CPSC 340:
Machine Learning and Data Mining

Non-Parametric Models

Summer 2021

In This Lecture

- Laplace Smoothing (5 minutes)
- Decision Theory (10 minutes)
- K-Nearest Neighbours (30 minutes)

Coming Up Next

LAPLACE SMOOTHING

Naïve Bayes

- Naïve Bayes formally:

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

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

same for all y_i values

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

Only needs easy probabilities.

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

Laplace Smoothing

- Our estimate of p('lactase' = 1| 'spam') is:

$$\frac{\#\text{spam messages with lactase} + 1}{\#\text{spam messages} + 2}$$

↖ # classes y.

  - But there is a problem if you have no spam messages with lactase:
    - p('lactase' | 'spam') = 0, so spam messages with lactase automatically get through.

  - Common fix is Laplace smoothing:
    - Add 1 to numerator,
      and 2 to denominator (for binary features).
      - Acts like a "fake" spam example that has lactase,
        and a "fake" spam example that doesn't.

$$\frac{(\#\text{Spam messages with lactase}) + 1}{(\#\text{spam messages}) + 2}$$

Laplace Smoothing

- Laplace smoothing:

$$\frac{(\#\text{spam messages with lactase}) + 1}{(\#\text{spam messages}) + 2}$$

$\beta = 1$

$\beta = 1000000 \gg\gg n$

$2 \cdot 1$

$2 \cdot \beta$

k classes in y.

$\beta \cdot k$

  - Typically you do this for all features.
    - Helps against overfitting by biasing towards the uniform distribution.
- A common variation is to use a real number β rather than 1.
  - Add 'βk' to denominator if feature has 'k' possible values (so it sums to 1).

$\beta > 0$

$$P(x_{ij} = c \mid y_i = class) \approx \frac{(\text{number of examples in class with } x_{ij} = c) + \beta}{(\text{number of examples in class}) + \beta K}$$

$\beta \to 0$

$\frac{\beta}{\beta k} = \frac{1}{k}$

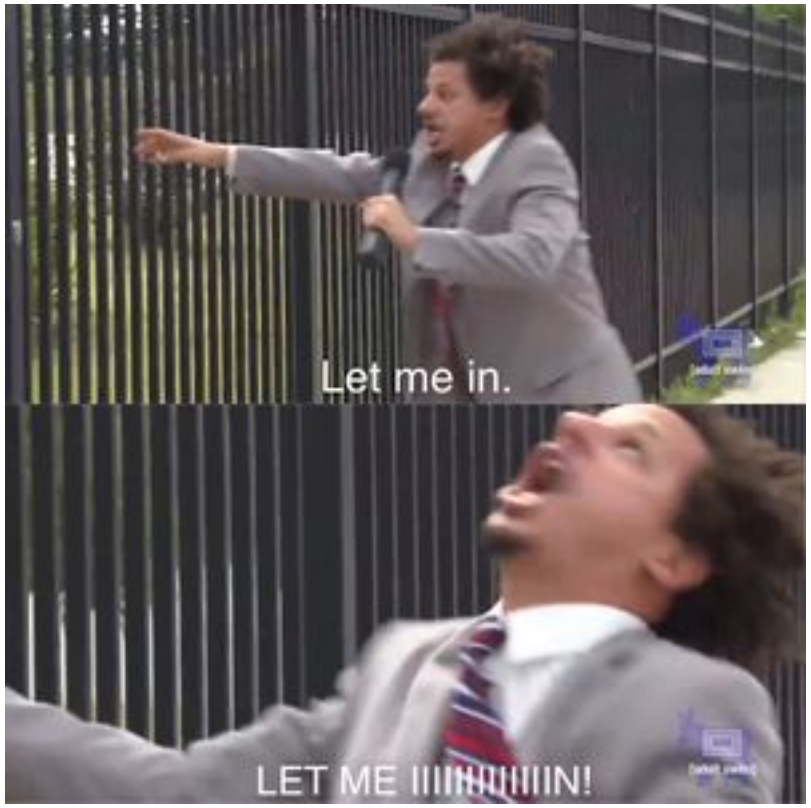
“Regularization”

- Laplace smoothing is a special case of regularization.
  - Regularization: control the complexity of model
  - We will see more examples of regularization in this class.

Coming Up Next

DECISION THEORY

My mother's day email when my mom's spam filter throws it out



Decision Theory

- Are we equally concerned about “spam” vs. “not spam”?
- True positives, false positives, false negatives, true negatives:

$\tilde{y}$

$\hat{y}$ 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:

$\tilde{y}$

$\hat{y}$	Predict / True	True 'spam'	True 'not spam'
	Predict 'spam'	0	100
	Predict 'not spam'	10	0

- Instead of most probable label, take $\hat{y}_i$ minimizing expected cost:

$$\mathbb{E}\left[\text{cost}(\hat{y}_i, \tilde{y}_i)\right]$$

expectation of model with respect to $\tilde{y}_i$

cost of predicting $\hat{y}_i$ if it's really $\tilde{y}_i$

- Even if "spam" has a higher probability, predicting "spam" might have a expected higher cost.

$\tilde{y}$ (probability, cost)

$\hat{y}$	Predict / True	True 'spam'	True 'not spam'
	Predict 'spam'	(0.6, 0)	(0.4, 100)
	Predict 'not spam'	(0.6, 10)	(0.4, 0)

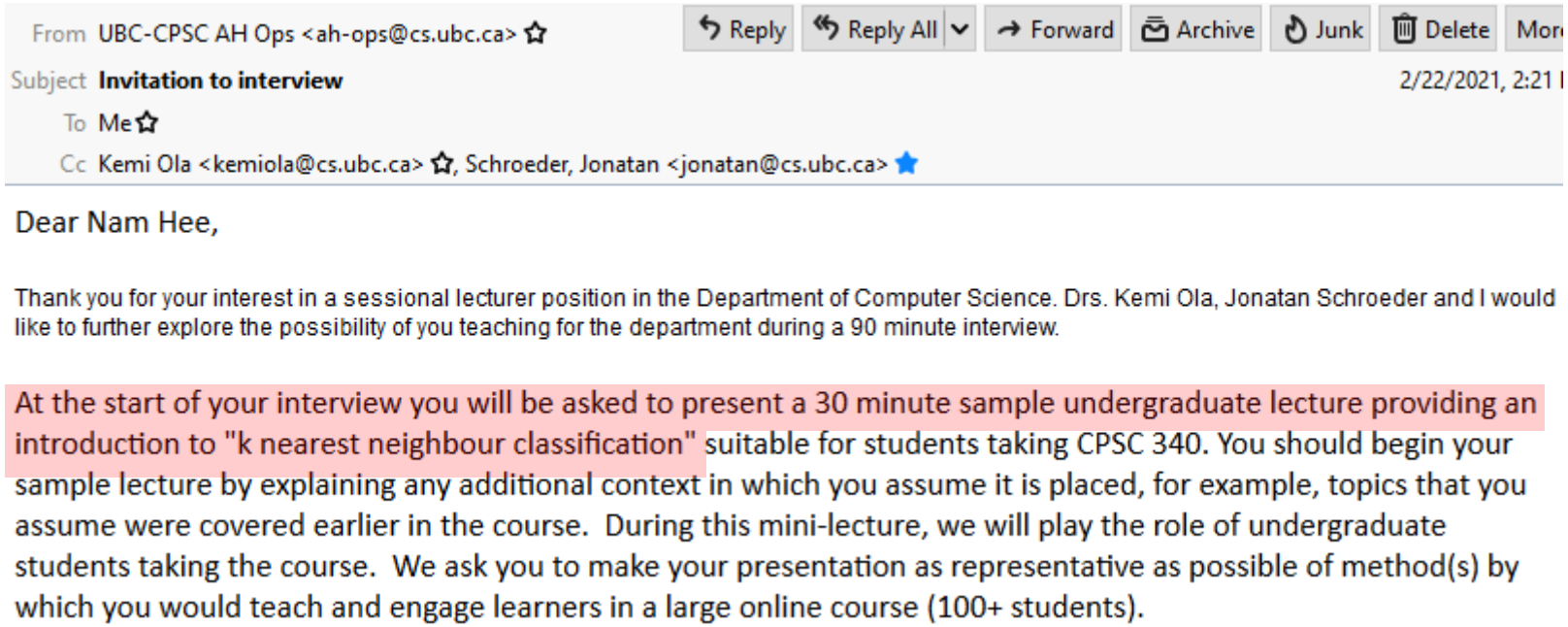
- Consider a test example we have $p(\tilde{y}_i = \text{"spam"} \mid \tilde{x}_i) = 0.6$, then:

$$\mathbb{E}[\text{cost}(\hat{y}_i = \text{"spam"}, \tilde{y}_i)] = p(\tilde{y}_i = \text{"spam"} \mid \tilde{x}_i)\,\text{cost}(\hat{y}_i = \text{"spam"}, \tilde{y}_i = \text{"spam"}) + p(\tilde{y}_i = \text{"not spam"} \mid \tilde{x}_i)\,\text{cost}(\hat{y}_i = \text{"spam"}, \tilde{y}_i = \text{"not spam"})$$

$$= (0.6)(0) + (0.4)(100) = 40$$

- Even though "spam" is more likely, we should predict "not spam".

$$\mathbb{E}[\text{cost}(\hat{y}_i = \text{"not spam"}, \tilde{y}_i)] = (0.6)(10) + (0.4)(0) = 6$$



From UBC-CPSC AH Ops <ah-ops@cs.ubc.ca>

Reply | Reply All | Forward | Archive | Junk | Delete | More

Subject **Invitation to interview** 2/22/2021, 2:21

To Me

Cc Kemi Ola <kemiola@cs.ubc.ca>, Schroeder, Jonatan <jonatan@cs.ubc.ca>

Dear Nam Hee,

Thank you for your interest in a sessional lecturer position in the Department of Computer Science. Drs. Kemi Ola, Jonatan Schroeder and I would like to further explore the possibility of you teaching for the department during a 90 minute interview.

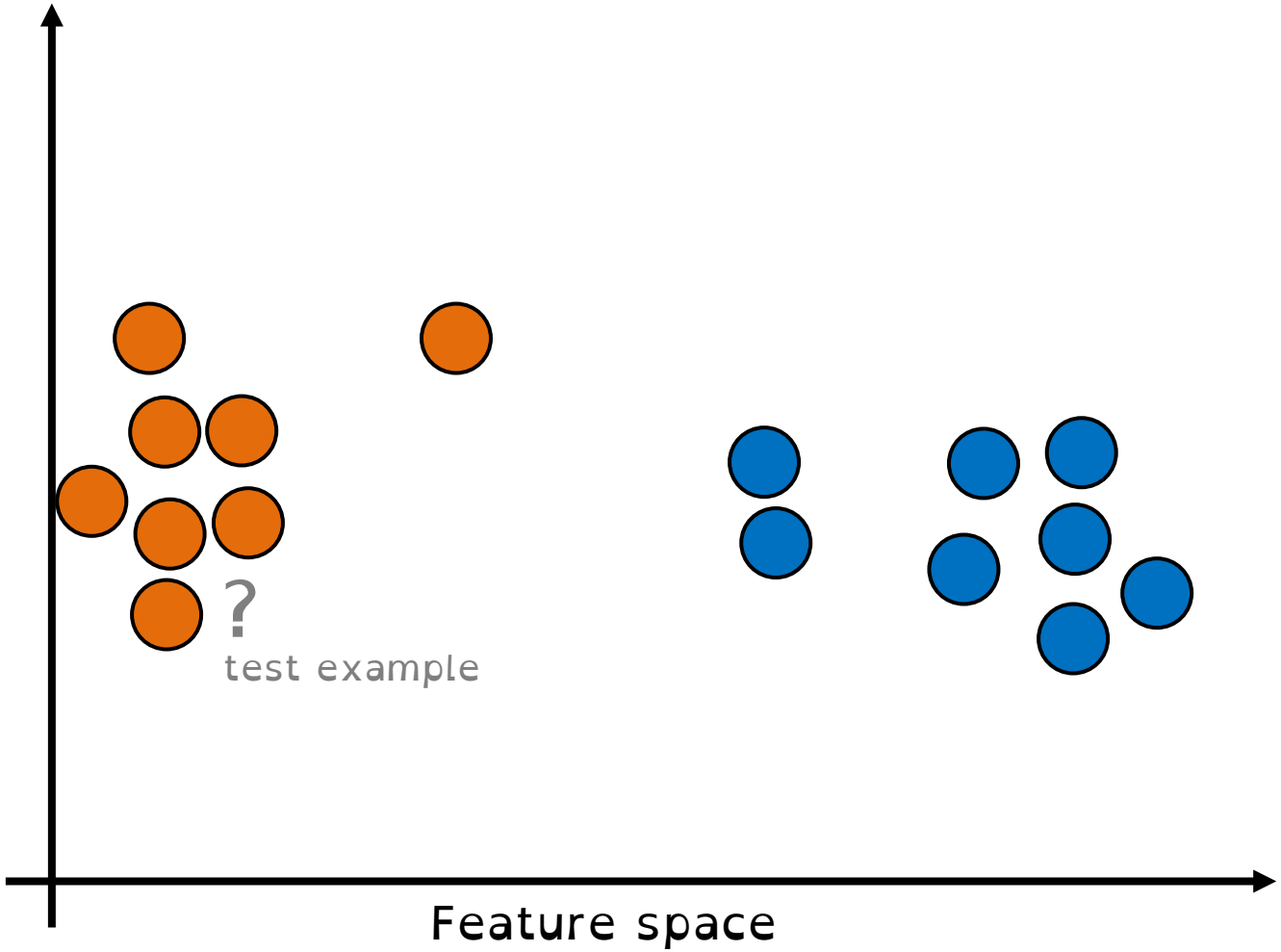
At the start of your interview you will be asked to present a 30 minute sample undergraduate lecture providing an introduction to "k nearest neighbour classification" suitable for students taking CPSC 340. You should begin your sample lecture by explaining any additional context in which you assume it is placed, for example, topics that you assume were covered earlier in the course. During this mini-lecture, we will play the role of undergraduate students taking the course. We ask you to make your presentation as representative as possible of method(s) by which you would teach and engage learners in a large online course (100+ students).

Coming Up Next

K-NEAREST NEIGHBOURS

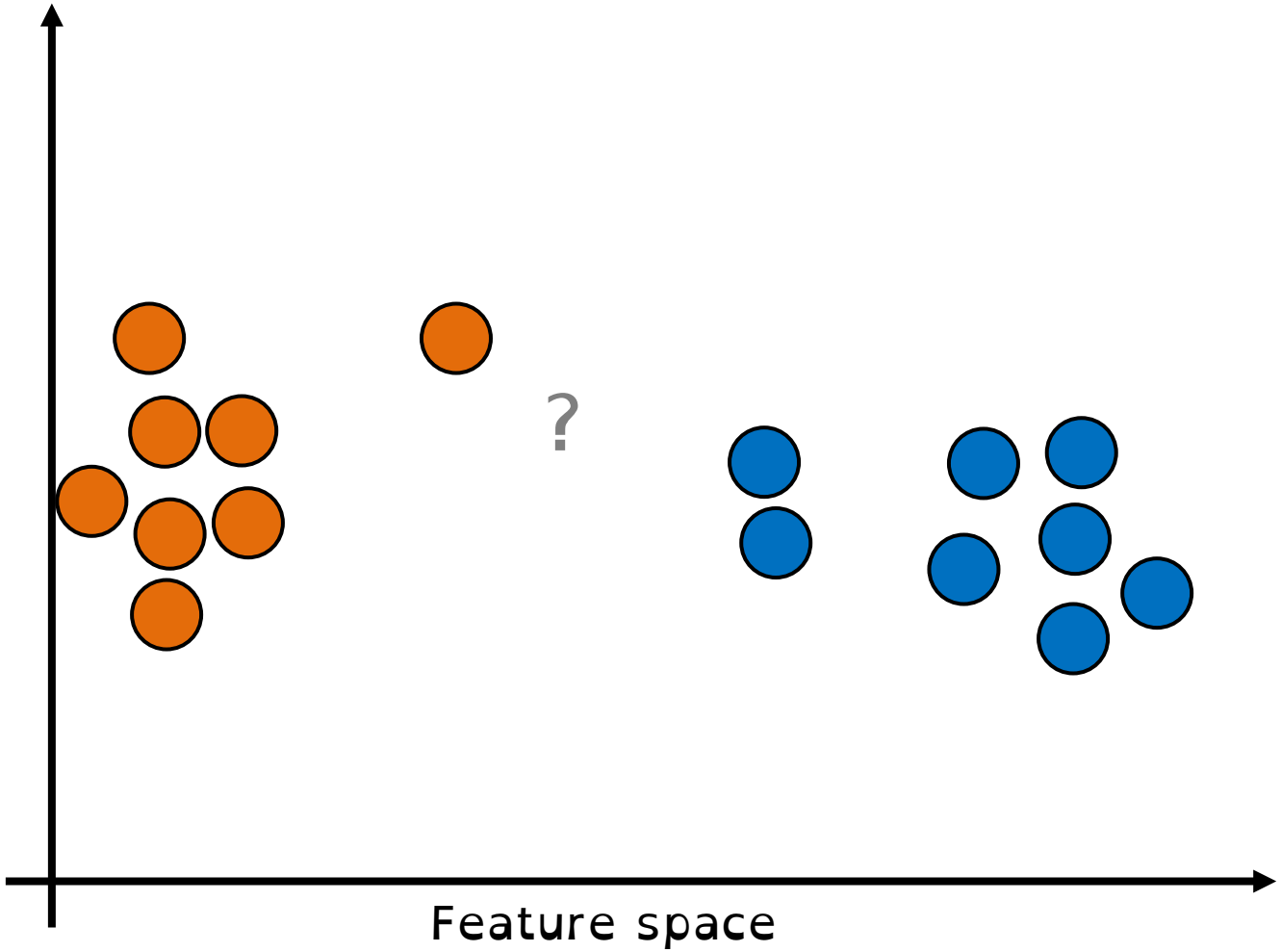
What is K-Nearest Neighbours (KNN)?

- Can you tell whether ? is orange or blue?



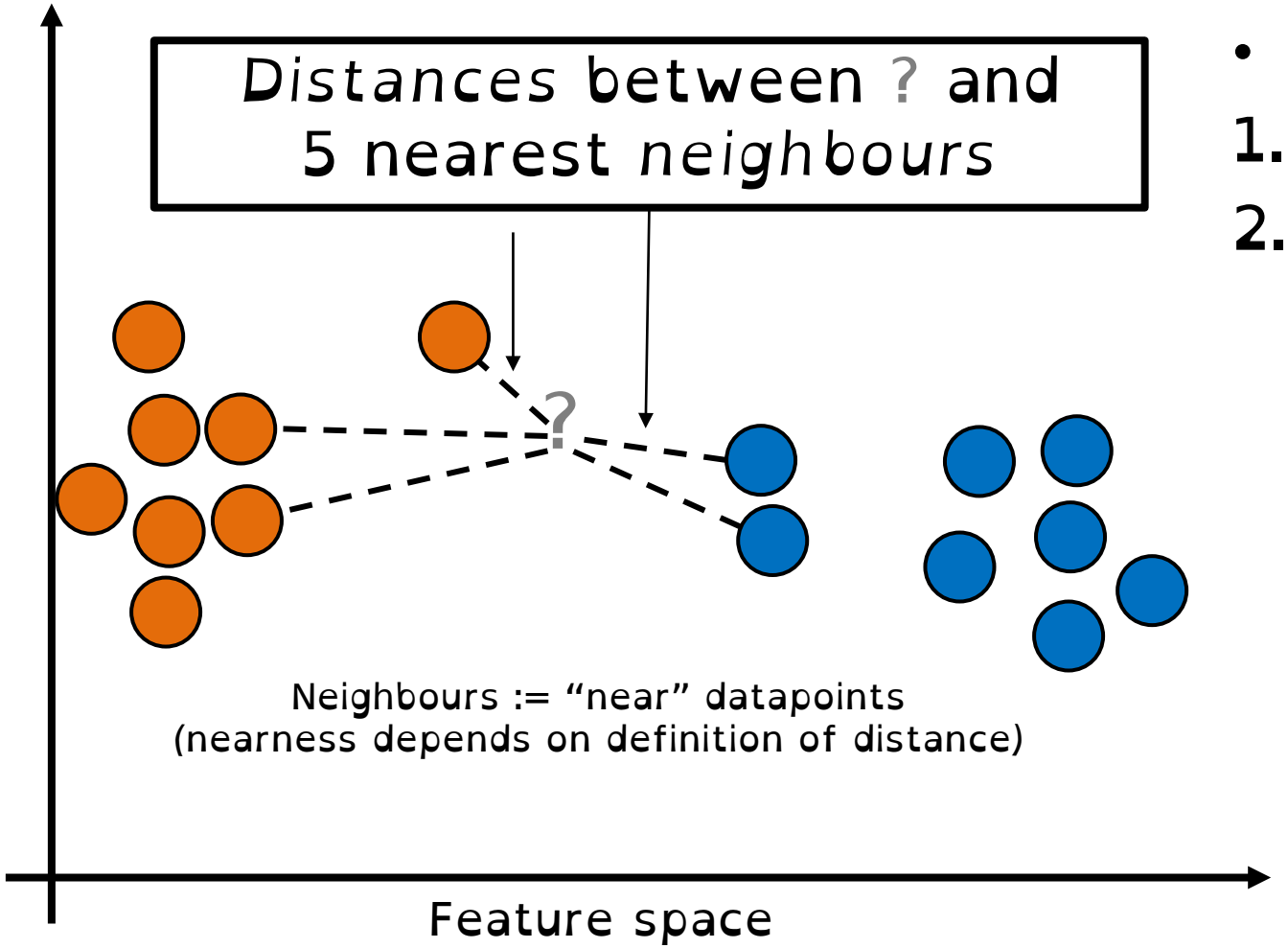
What is K-Nearest Neighbours (KNN)?

- Can you tell whether ? is orange or blue?



What is K-Nearest Neighbours (KNN)?

- Can you tell whether ? is orange or blue?

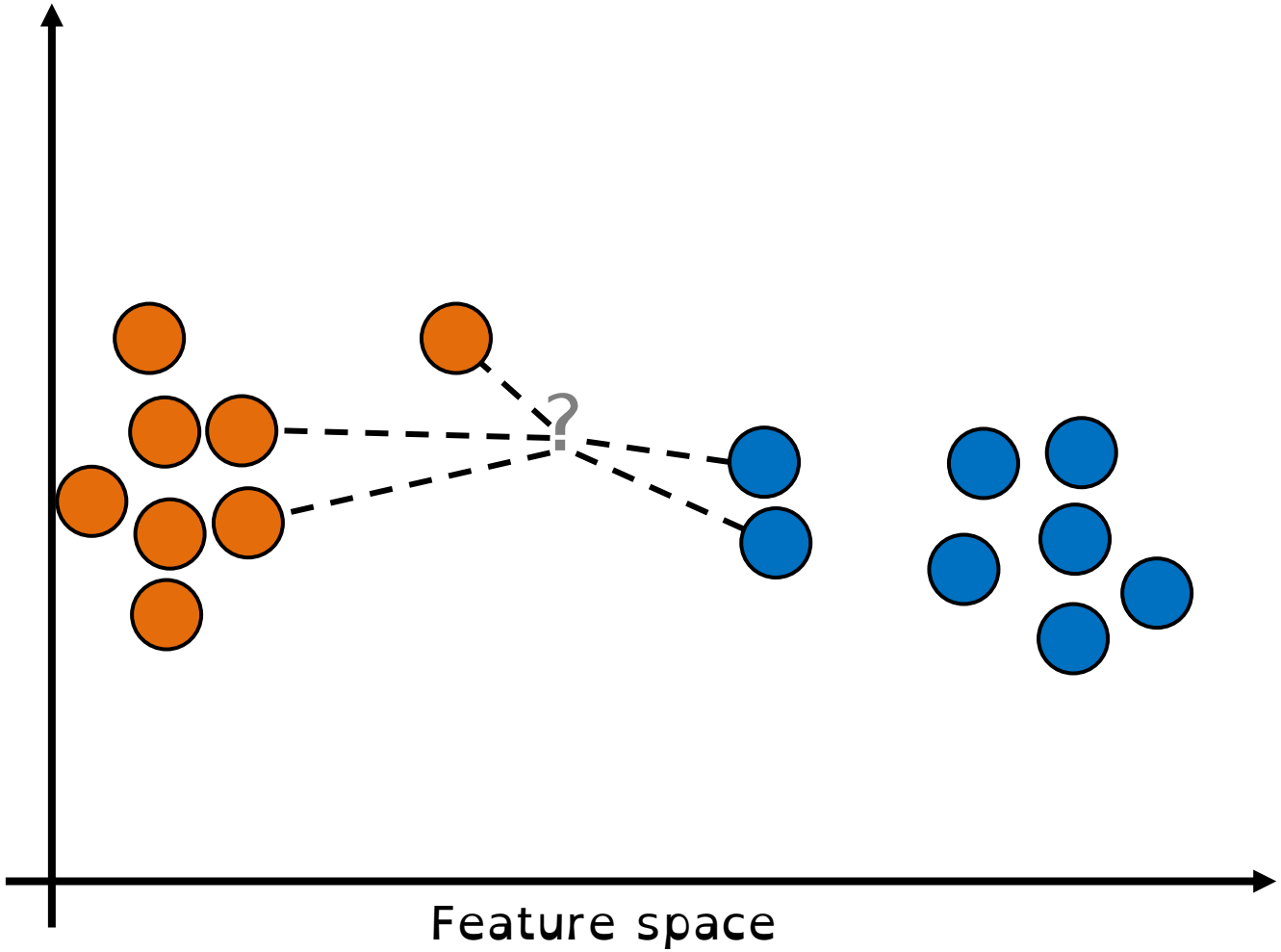


- You probably did this:

1. Look at the *neighbours* of ?
2. See if there are more oranges or blues in the neighbourhood

What is K-Nearest Neighbours (KNN)?

- Can you tell whether ? is orange or blue?



Q: How many neighbours should we look at?

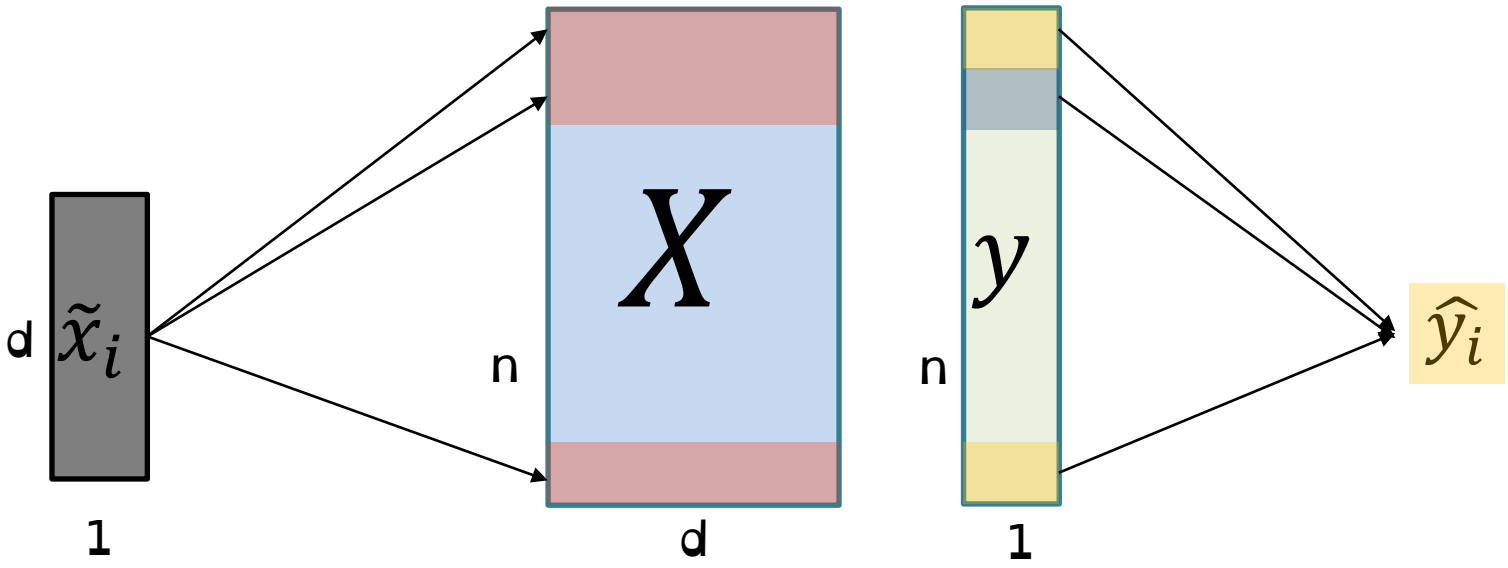
1-nearest neighbour → orange
3-nearest neighbours → blue
5-nearest neighbours → orange

Coming Up Next:

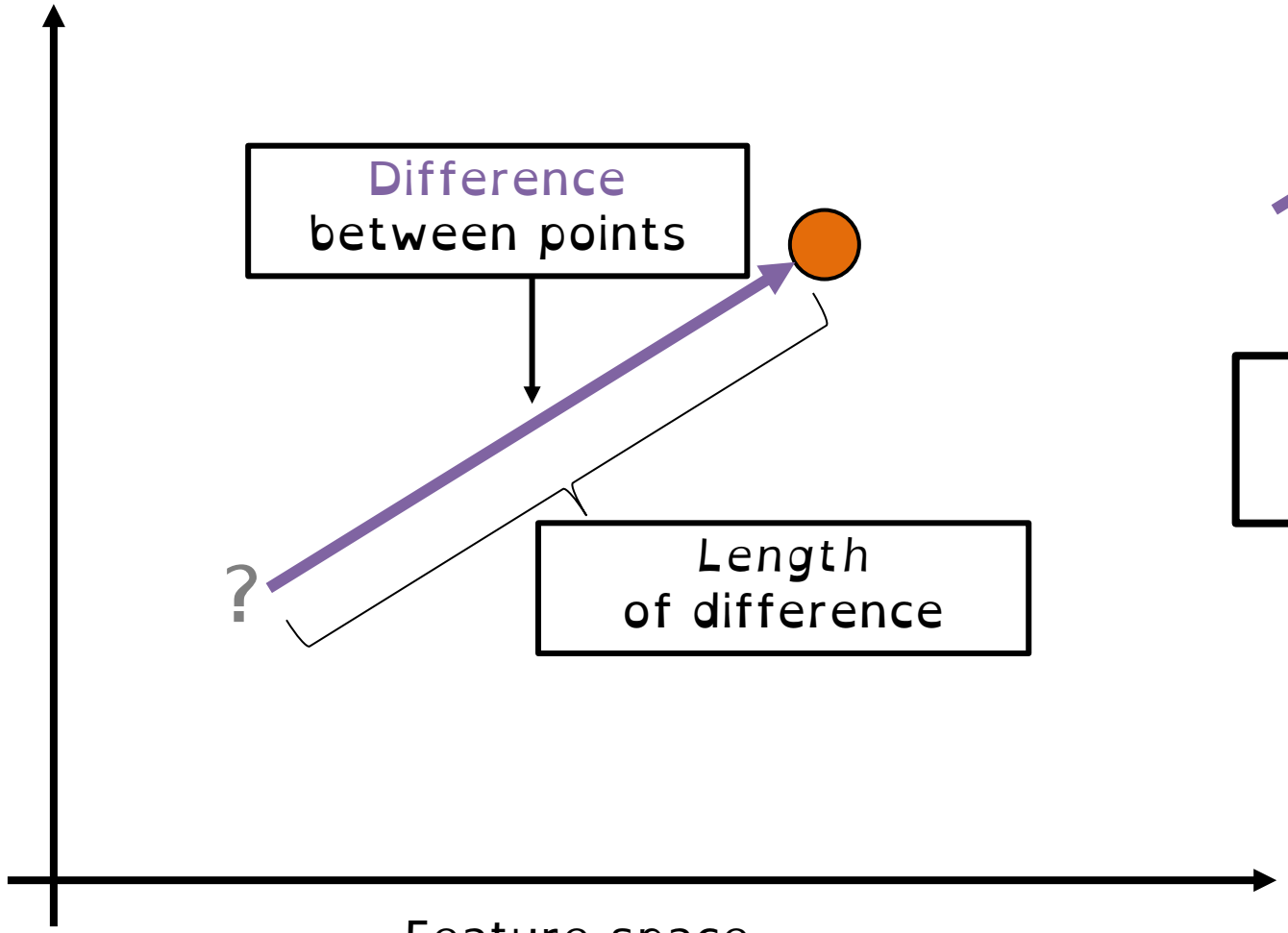
MORE FORMAL DISCUSSION OF KNN

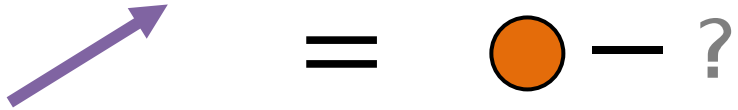
What is K-Nearest Neighbours (KNN)?

- An old/simple classifier: k-nearest neighbours (KNN).
- To classify an example $\tilde{x}_i$:
  1. Find the 'k' training examples x_i that are "nearest" to $\tilde{x}_i$.
  2. Classify using the most common label of "nearest" training examples.



Defining "Distance"





Q: How do we measure the length of a vector?

Defining "Distance" with "Norms"

- A common way to define the "distance" between examples:
  - Take the "norm" of the difference between feature vectors.
- Norms are a way to measure the "length" of a vector.
  - The most common norm is the "L2-norm" (or "Euclidean norm"):

$$\|r\|_2 = \sqrt{\sum_{j=1}^{d} r_j^2}$$

$$\|x_i - \tilde{x}_{\tilde{i}}\|_2 = \sqrt{\sum_{j=1}^{d} (x_{ij} - \tilde{x}_{\tilde{i}j})^2}$$

train example

test example

"L_2-norm"

  - Here, the "norm" of the difference is the standard Euclidean distance.
  - There are many other ways to define distance (bonus slides)

L2-norm, L1-norm, and L∞-Norms.

- The three most common norms: L2-norm, L1-norm, and L∞-norm.
  - Visualizing 2D cases:

L_2 or "Euclidean" norm.

$$\|r\|_2 = \sqrt{r_1^2 + r_2^2}$$

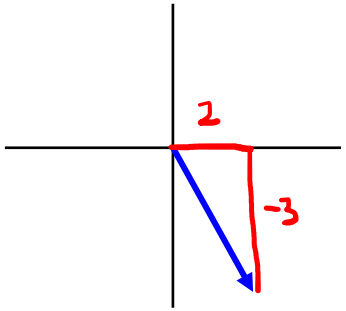
L_1 or "Manhattan" norm:

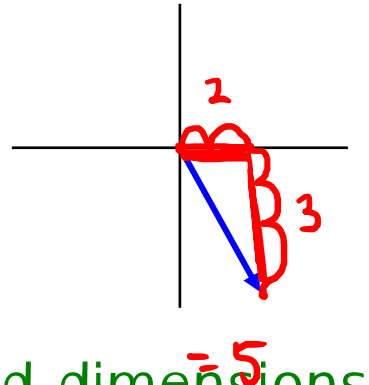
$$\|r\|_1 = |r_1| + |r_2|$$

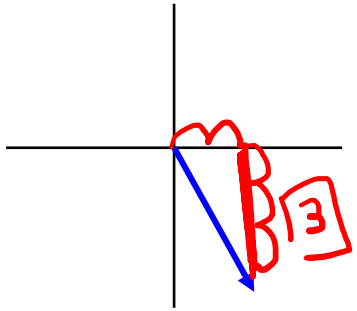
L_∞ or "max" norm:

$$\|r\|_\infty = \max\{|r_1|, |r_2|\}$$

$$r = \begin{bmatrix} 2 \\ -3 \end{bmatrix}$$







  - Definitions of these norms in d-dimensions.

L_2: $\|r\|_2 = \sqrt{\sum_{j=1}^{d} r_j^2}$

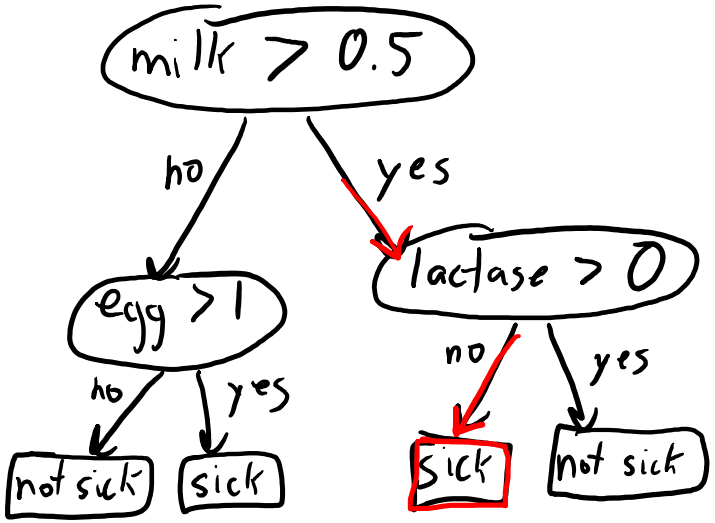
L_1: $\|r\|_1 = \sum_{j=1}^{d} |r_j|$

L_∞: $\max_j \{|r_j|\}$

Infinite Series Video

Decision Trees vs. KNN

Trained decision tree



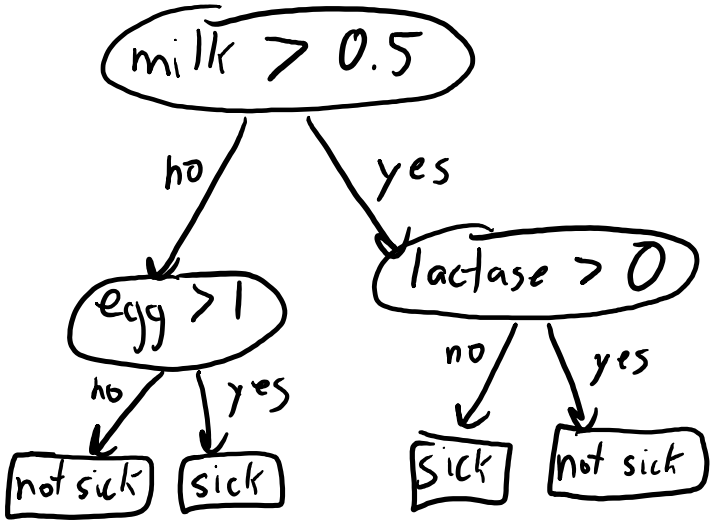
$\hat{x}_1 = (\text{milk} = 0.6, \text{egg} = 2, \text{lactase} = 0, ??)$

Q: What does this decision tree predict?

Sick.

Decision Trees vs. KNN

Trained decision tree



$\tilde{x}_1 = (milk = 0.6, egg = 2, lactase = 0, ??)$

Q: What does 1-nearest neighbour predict? L2-distance.

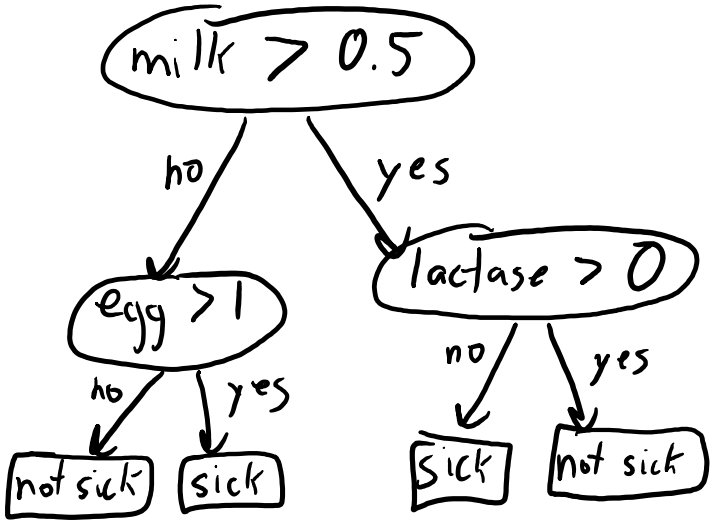
sick.

Dataset

milk	egg	lactose	sick
0	0	0	0
0.5	0	0	0
0.7	2	0	1
1.0	3	3	1
3	3	3	1

Decision Trees vs. KNN

Trained decision tree



(milk = 0.6, egg = 2, lactase = 0, ??)

Q: Why is there no model structure?
What does a trained KNN model look like?

Dataset

milk	egg	lactose	sick
0	0	0	0
0.5	0	0	0
0.7	2	0	1
1.0	3	3	1
3	3	3	1

How Do We "Train" KNN?

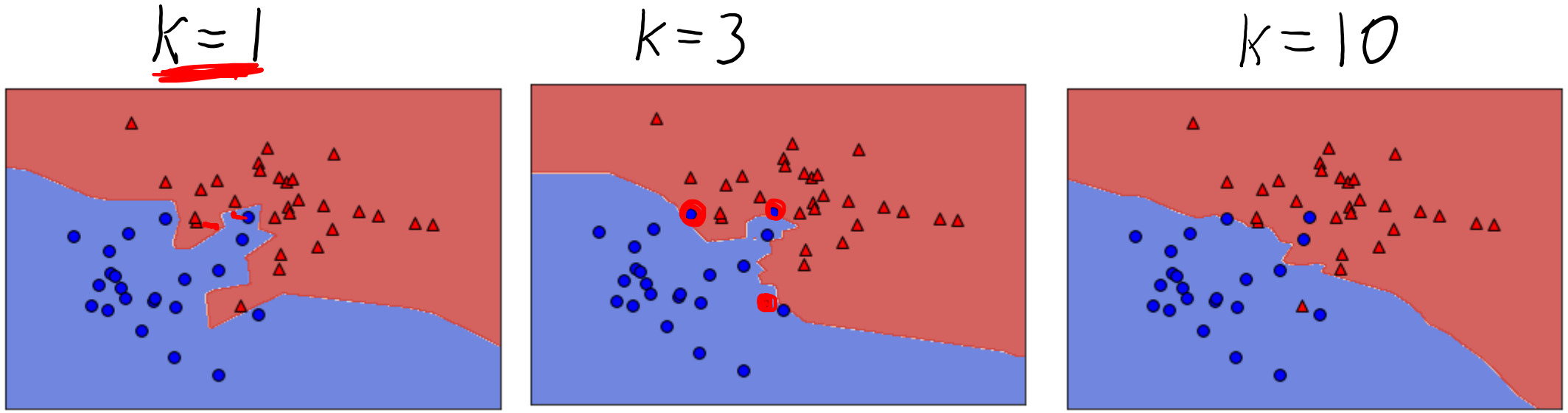
- There is no training phase in KNN ("lazy" learning).
  - You just store the training data.
  - Costs O(1) if you use a pointer.

- But predictions are expensive: O(nd) to classify 1 test example.
  - Need to do O(d) distance calculation for all 'n' training examples.
  - So prediction time grows with number of training examples.
    - Tons of work on reducing this cost (we'll discuss this later).

- But storage is expensive: needs O(nd) memory to store 'X' and 'y'.
  - So memory grows with number of training examples.
  - When storage depends on 'n', we call it a non-parametric model.

How Does ‘k’ Affect KNN’s Behaviour?

- With large ‘k’ (hyper-parameter), KNN model will be very simple.
  - With k=n, you just return the mode of the labels.
  - Model gets more complicated as ‘k’ decreases. (WHY?)
    - The 1st nearest neighbour is very sensitive to the trend of the data



- Effect of ‘k’ on fundamental trade-off:
  - As ‘k’ grows, training error increases and approximation error decreases.

When you train KNN

distance (X_1, X_2)

$\|X_1 - X_2\|_2$

$= \sqrt{[w_1(X_{11} - X_{21})]^2 + [w_2(X_{12} - X_{22})]^2}$

Coming Up Next

~~NON-PARAMETRIC MODELS~~

$\tilde{X}_i = d\begin{bmatrix} \\ \\ \end{bmatrix}$ $X_j = d\begin{bmatrix} \\ \\ \end{bmatrix}$ $\|\tilde{X}_i - X_j\|_2$

$(\tilde{X}_i - X_j) = d\begin{bmatrix} \tilde{X}_{i1} - X_{j1} \\ \tilde{X}_{i2} - X_{j2} \\ \vdots \end{bmatrix}$

$O(d)$

n distance computations.

$O(d)$ for one distance.

$X = \begin{bmatrix} & & \\ & n & \\ & & \end{bmatrix}$ d

$\tilde{X}_i = d\begin{bmatrix} \\ \\ \end{bmatrix}$

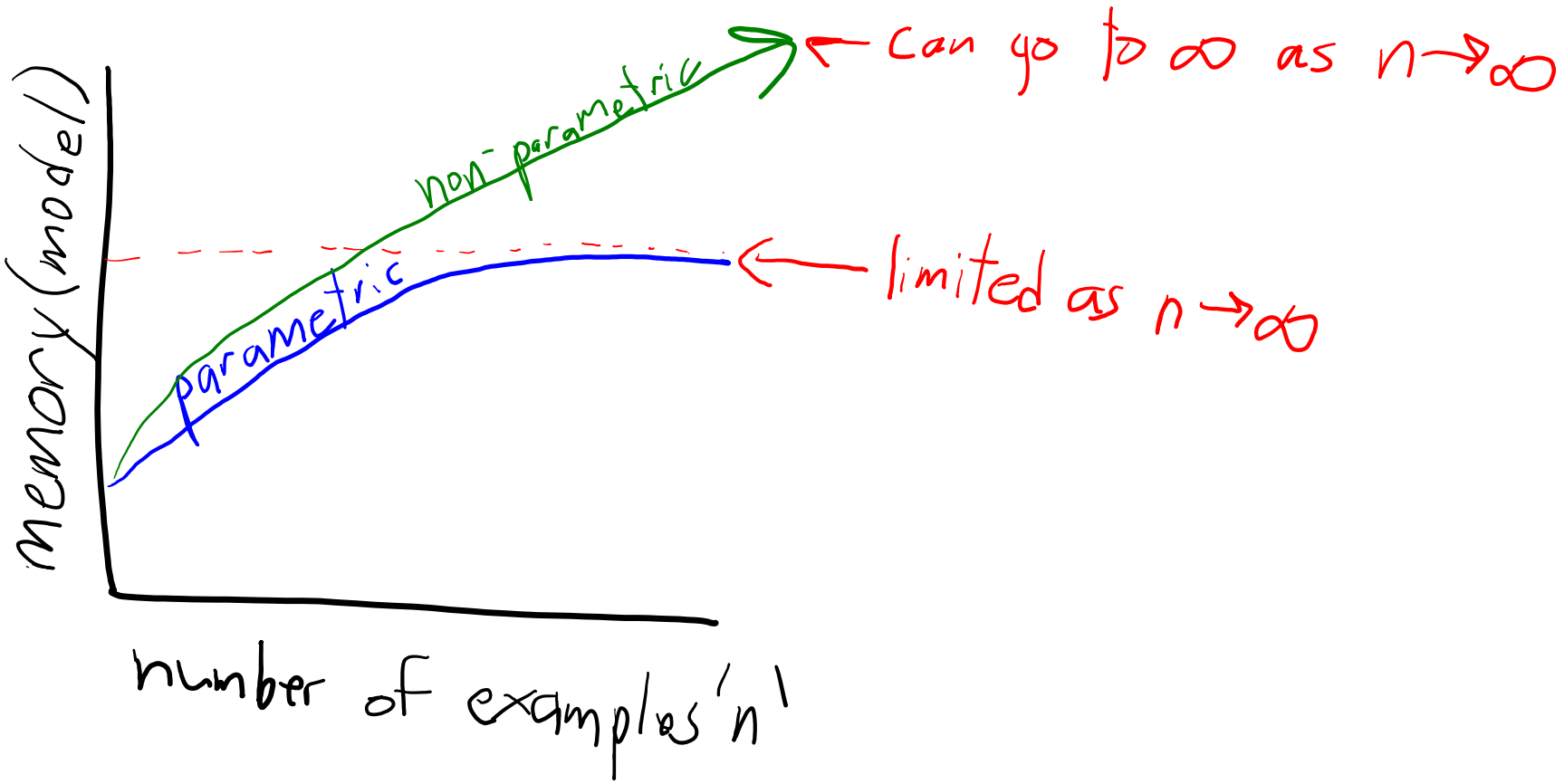
Parametric vs. Non-Parametric

- Parametric models:
  - Have fixed number of parameters: trained "model" size is O(1) in terms 'n'.
    - E.g., naïve Bayes just stores counts.
    - E.g., fixed-depth decision tree just stores rules for that depth.
  - 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., KNN stores all the training data, so size of "model" is O(nd).
    - E.g., decision tree whose depth *grows with the number of examples*.

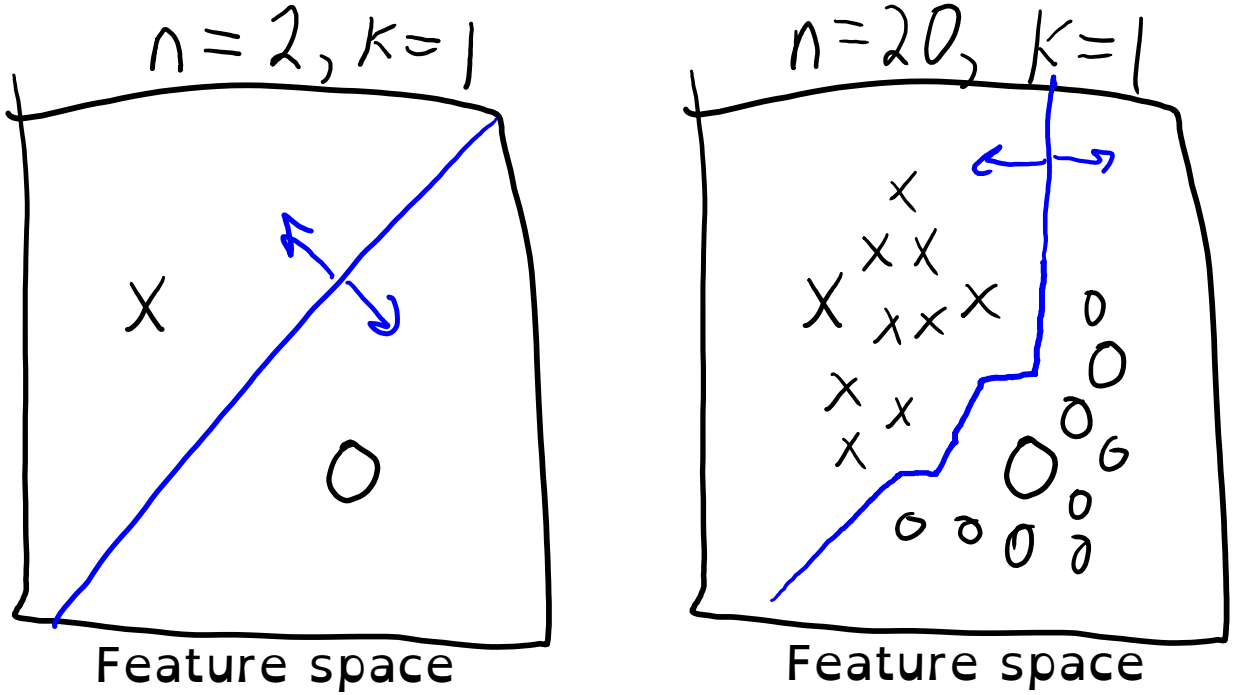
Parametric vs. Non-Parametric Models

- Parametric models have bounded memory.
- Non-parametric models can have unbounded memory.



Effect of ‘n’ in KNN.

- With a small ‘n’, KNN model will be very simple.



- Model gets more complicated as ‘n’ increases.
  - Requires more memory, but detects subtle differences between examples.

Q: Does that mean we overfit with large n?

Consistency of KNN ('n' going to '∞')

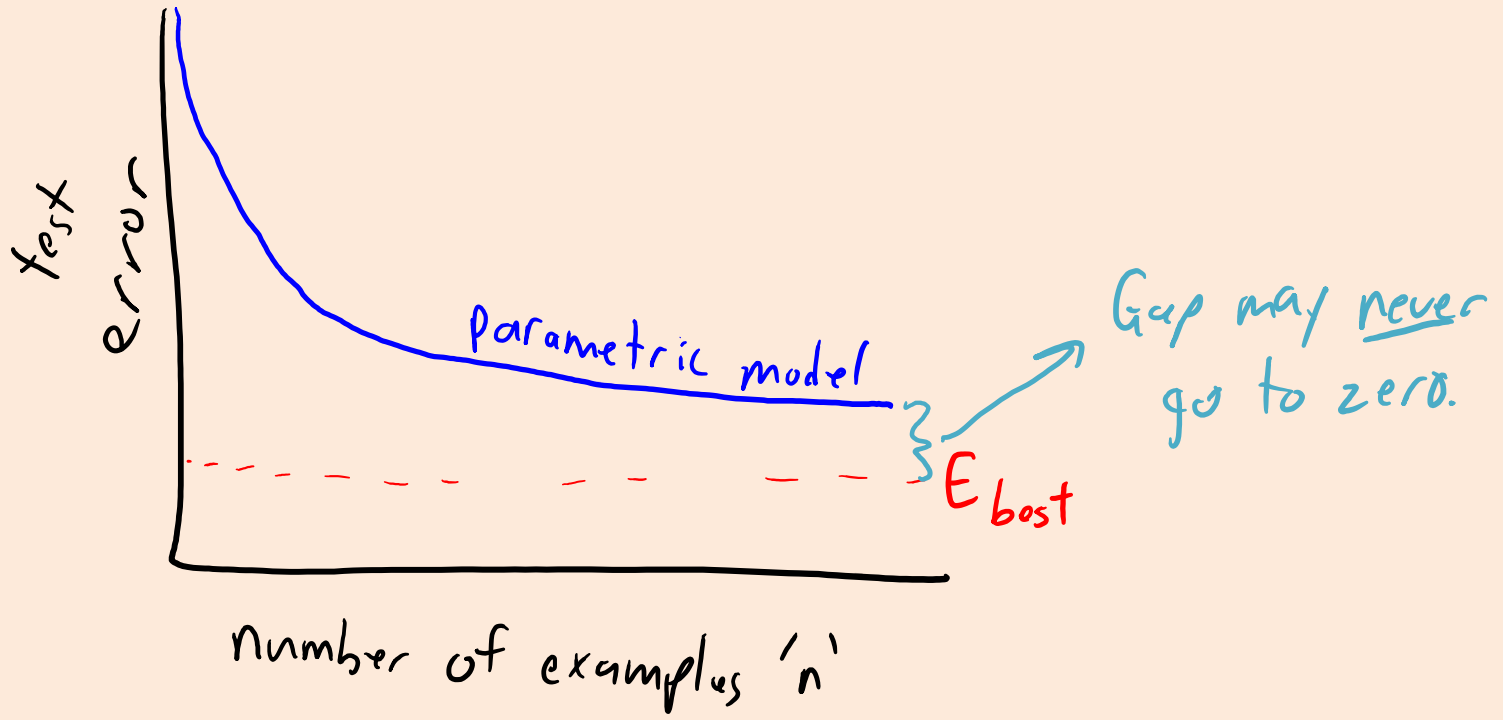
- KNN has appealing consistency properties:
  - As 'n' goes to ∞, KNN test error is less than twice best possible error.
    - For fixed 'k' and binary labels (under mild assumptions).

- Stone's Theorem: KNN is "universally consistent".
  - If k/n goes to zero and 'k' goes to ∞, converges to the best possible error.
    - For example, k = log(n).
    - First algorithm shown to have this property.

- Does Stone's Theorem violate the no free lunch theorem?
  - No: it requires a continuity assumption on the labels.
  - Consistency says nothing about finite 'n' (see "Dont Trust Asymptotics").

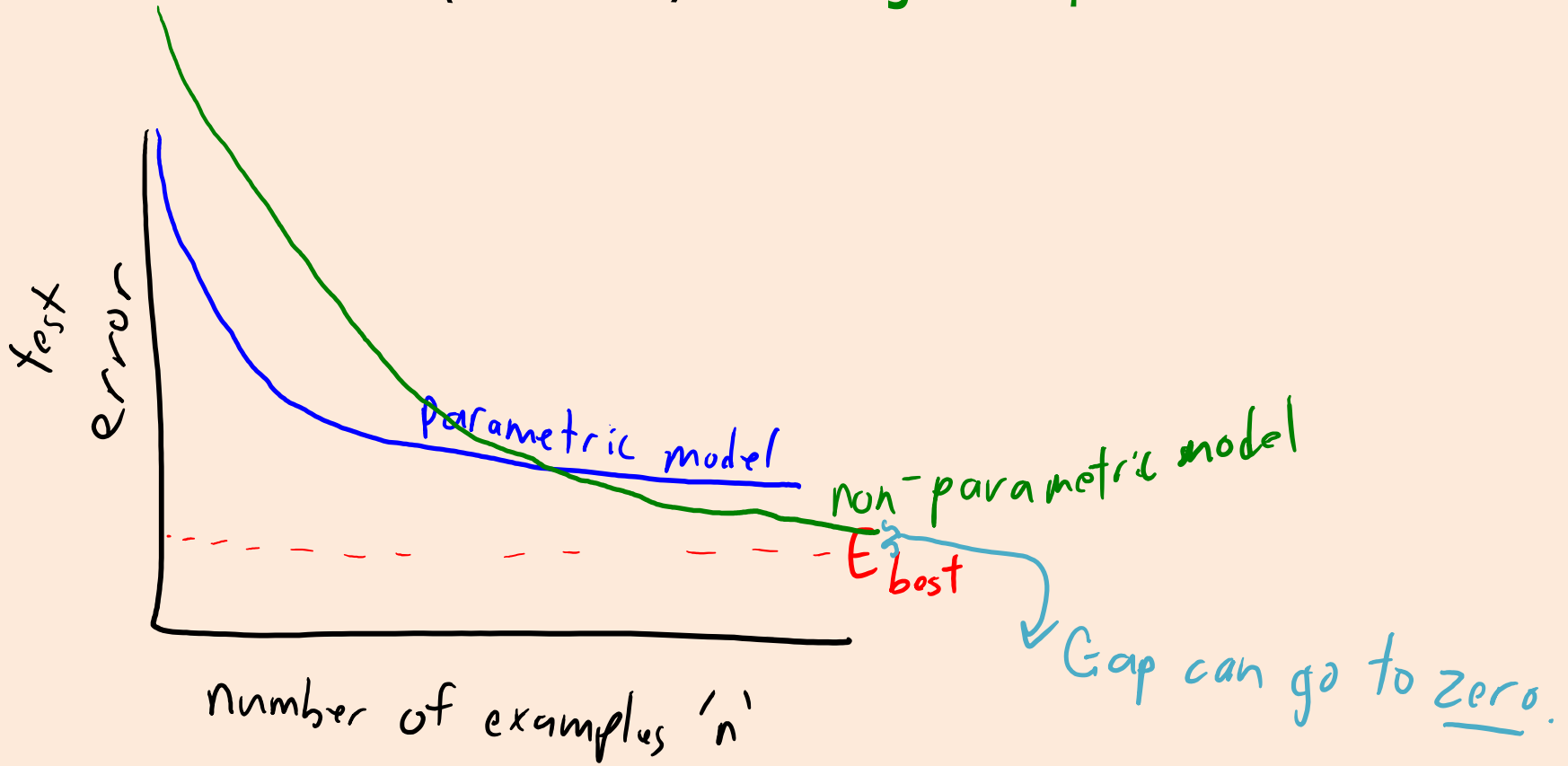
Parametric vs. Non-Parametric Models

- With parametric models, there is an accuracy limit.
  - Even with infinite 'n', may not be able to achieve optimal error (E_{best}).



Parametric vs. Non-Parametric Models

- With parametric models, there is an accuracy limit.
  - Even with infinite 'n', may not be able to achieve optimal error (E_{best}).
- Many non-parametric models (like KNN) converge to optimal error.



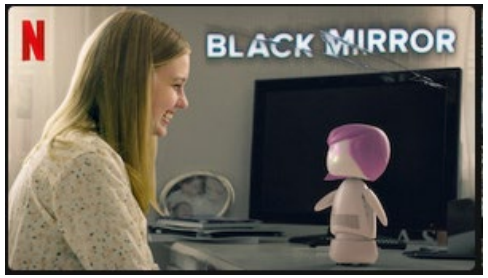
Coming Up Next

CURSE OF DIMENSIONALITY

Application: Netflix Show Recommendation

- I want to recommend shows according to "likes":
  - A simplified case of "recommender systems"

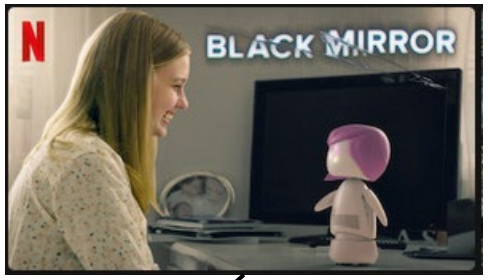
👍 👍



All shows on Netflix

Should I recommend this show?

Application: Netflix Show Recommendation



x_{ij} := 1 if user i liked show j
0 otherwise

x^1	x^2	x^3	x^4	y
1	0	1	0	0
0	0	0	0	0
0	0	0	1	1
0	0	1	0	0
1	1	1	0	1

y_i := 1 if user i liked Space Force
0 otherwise

Your preference

1	1	0	0

Q: Should I recommend Space Force?

Curse of Dimensionality

- What if I have n=5 users and d=10000 shows?
  - Much less likely that nearest neighbours have "perfect match"
  - In fact, not very likely to have similar preferences at all.
  - "Curse of dimensionality": problems with high-dimensional spaces.
  - For each additional show, we need exponentially more users to preserve the usefulness of nearest neighbours
- KNN is also problematic if features have very different scales.
  - What if feature 1 is binary and feature 2 is continuous and can be huge?
- Nevertheless, KNN is really easy to use and often hard to beat!

Summary

- Decision theory allows us to consider costs of predictions.
- K-Nearest Neighbours: use most common label of nearest examples.
  - Often works surprisingly well.
  - Suffers from high prediction and memory cost.
  - Canonical example of a "non-parametric" model.
  - Can suffer from the "curse of dimensionality". ← next
- Non-parametric models grow with number of training examples.
  - Can have appealing "consistency" properties.

will cover next

- Next Time:
  - Fighting the fundamental trade-off and Microsoft Kinect.

Review Questions

- Q1: Suppose I am learning Naïve Bayes with Laplace Smoothing.
  If I have 0 examples in class c, what probability do I assign to $p(x_i \mid y_i = c)$?

- Q2: Increasing true positives can increase false positives.
  When is this an acceptable risk?

- Q3: How do we choose the value of k in KNN?

- Q4: Is decision tree a parametric model?

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 $(y_i = c, x_{ij} = k)$

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

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

$n_{121} = 4$

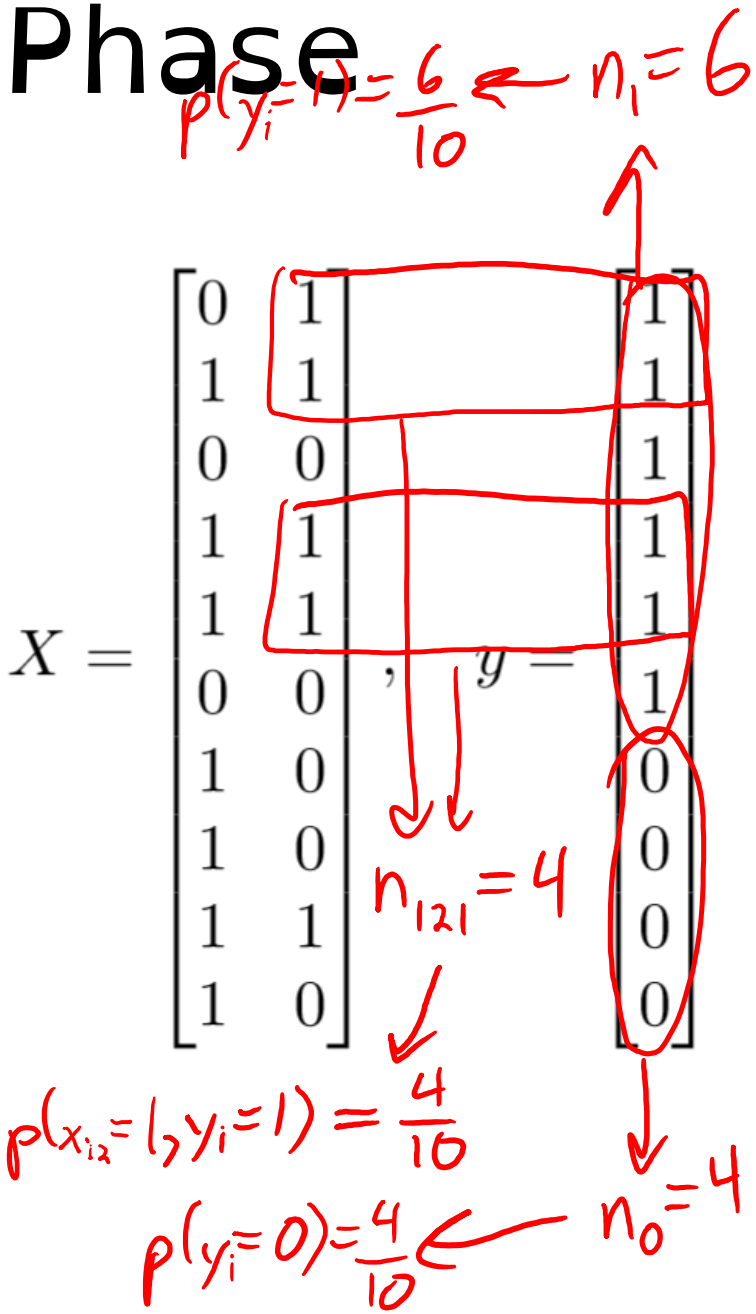
$n_0 = 4$

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

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 $\frac{n_{cjk}}{n}$

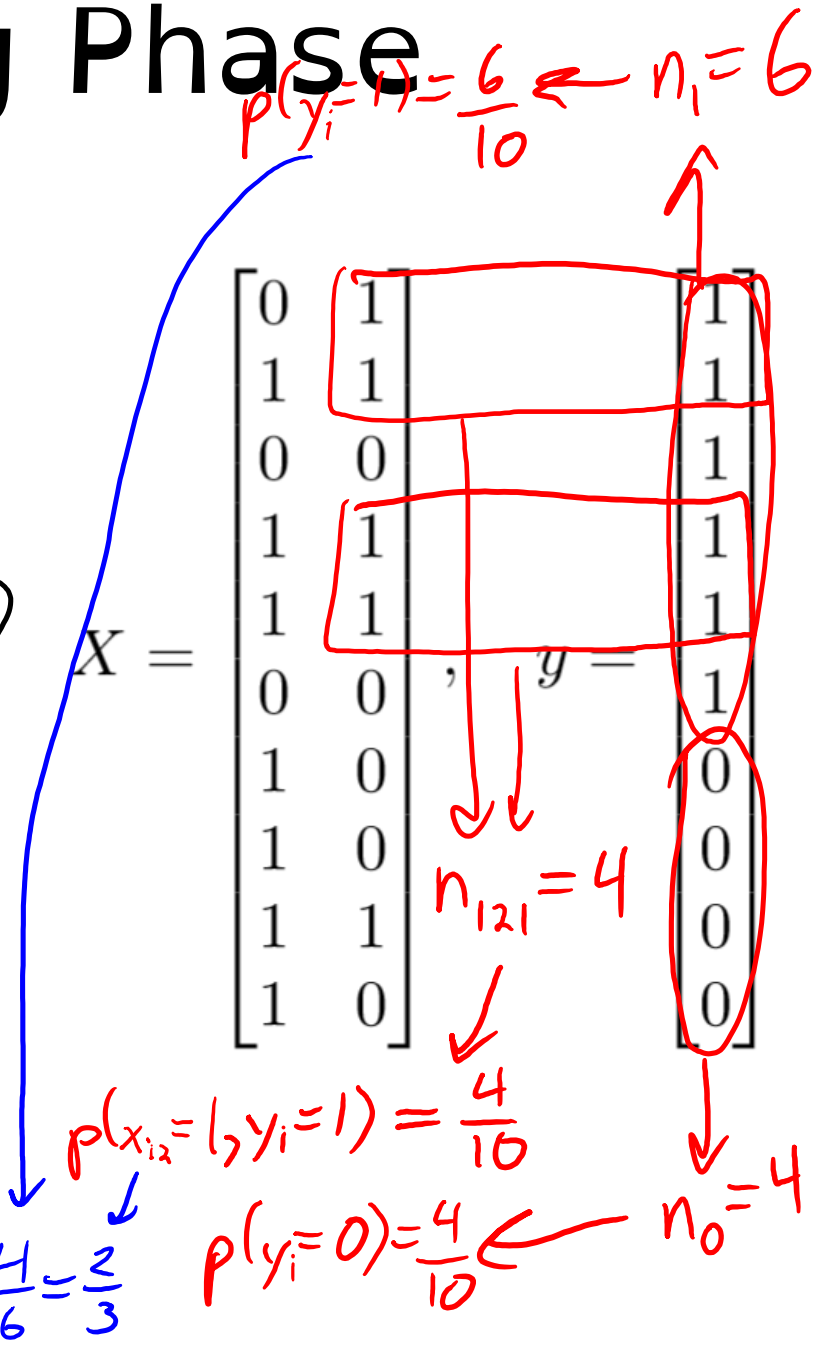


Naïve Bayes Training Phase

- **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 $\longrightarrow$

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

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

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

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 $p(\tilde{x}_i)$)

“Proportional to” for Probabilities

- When we say “$p(y) \propto \exp(-y^2)$” for a function ‘p’, we mean:

$$p(y) = \beta \exp(-y^2) \quad \text{for some constant } \beta.$$

- However, if ‘p’ is a probability then it must sum to 1.
  - If $y \in \{1,2,3,4\}$ then $p(1)+p(2)+p(3)+p(4)=1$
- Using this fact, we can find β:

$$\beta\exp(-1^2)+\beta\exp(-2^2)+\beta\exp(-3^2)+\beta\exp(-4^2)=1$$
$$\Leftrightarrow \beta\left[\exp(-1^2)+\exp(-2^2)+\exp(-3^2)+\exp(-4^2)\right]=1$$
$$\Leftrightarrow \beta = \frac{1}{\exp(-1^2)+\exp(-2^2)+\exp(-3^2)+\exp(-4^2)}$$

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 $\log\left(\prod_i a_i\right) = \sum_i \log(a_i)$

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

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.

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.

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".

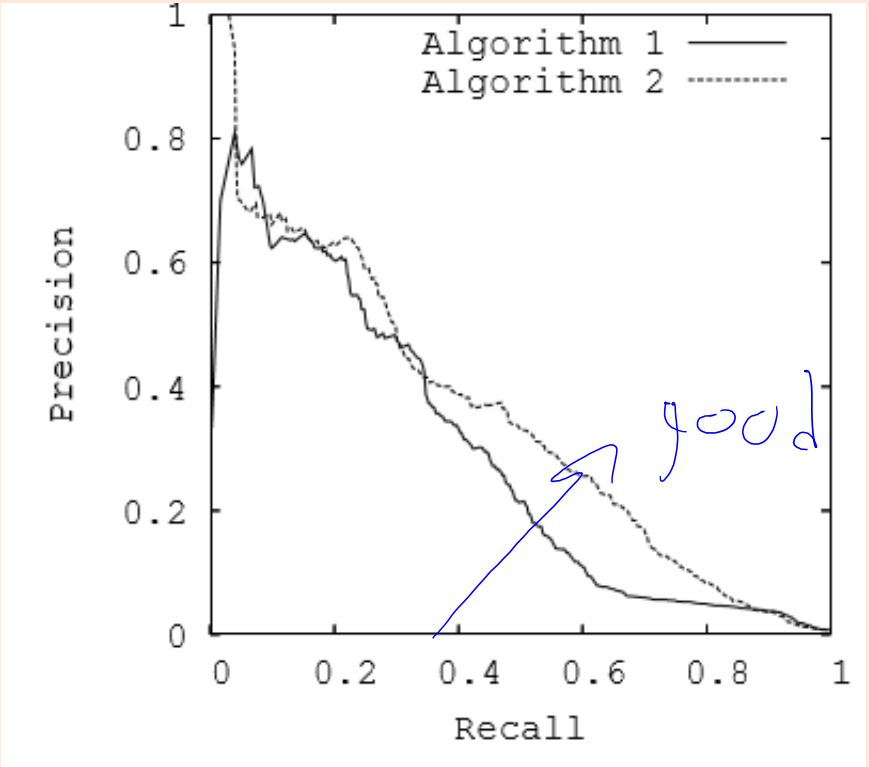
Other Performance Measures

- 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.

Precision-Recall Curve

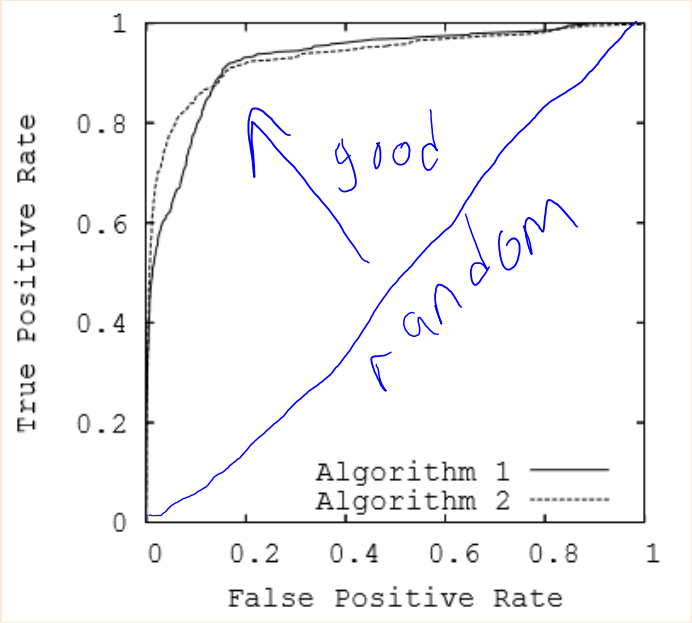
- Consider the rule $p(y_i = \text{'spam'} \mid x_i) > t$, for threshold 't'.
- Precision-recall (PR) curve plots precision vs. recall as 't' varies.



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

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).
    - Reflects performance for different possible thresholds on the probability.

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

More on Unbalanced Classes

- With unbalanced classes, there are many alternatives to accuracy as a measure of performance:
  - Two common ones are the Jaccard coefficient and the F-score.

- Some machine learning models don't work well with unbalanced data. Some common heuristics to improve performance are:
  - Under-sample the majority class (only take 5% of the spam messages).
    - https://www.jair.org/media/953/live-953-2037-jair.pdf
  - Re-weight the examples in the accuracy measure (multiply training error of getting non-spam messages wrong by 10).
  - Some notes on this issue are here.

Decision Theory Discussion

- In other applications, the costs could be different.
  - In cancer screening, maybe false positives are ok, but don't want to have false negatives.

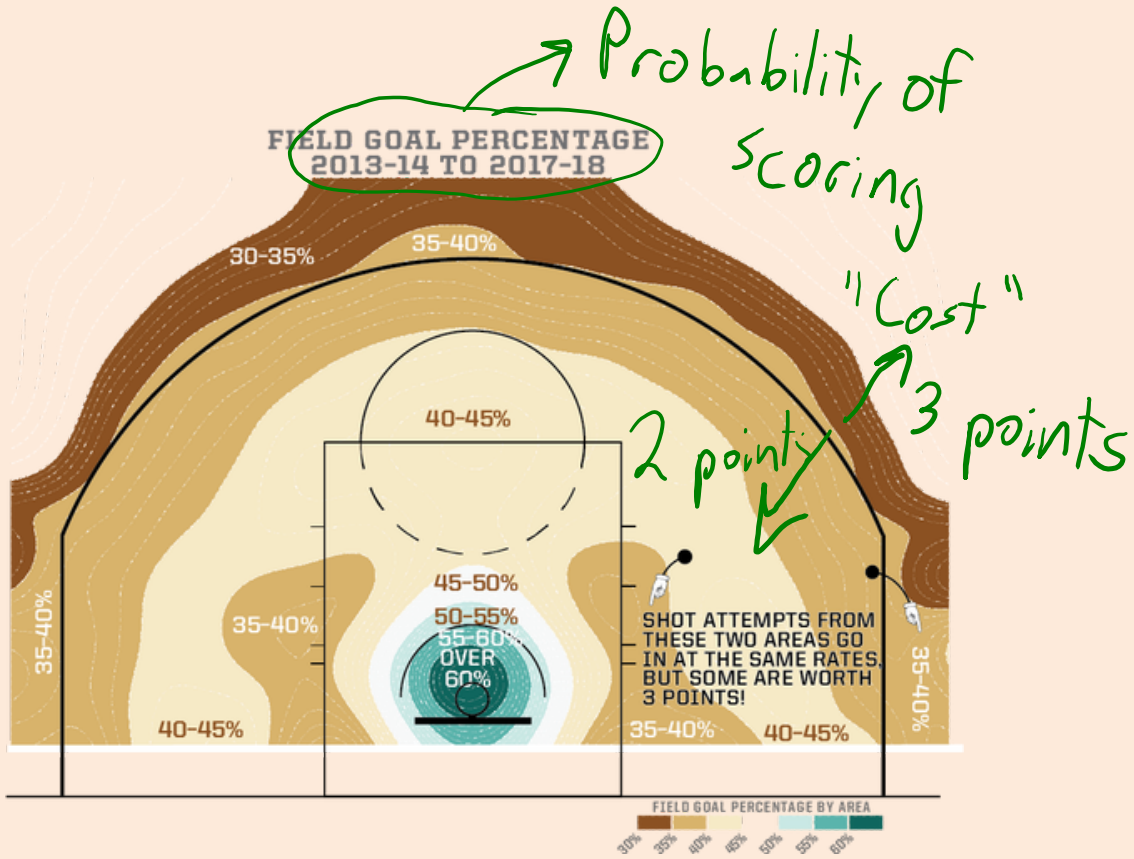
- Decision theory and "darts":
  - http://www.datagenetics.com/blog/january12012/index.html

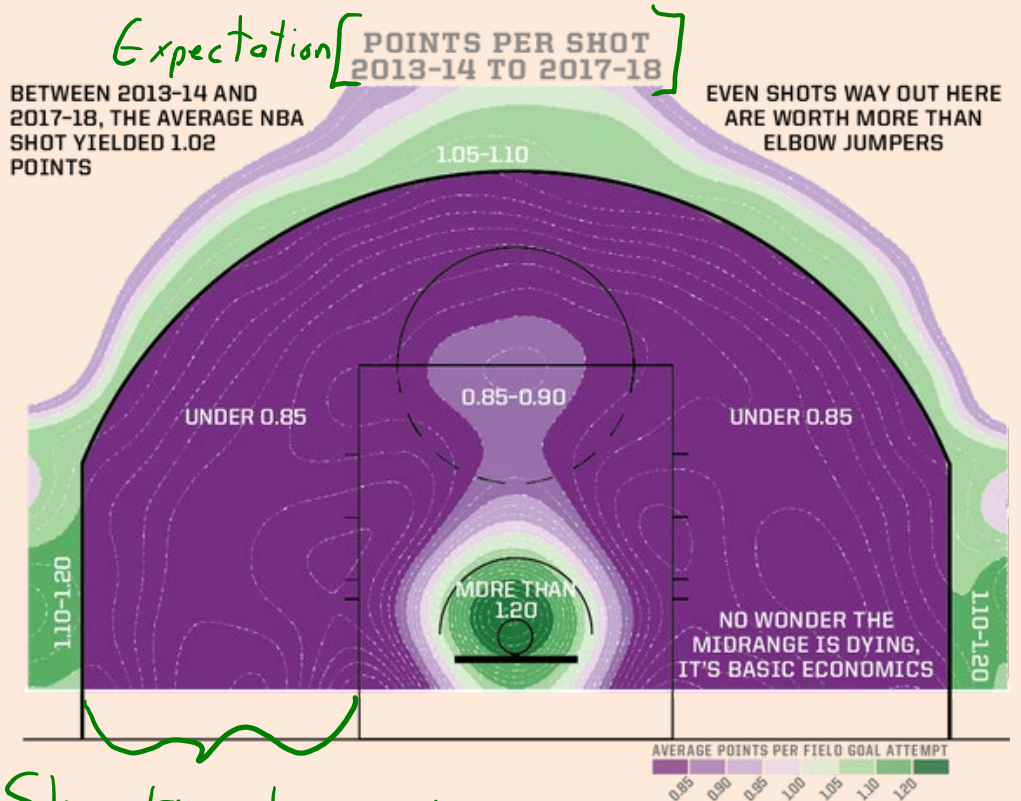
- Decision theory and video poker:
  - http://datagenetics.com/blog/july32019/index.html

- Decision theory can help with "unbalanced" class labels:
  - If 99% of e-mails are spam, you get 99% accuracy by always predicting "spam".
  - Decision theory approach avoids this.
  - See also precision/recall curves and ROC curves in the bonus material.

Decision Theory and Basketball

- "How Mapping Shots In The NBA Changed It Forever"





Shooting here is a bad decision

https://fivethirtyeight.com/features/how-mapping-shots-in-the-nba-changed-it-forever/

More on Weirdness of High Dimensions

- In high dimensions:
  - Distances become less meaningful:
    - All vectors may have similar distances.
  - Emergence of "hubs" (even with random data):
    - Some datapoints are neighbours to many more points than average.
  - Visualizing high dimensions and sphere-packing

Vectorized Distance Calculation

- To classify 't' test examples based on KNN, cost is O(ndt).
  - Need to compare 'n' training examples to 't' test examples, and computing a distance between two examples costs O(d).

- You can do this slightly faster using fast matrix multiplication:
  - Let D be a matrix such that D_{ij} contains:

$$||x_i - y_j||^2 = ||x_i||^2 - 2x_i^T x_j + ||x_j||^2$$

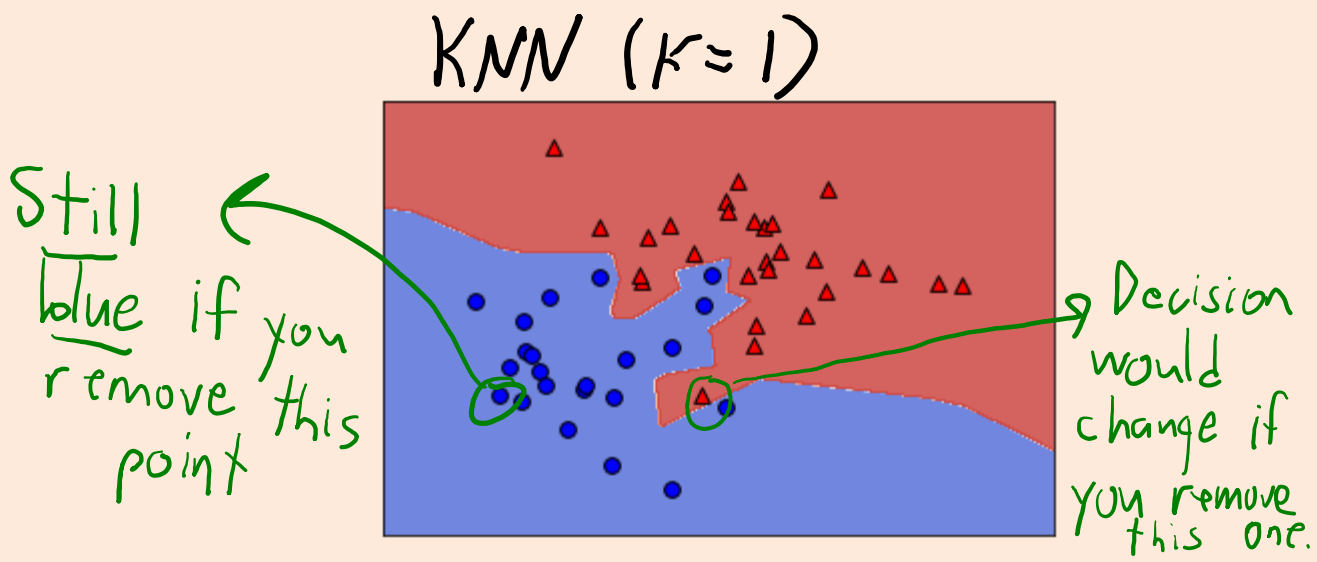
    where 'i' is a training example and 'j' is a test example.
  - We can compute D in Julia using:

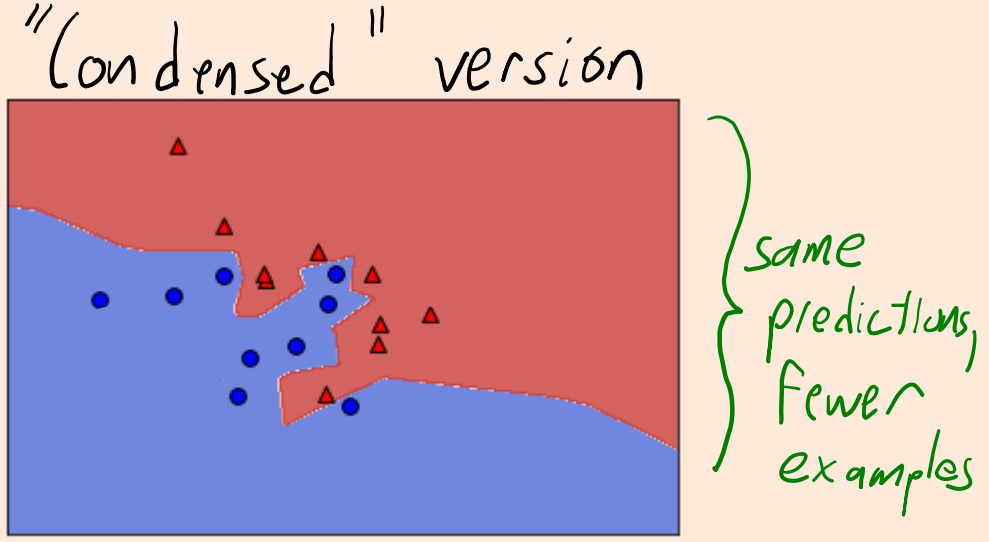
  - And you get an extra boost because Julia uses multiple cores.

```
X1.^2*ones(d,t) .+ ones(n,d)*(X2').^2 .- 2X1*X2'
```

Condensed Nearest Neighbours

- Disadvantage of KNN is slow prediction time (depending on 'n').
- Condensed nearest neighbours:
  - Identify a set of 'm' "prototype" training examples.
  - Make predictions by using these "prototypes" as the training data.
- Reduces runtime from O(nd) down to O(md).





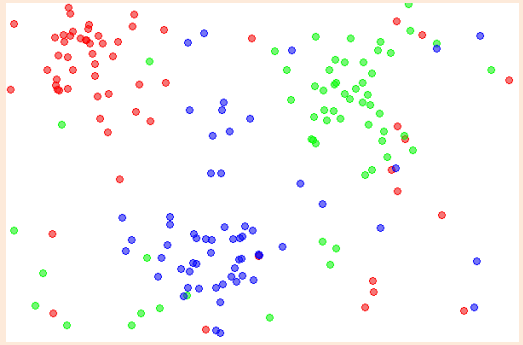
Condensed Nearest Neighbours

- Classic condensed nearest neighbours:
  - Start with no examples among prototypes.
  - Loop through the non-prototype examples ‘i’ in some order:
    - Classify x_i based on the current prototypes.
    - If prediction is not the true y_i, add it to the prototypes.
  - Repeat the above loop until all examples are classified correctly.
- Some variants first remove points from the original data, if a full-data KNN classifier classifies them incorrectly (“outliers”).

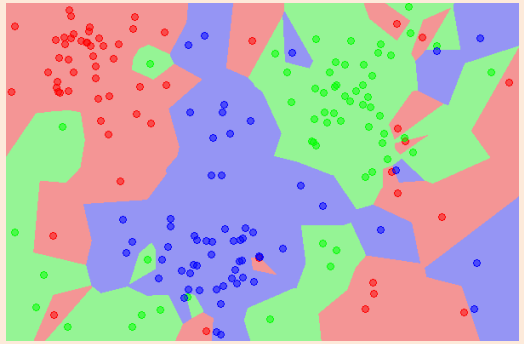
Condensed Nearest Neighbours

- Classic condensed nearest neighbours:

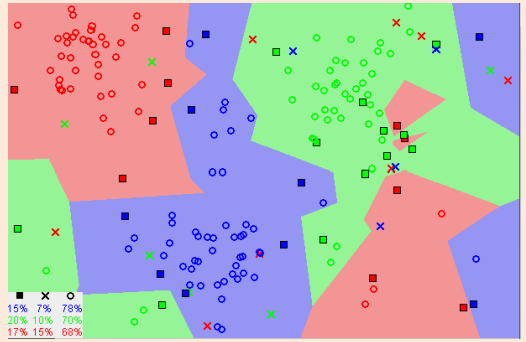
Data:



1NN



CNN:
□: prototype
○: removed
×: outlier
(using 3NN)



- Recent work shows that finding optimal compression is NP-hard.
  - An approximation algorithm algorithm was published in 2018:
    - "Near optimal sample compression for nearest neighbors"

https://en.wikipedia.org/wiki/K-nearest_neighbors_algorithm

Refined Fundamental Trade-Off

- Let E_{best} be the irreducible error (lowest possible error for *any* model).
  - For example, irreducible error for predicting coin flips is 0.5.
- Some learning theory results use E_{best} to further decompose E_{test}:

$$E_{test} = \underbrace{(E_{test} - E_{train})}_{E_{approx}} + \underbrace{(E_{train} - E_{best})}_{E_{model}} + \underbrace{E_{best}}_{\text{"noise"}}$$

- E_{approx} measures *how sensitive we are to training data.*
- E_{model} measures *if our model is complicated enough to fit data.*
- E_{best} measures how low can **any** model make test error.
  - E_{best} does not depend on what model you choose.

Consistency and Universal Consistency

- A model is consistent for a particular learning problem if:
  - E_{test} converges to E_{best} as 'n' goes to infinity, for that particular problem.
- A model is universally consistent for a class of learning problems if:
  - E_{test} converges to E_{best} as 'n' goes to infinity, for all problems in the class.
- Class of learning problems will usually be "all problems satisfying":
  - A continuity assumption on the labels y^i as a function of x^i.
    - E.g., if x^i is close to x^j then they are likely to receive the same label.
  - A boundedness assumption of the set of x^i.

Consistency of KNN (Discrete/Deterministic Case)

- Let's show universal consistency of KNN in a simplified setting.
  - The x^i and y^i are binary, and y^i being a deterministic function of x^i.
    - Deterministic y^i implies that E_{best} is 0.

- Consider KNN with k=1:
  - After we observe an x_i, KNN makes right test prediction for that vector.
  - As 'n' goes to ∞, each feature vectors with non-zero probability is observed.
  - We have $E_{test} = 0$ once we've seen all feature vectors with non-zero probability.

- Notes:
  - "No free lunch" isn't relevant as 'n' goes to ∞: we eventually see everything.
    - But there are 2^d possible feature vectors, so might need a huge number of training examples.
  - It's more complicated if labels aren't deterministic and features are continuous.

Consistency of Non-Parametric Models

- Universal consistency can be been shown for many models we'll cover:
  - Linear models with polynomial basis.
  - Linear models with Gaussian RBFs.
  - Neural networks with one hidden layer and standard activations.
    - Sigmoid, tanh, ReLU, etc.

- But it's always the non-parametric versions that are consistent:
  - Where size of model is a function of 'n'.
  - Examples:
    - KNN needs to store all 'n' training examples.
    - Degree of polynomial must grow with 'n' (not true for fixed polynomial).
    - Number of hidden units must grow with 'n' (not true for fixed neural network).