware-dependent vector and matrix operations. By taking advantage of SIMD registers and multiple cores (and faster matrix-multiplication algorithms), vector and matrix operations in Matlab will often be several times faster than if you implemented them yourself in a fast language like C. If you find that calculating the Euclidean distances between all pairs of points takes too long, the following code will form a matrix containing the squared Euclidean distances between all training and test points:

```
[n,d] = size(X);
[t,d] = size(Xtest);
D = X.^2*ones(d,t) + ones(n,d)*(Xtest’).^2 - 2*X*Xtest’;
```

Element $D(i, j)$ gives the squared Euclidean distance between training point i and testing point j.

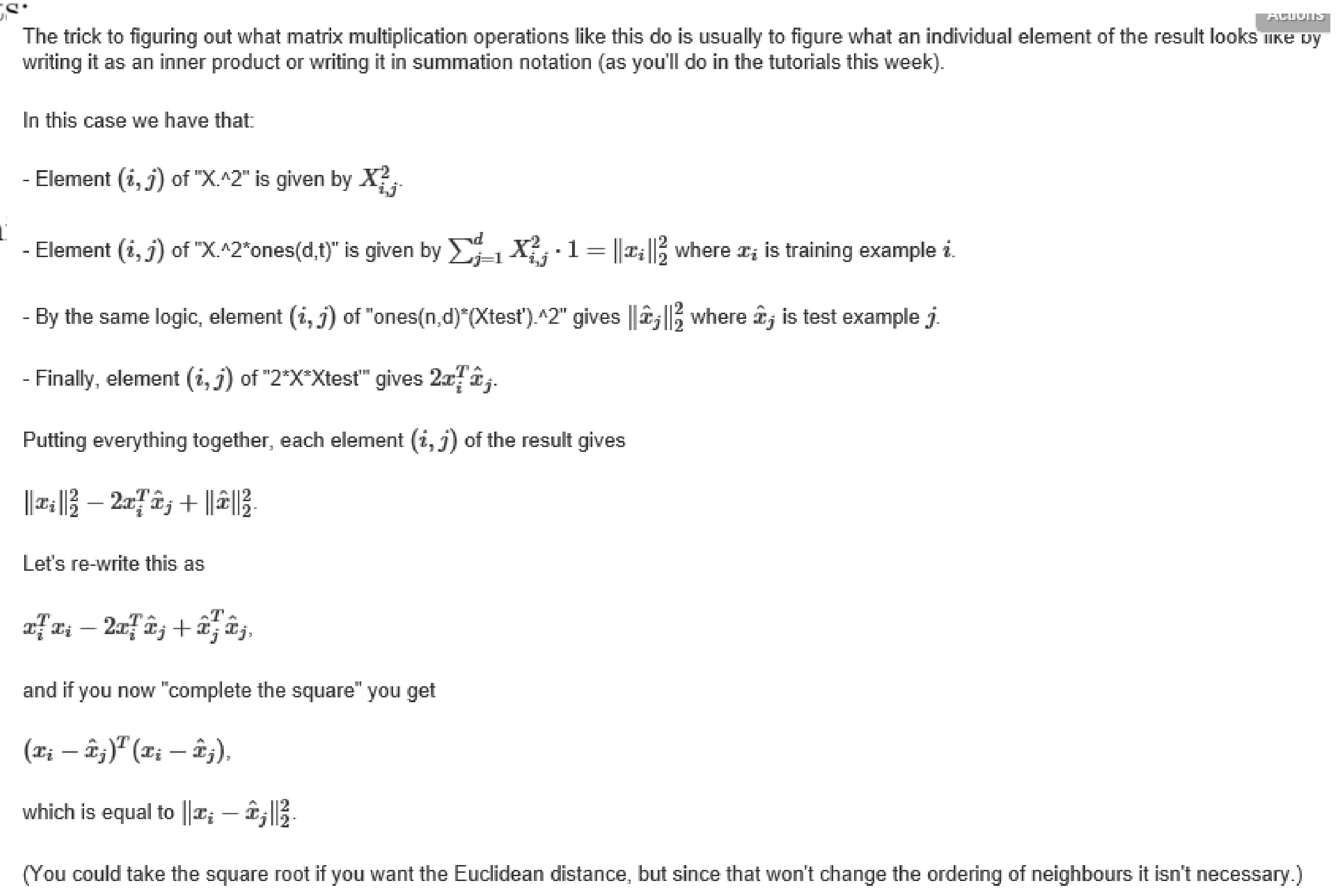
Bonus Slide: Computing all distances in Matlab



Note: Matlab can be slow at executing operations in 'for' loops, but allows extremely-fast hardware-dependent vector and matrix operations. By taking advantage of SIMD registers and multiple cores (and faster matrix-multiplication algorithms), vector and matrix operations in Matlab will often be several times faster than if you implemented them yourself in a fast language like C. If you find that calculating the Euclidean distances between all pairs of points takes too long, the following code will form a matrix containing the squared Euclidean distances between all training and test points

```
[n,d] = size(X);
[t,d] = size(Xtest);
D = X.^2*ones(d,t) + ones(n,d)*(Xtest').^2 - 2*X*Xtest';
```

Element $D(i,j)$ gives the squared Euclidean distance between train



The trick to figuring out what matrix multiplication operations like this do is usually to figure what an individual element of the result looks like by writing it as an inner product or writing it in summation notation (as you'll do in the tutorials this week).

In this case we have that:

- Element (i,j) of "X.^2" is given by $X_{i,j}^2$.

- Element (i,j) of "X.^2*ones(d,t)" is given by $\sum_{j=1}^d X_{i,j}^2 \cdot 1 = \|x_i\|_2^2$ where x_i is training example i.

- By the same logic, element (i,j) of "ones(n,d)*(Xtest').^2" gives $\|\hat{x}_j\|_2^2$ where $\hat{x}_j$ is test example j.

- Finally, element (i,j) of "2*X*Xtest'" gives $2x_i^T\hat{x}_j$.

Putting everything together, each element (i,j) of the result gives

$\|x_i\|_2^2 - 2x_i^T\hat{x}_j + \|\hat{x}\|_2^2$.

Let's re-write this as

$x_i^Tx_i - 2x_i^T\hat{x}_j + \hat{x}_j^T\hat{x}_j$,

and if you now "complete the square" you get

$(x_i - \hat{x}_j)^T(x_i - \hat{x}_j)$,

which is equal to $\|x_i - \hat{x}_j\|_2^2$.

(You could take the square root if you want the Euclidean distance, but since that won't change the ordering of neighbours it isn't necessary.)