CPSC 340: Machine Learning and Data Mining

Non-Parametric Models

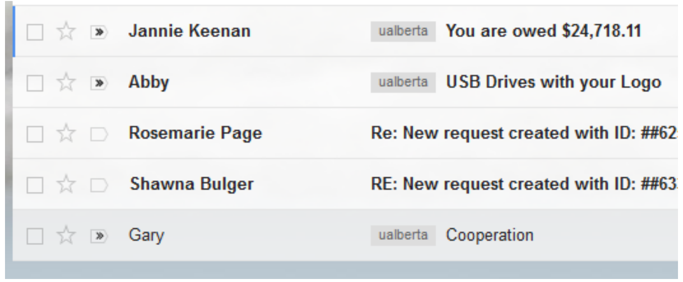
Fall 2022

Admin

- Welcome to the course!
  - If you have remaining forms, bring them to me after class and good luck.
- Assignment 1:
  - 1 late day to hand in tonight, 2 for Friday.
- Assignment 2 is out.
  - Due Friday of next week. It is long so start early.

Last Time: 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.
  - (x_{ij} = 1) if word/phrase 'j' is in e-mail 'i', (x_{ij} = 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
…

Last Time: Naïve Bayes

- We considered spam filtering methods based on naïve Bayes:

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

- Makes conditional independence assumption to make learning practical:

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

- Predict “spam” if $p(y_i = \text{“spam”} \mid x_i) > p(y_i = \text{“not spam”} \mid x_i)$.
  - We don’t need $p(x_i)$ to test this.

Naïve Bayes

- Naïve Bayes formally:

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

(first use Bayes rule)

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

("denominator doesn't matter") → same for all y_i values

$$\approx \prod_{j=1}^{d} \underbrace{\left[p(x_{ij} | y_i)\right] p(y_i)}_{\text{Only needs easy probabilities.}}$$

(conditional independence assumption)

If $f(y) \propto g(y)$, it means $f(y) = \kappa g(y)$ for all 'y' for some constant 'κ'

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

Laplace Smoothing

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

$$\frac{\text{\# spam messages with lactase}}{\text{\# spam messages}}$$

  – 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

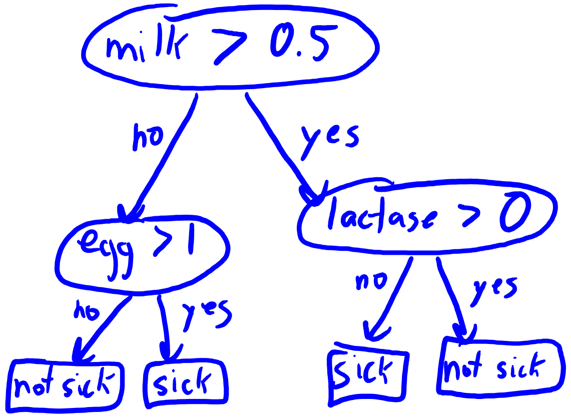
- Laplace smoothing: $\frac{(\#\text{spam messages with lactase})+1}{(\#\text{spam messages})+2}$
  - Typically you do this for all features.
    - Helps against overfitting by biasing towards the uniform distribution.
- A common variation is to use a real number β rather than 1.
  - Add 'βk' to denominator if feature has 'k' possible values (so it sums to 1).

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

This is a "maximum a posteriori" (MAP) estimate of the probabiliy. We'll discuss MAP and how to derive this formula later.

Decision Trees vs. Naïve Bayes

- Decision trees:



1. Sequence of rules based on 1 feature.
2. Training: 1 pass over data per depth.
3. Greedy splitting as approximation.
4. Testing: just look at features in rules.
5. New data: might need to change tree.
6. Accuracy: good if simple rules based on individual features work (“symptoms”).
7. Interpretability: easy to see how decisions are made.

- 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. Conditional independence assumption.
4. Testing: look at all features.
5. New data: just update counts.
6. Accuracy: good if features almost independent given label (bag of words).
7. Interpretability: can see how each feature influences decision.

Decision Theory

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

Predict / True	True 'spam'	True 'not spam'
Predict 'spam'	True Positive	False Positive
Predict 'not spam'	False Negative	True Negative

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

$$\underbrace{\mathbb{E}}_{\text{expectation of model with respect to } \tilde{y}_i}\Big[\underbrace{\text{cost}(\hat{y}_i, \tilde{y}_i)}_{\text{cost of predicting } \hat{y}_i \text{ if it's really } \tilde{y}_i}\Big]$$

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

Decision Theory Example

Predict / True	True ‘spam’	True ‘not spam’
Predict ‘spam’	0	100
Predict ‘not spam’	10	0

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

$$\mathbb{E}[\text{cost}(\hat{y}_i = \text{"spam"}, \tilde{y}_i)] = p(\tilde{y}_i = \text{"spam"} \mid \tilde{x}_i)\,\text{cost}(\hat{y}_i = \text{"spam"}, \tilde{y}_i = \text{"spam"})$$

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

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

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

- Even though “spam” is more likely, we should predict “not spam”.
  - With above costs, only classify as “spam” if $p(\tilde{y}_i = \text{“spam”} \mid \tilde{x}_i) \geq 0.91$.

Decision Theory Discussion

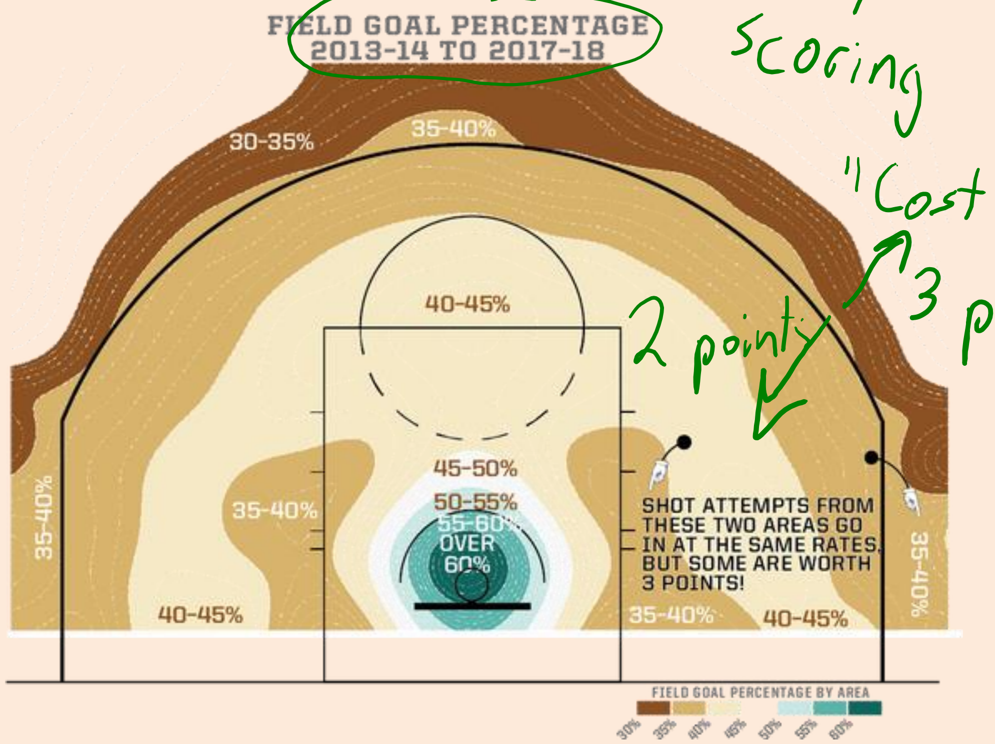
- In other applications, the costs could be different.
  - In cancer screening, maybe false positives are ok, but don't want to have false negatives.

- 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 and Basketball

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



FIELD GOAL PERCENTAGE 2013-14 TO 2017-18

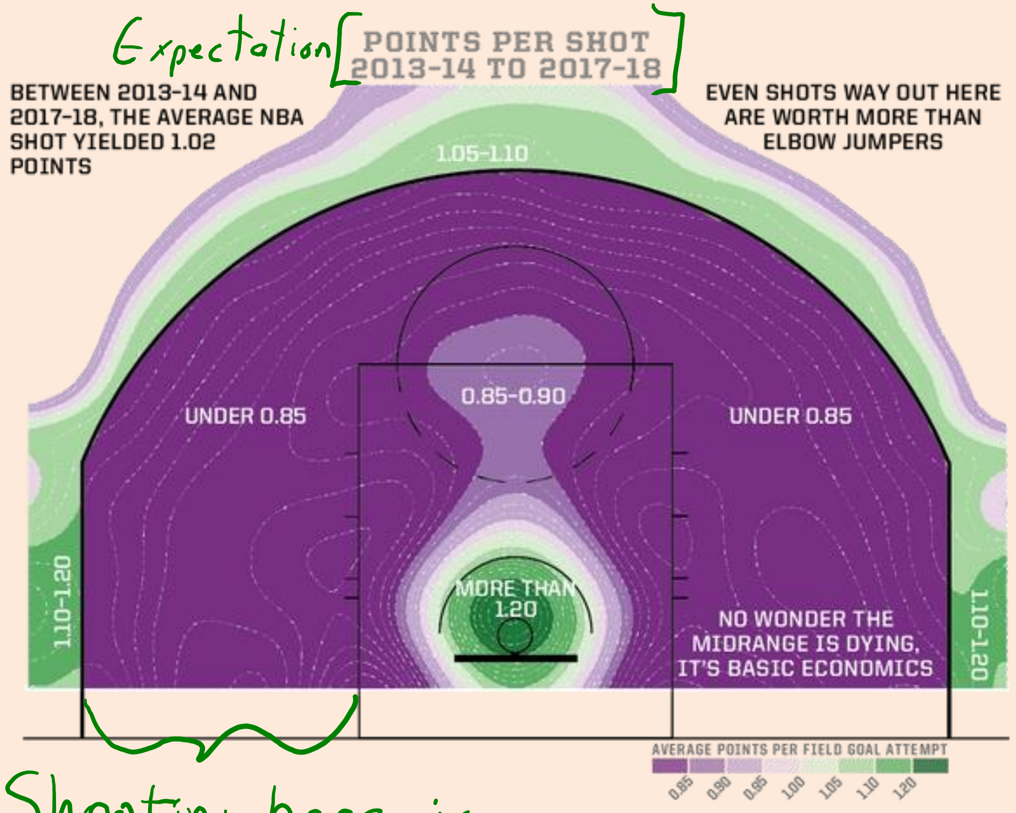
Probability of scoring

"Cost"

2 points

3 points

SHOT ATTEMPTS FROM THESE TWO AREAS GO IN AT THE SAME RATES, BUT SOME ARE WORTH 3 POINTS!



Expectation

POINTS PER SHOT 2013-14 TO 2017-18

BETWEEN 2013-14 AND 2017-18, THE AVERAGE NBA SHOT YIELDED 1.02 POINTS

EVEN SHOTS WAY OUT HERE ARE WORTH MORE THAN ELBOW JUMPERS

NO WONDER THE MIDRANGE IS DYING, IT'S BASIC ECONOMICS

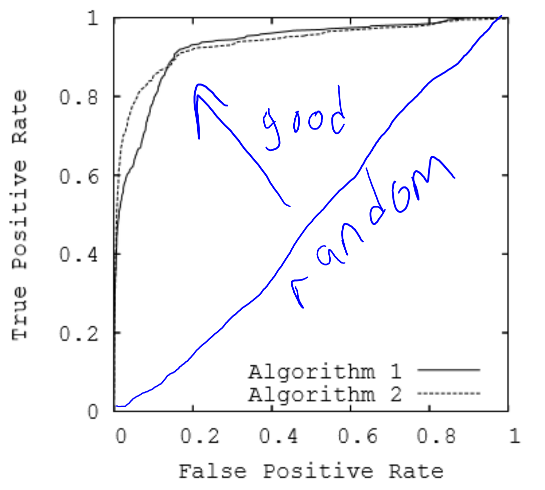
Shooting here is a bad decision

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

Unbalanced Class Labels

- A related idea is that of "unbalanced" class labels.
  - If 99% of the e-mails are spam,
    you can get 99% accuracy by always predicting spam.

- There are a variety of other performance measures available:
  - Weighted classification error.
  - Jaccard similarity.
  - Precision and recall.
  - False positive and false negative rate.
  - ROC curves.



- See the post-lecture bonus slides for additional details.

Next Topic: Non-Parametric Models

Digression: “Debugging by Frustration/TA”

- Here is one way to write a complicated program:
  1. Write the entire function at once.
  2. Try it out to “see if it works”.
  3. Spend hours fiddling with commands, to find magic working combination.
  4. Send code to the TA, asking “what is wrong?”

- If you are lucky, Step 2 works and you are done!

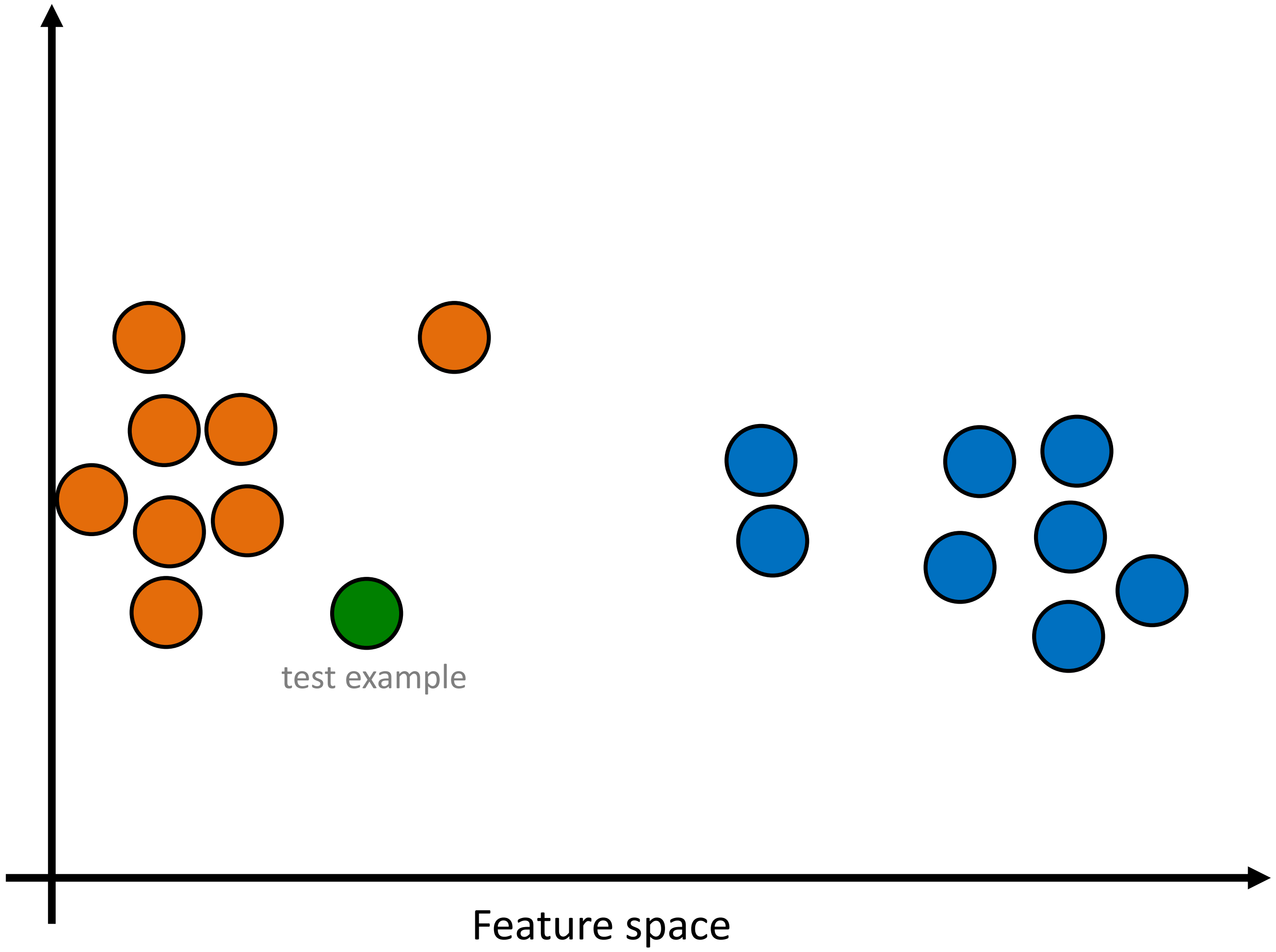
- If you are not lucky, takes way longer than principled coding methods.
  - This is also a great way to introduce bugs into your code.
  - And you will not be able to do Step 4 when you graduate.

Digression: Debugging 101

- What strategies could we use to debug an ML implementation?
  - Use "print" statements to see what is happening at each step of the code.
    - Or use a debugger.
  - Develop one or more simple "test cases", were you worked out the result by hand.
    - Maybe one of the functions you are using does not work the way you think it does.
  - Check if the "predict" functionality works correctly on its own.
    - Maybe the training works but the prediction does not.
  - Check if the "training" functionality works correctly on its own.
    - Maybe the prediction works but the training does not.
  - Try the implementation with only one training example or only one feature.
    - Maybe there is an indexing problem, or things are not being aggregated properly.
  - Try the implementation with only two features so you can visualize the decision surface.
    - May be able to see obvious problems.
  - Make a "brute force" implementation to compare to your "fast/clever" implementation.
    - Maybe you made a mistake when trying to be fast/clever.
- With these strategies, you should be able to diagnose locations of problems.

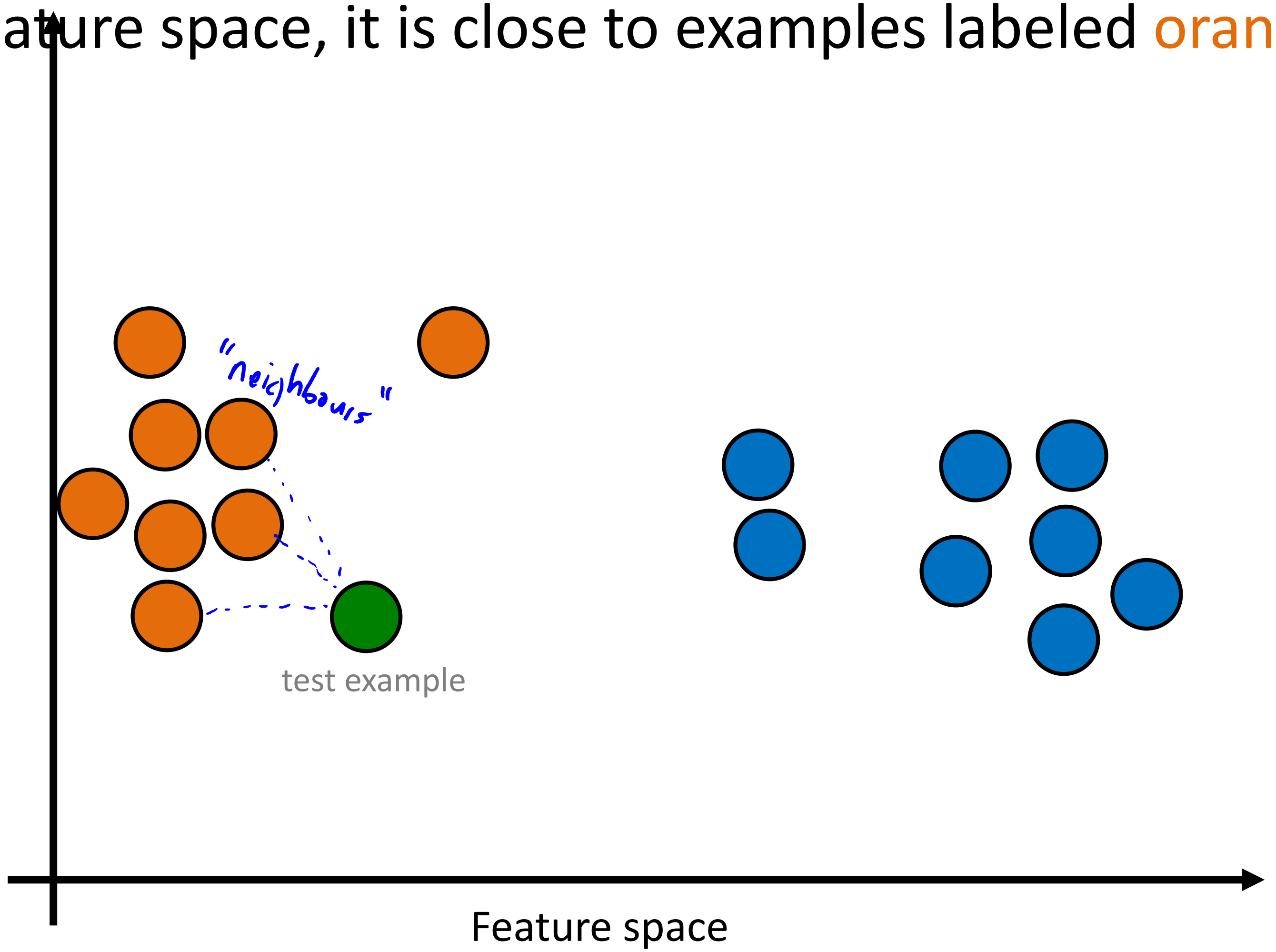
Back to ML: Geometric Motivation

- Do you think the green example should be orange or blue?



Back to ML: Geometric Motivation

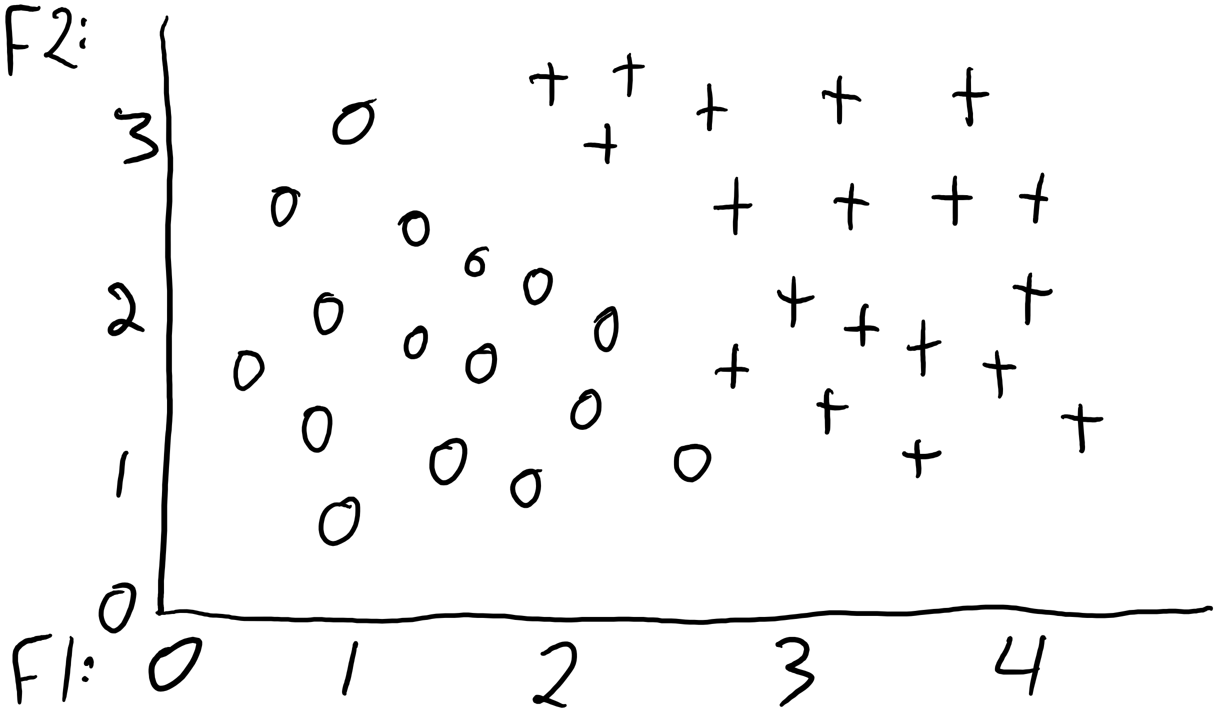
- Do you think the green example should be orange or blue?
  - In the feature space, it is close to examples labeled orange ("neighbours").



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.

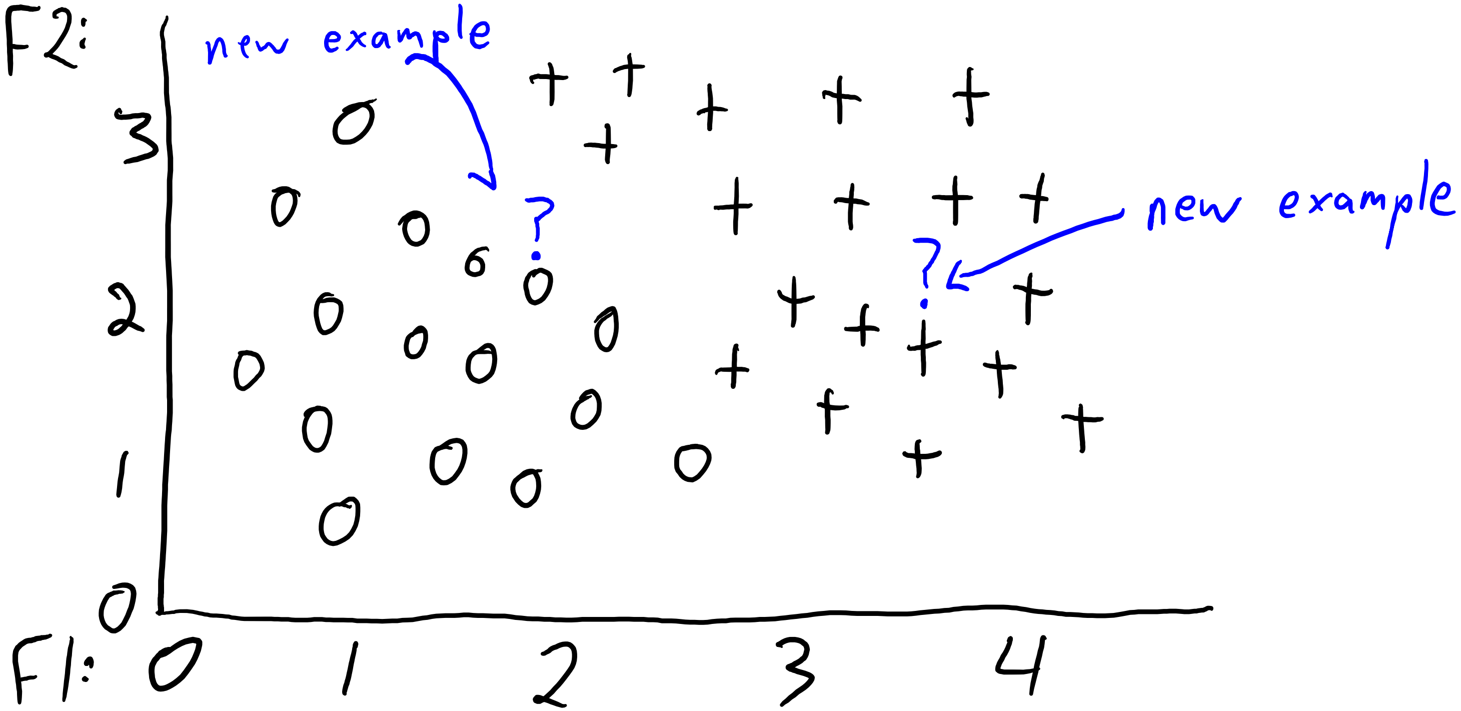
F1	F2		Label
1	3	→	O
2	3	→	+
3	2	→	+
2.5	1	→	O
3.5	1	→	+
…	…		…



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.

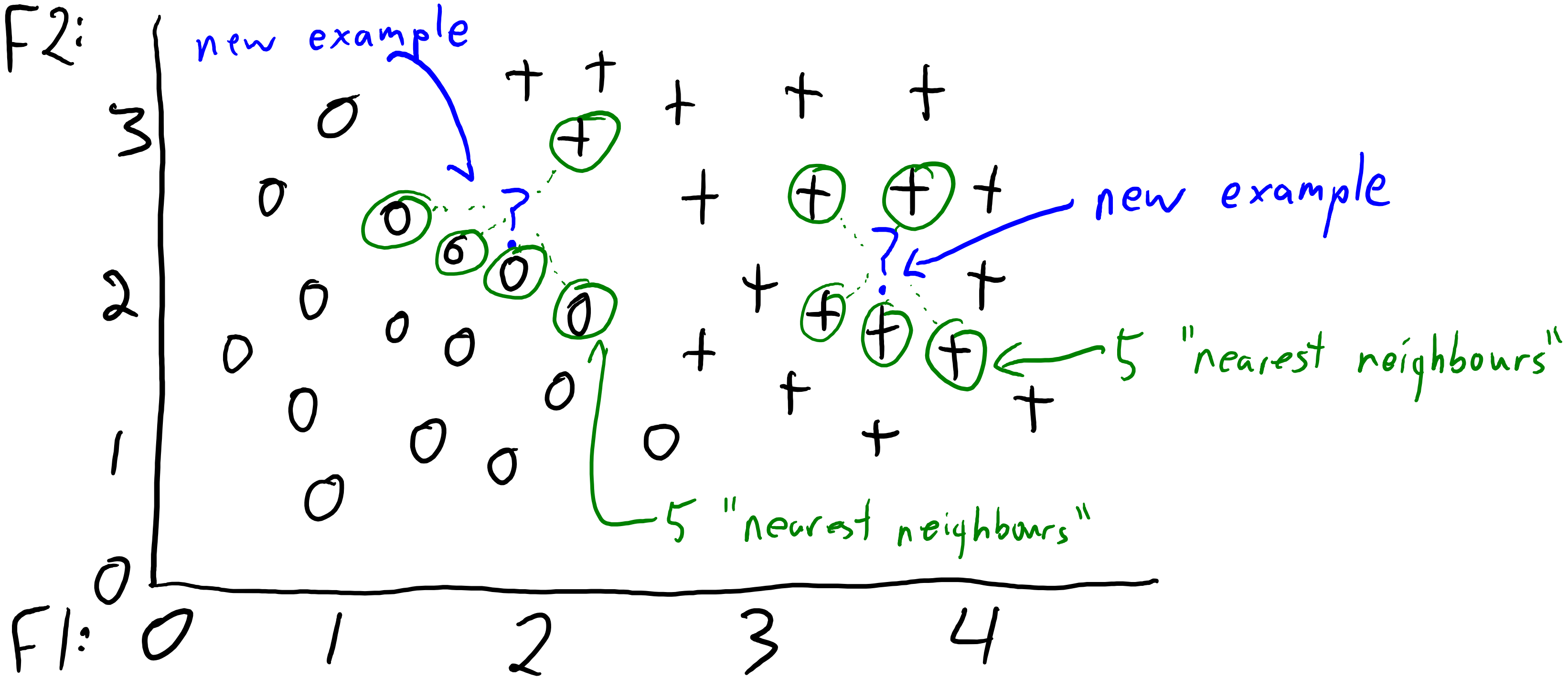
F1	F2	Label
1	3	O
2	3	+
3	2	+
2.5	1	O
3.5	1	+
…	…	…



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.

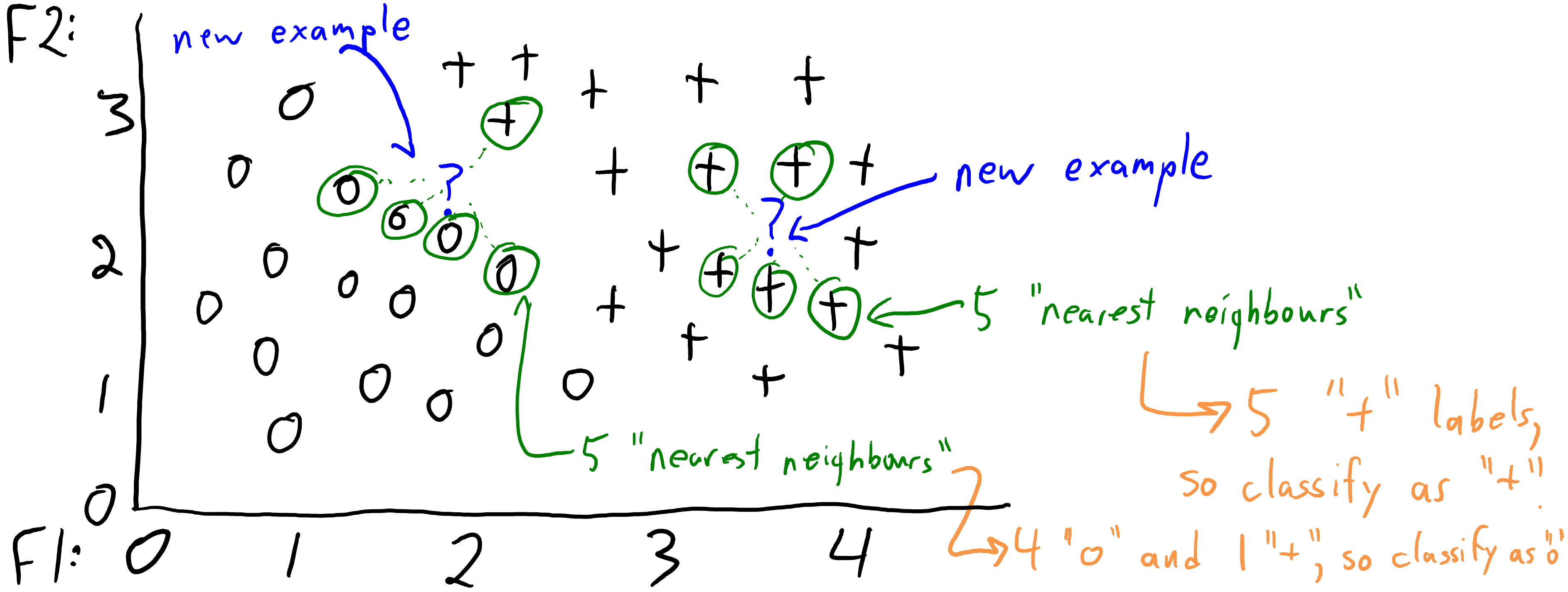
F1	F2	Label
1	3	O
2	3	+
3	2	+
2.5	1	O
3.5	1	+
…	…	…



K-Nearest Neighbours (KNN)

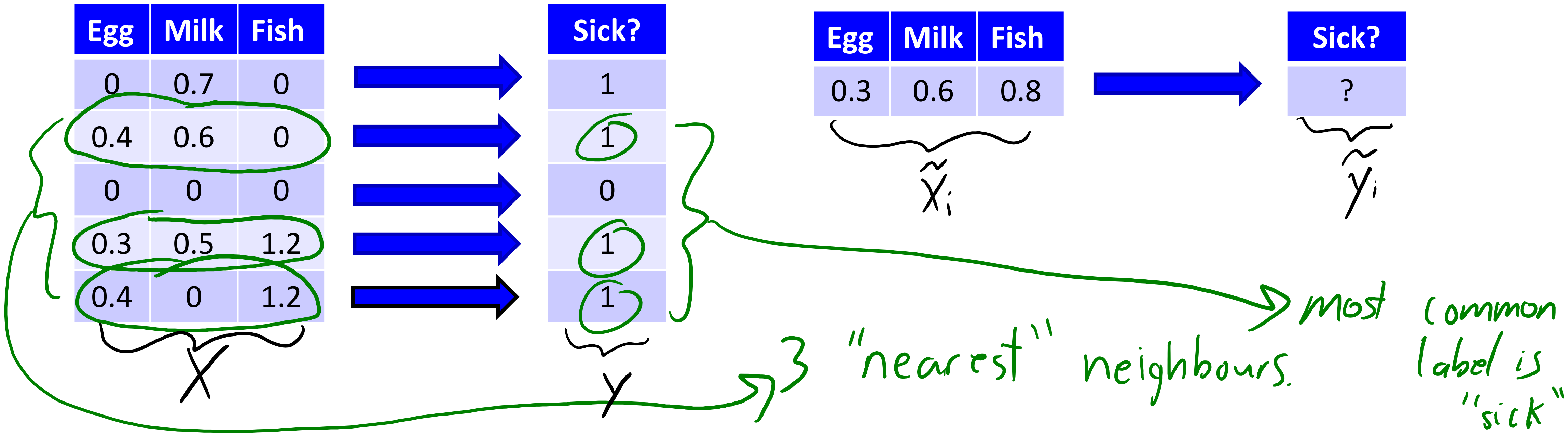
- An old/simple classifier: k-nearest neighbours (KNN).
- To classify an example $\tilde{x}_i$:
  1. Find the 'k' training examples x_i that are "nearest" to $\tilde{x}_i$.
  2. Classify using the most common label of "nearest" training examples.

F1	F2	Label
1	3	O
2	3	+
3	2	+
2.5	1	O
3.5	1	+
…	…	…



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.



Egg	Milk	Fish
0	0.7	0
0.4	0.6	0
0	0	0
0.3	0.5	1.2
0.4	0	1.2

Sick?
1
1
0
1
1

Egg	Milk	Fish
0.3	0.6	0.8

Sick?
?

3 "nearest" neighbours

most common label is "sick"

K-Nearest Neighbours (KNN)

- Assumption:
  - Examples with similar features are likely to have similar labels.

- Seems strong, but all good classifiers basically rely on this assumption.
  - If not true there may be nothing to learn and you are in “no free lunch” territory.
  - Methods just differ in how you define “similarity”.

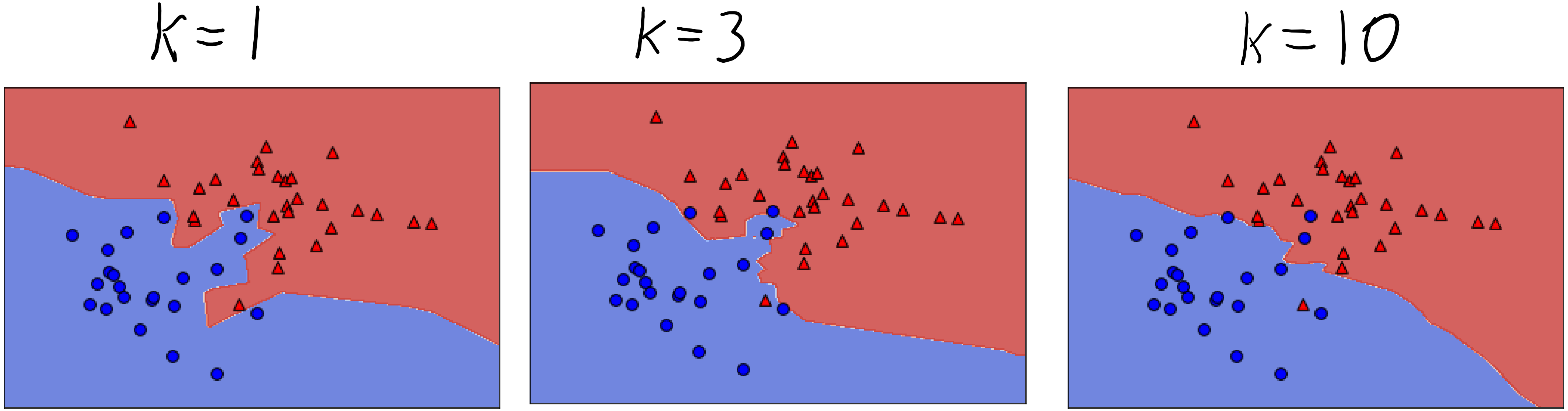
- Most common distance function is Euclidean distance:

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

  - x_i is features of training example ‘i’, and $\tilde{x}_{\tilde{\imath}}$ is features of test example ‘$\tilde{\imath}$’.
  - Costs O(d) to calculate for a pair of examples.

Effect of ‘k’ in KNN.

- With large ‘k’ (hyper-parameter), KNN model will be very simple.
  - With k=n, you just predict the mode of the labels.
  - Model gets more complicated as ‘k’ decreases (with k=1 it’s very sensitive).



k=1

k=3

k=10

- Effect of ‘k’ on fundamental trade-off:
  - As ‘k’ grows, training error tends to increase.
  - As ‘k’ grows, approximation error tends to decrease.

KNN Implementation

- 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 (for example, “condensed nearest neighbor”).

- 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.

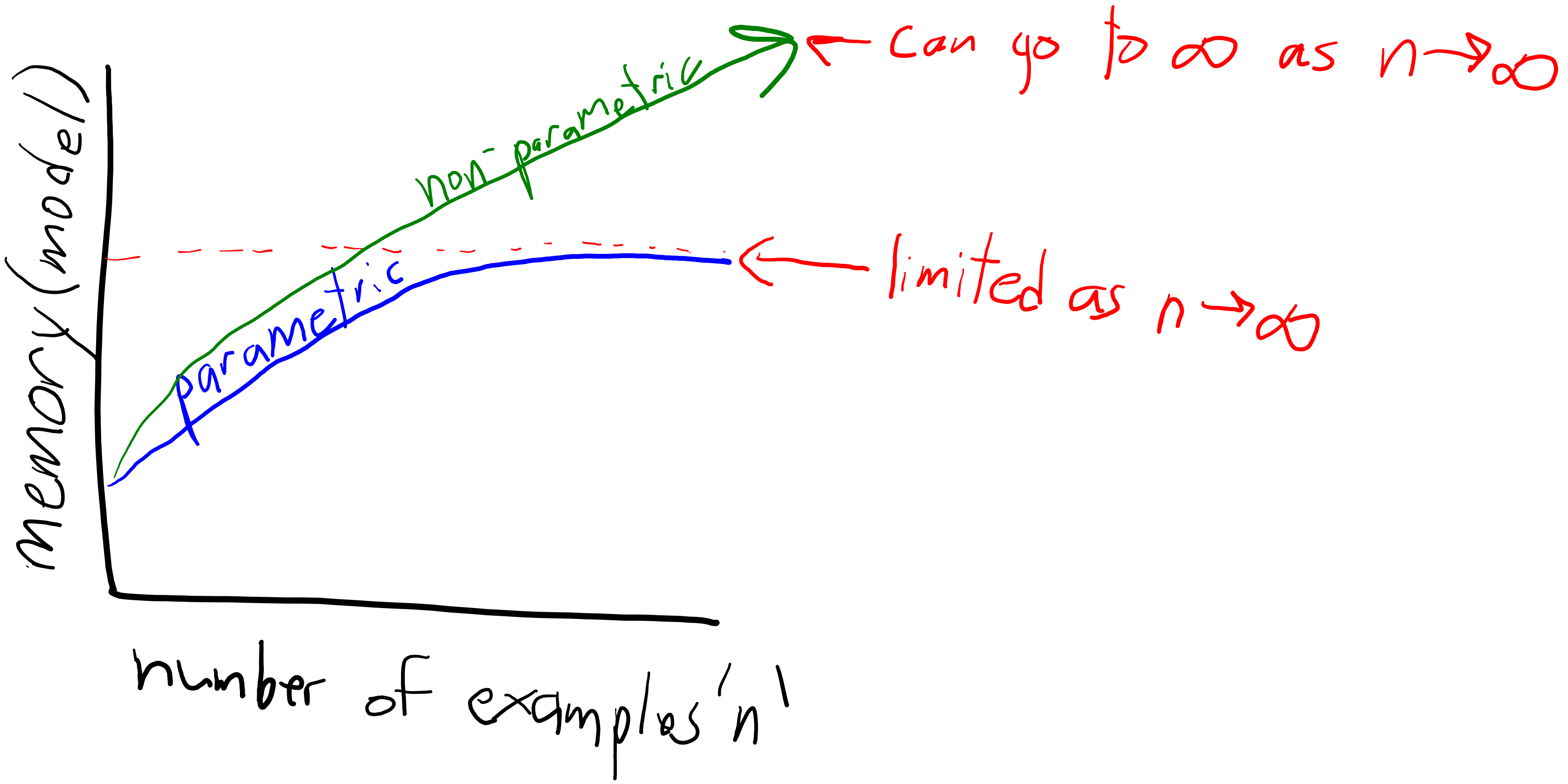
Parametric vs. Non-Parametric

- 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 does not 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*.

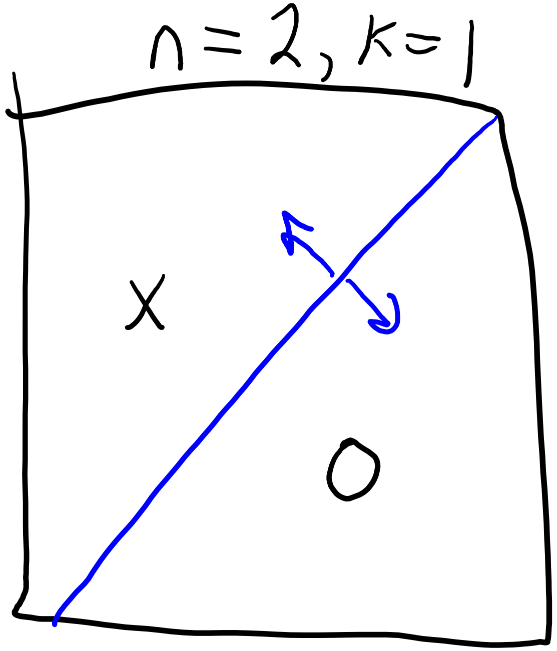
Parametric vs. Non-Parametric Models

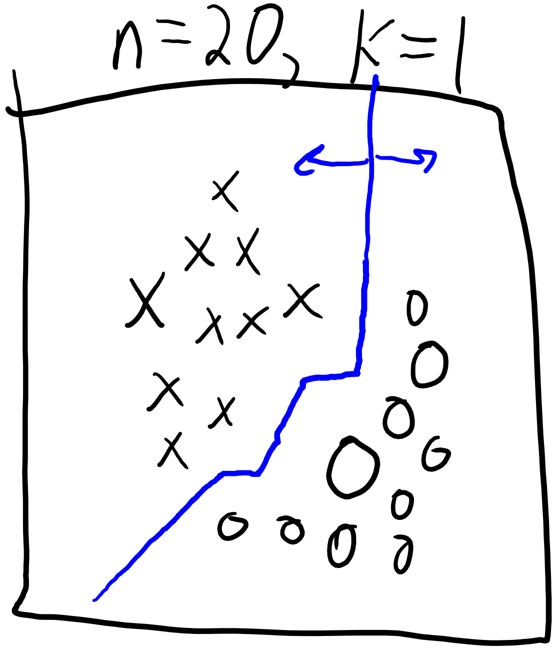
- Parametric models have bounded memory.
- Non-parametric models can have unbounded memory.



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.

Consistency of KNN (‘n’ going to ‘∞’)

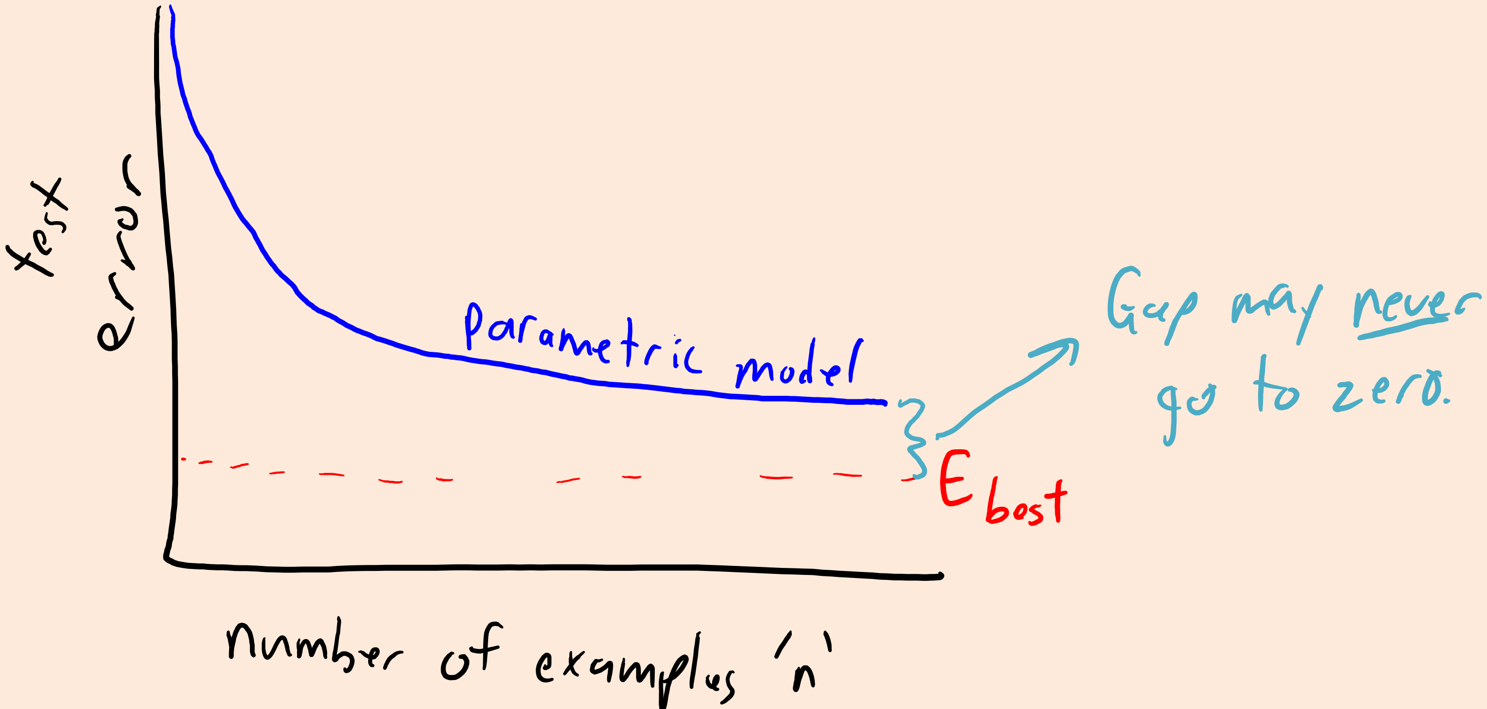
- KNN has appealing consistency properties:
  - As ‘n’ goes to ∞, KNN test error is less than twice best possible error.
    - For fixed ‘k’ and binary labels (under mild assumptions).

- 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”).
    - The “speed” at which universal consistency happens is exponential in the dimension ‘d’.

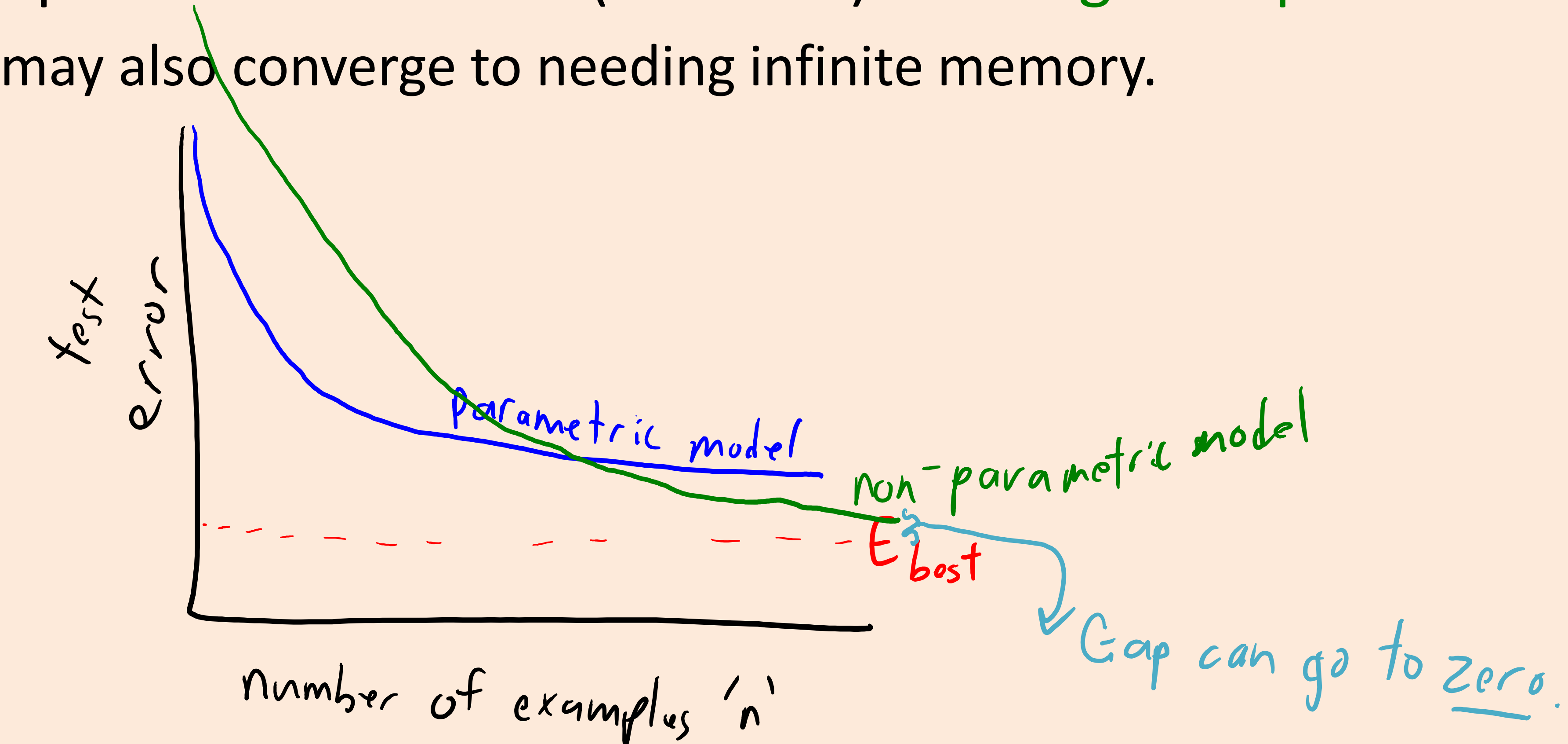
Parametric vs. Non-Parametric Models

- 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.
  - Though may also converge to needing infinite memory.



Summary

- Decision theory allows us to consider costs of predictions.
- Debugging 101: ideas to find bugs and write code with fewer bugs.
- 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.
- Non-parametric models grow with number of training examples.
  - Can have appealing "consistency" properties.

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

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 $(y_i = c)$.

$n_1 = 6$

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

$n_0 = 4$

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 $\frac{n_c}{n}$.

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

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

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

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

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

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

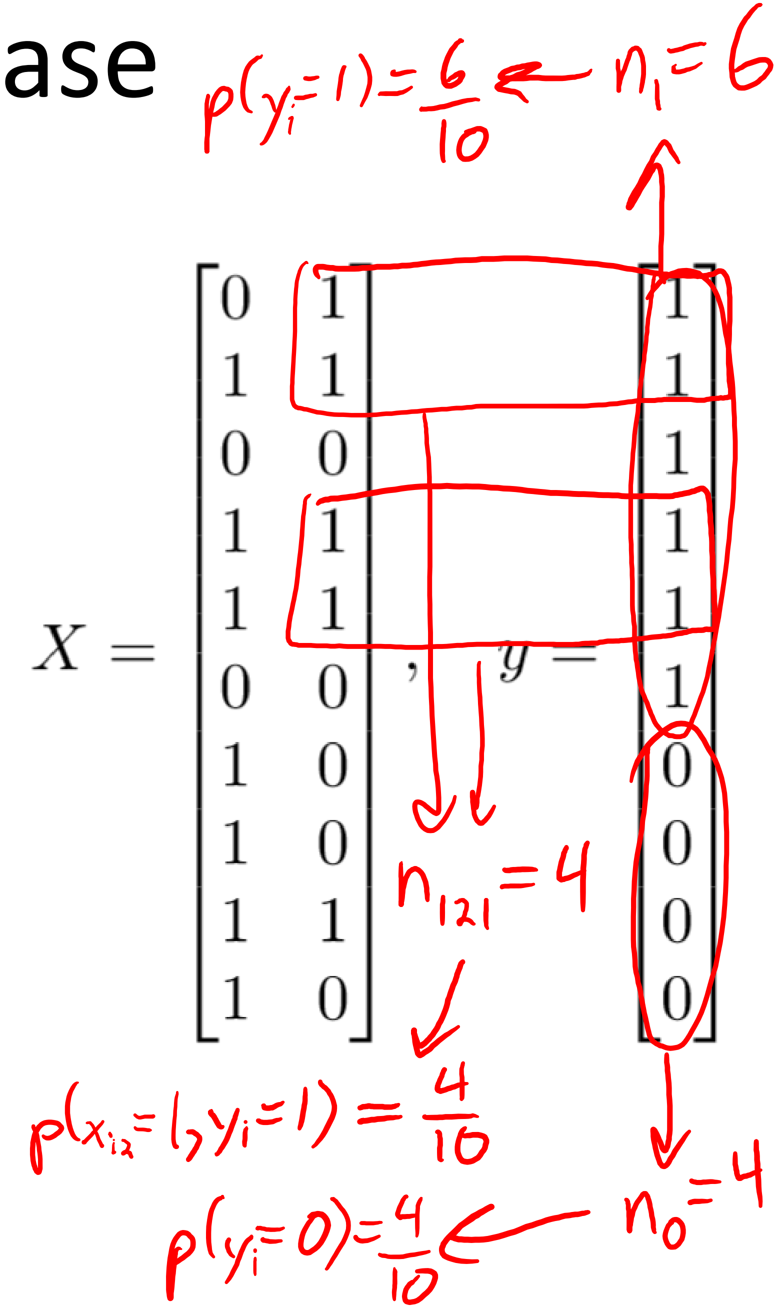
$n_{121}=4$

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

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 $(y_i = c, x_{ij} = k)$
4. Estimate $p(x_{ij} = k, y_i = c)$ as $\frac{n_{cjk}}{n}$.



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

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

$n_{121} = 4$

$p(x_{i2} = 1, y_i = 1) = \frac{4}{10}$

$n_0 = 4$

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

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 $(y_i = c, x_{ij} = k)$
4. Estimate $p(x_{ij} = k, y_i = c)$ as $\frac{n_{cjk}}{n}$
5. Use that $p(x_{ij} = k \mid y_i = c) = \frac{p(x_{ij} = k, y_i = c)}{p(y_i = c)}$

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

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

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

$$n_{121} = 4$$

$$p(x_{i2} = 1, y_i = 1) = \frac{4}{10}$$

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

$$p(x_{i2} = 1 \mid y_i = 1) = \frac{4}{6} = \frac{2}{3}$$

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 ⟶

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

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

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

Naïve Bayes Prediction Phase

- Prediction in a naïve Bayes model:

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

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

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

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

$$= (0.5) \quad (0.666\ldots) \quad (0.6) = 0.2$$

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

Naïve Bayes Prediction Phase

- Prediction in a naïve Bayes model:

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

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

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

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

$$= (0.5) \qquad (0.666\ldots) \qquad (0.6) = 0.2$$

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

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) ∝ exp(-y²)” for a function ‘p’, we mean:

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

- 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:

$$\beta\exp(-1^2)+\beta\exp(-2^2)+\beta\exp(-3^2)+\beta\exp(-4^2)=1$$

$$\Leftrightarrow \beta[\exp(-1^2)+\exp(-2^2)+\exp(-3^2)+\exp(-4^2)=1$$

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

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:

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

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

Less-Naïve Bayes

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

$$\begin{aligned}
p(y|x_1,x_2,\dots,x_d) &\propto p(y)p(x_1,x_2,\dots,x_d|y) \qquad \text{product rule applied } \underline{\text{repeatedly}} \\
&= p(y)p(x_1|y)p(x_2|x_1,y)p(x_3|x_2,x_1,y)\cdots p(x_d|x_1,x_2,\dots,x_{d-1},y) \\
&\approx p(y)p(x_1|y)p(x_2|y)p(x_3|y)\cdots p(x_d|y) \qquad \text{(naive Bayes assumption)}
\end{aligned}$$

- 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:

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

    - 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:

$$
\begin{aligned}
p(x_i) &= \sum_{c=1}^{K} p(x_i, y=c) && \text{(marginalization rule)}\\
&= \sum_{c=1}^{K} p(x_i \mid y=c)\, p(y=c) && \text{(product rule)}\\
&= \sum_{c=1}^{K} \underbrace{\left[\prod_{j=1}^{d} p(x_{ij} \mid y=c)\right] p(y=c)}_{\substack{\text{These are the quantities}\\ \text{we compute during training.}}} && \text{(naive Bayes assumption)}
\end{aligned}
$$

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".

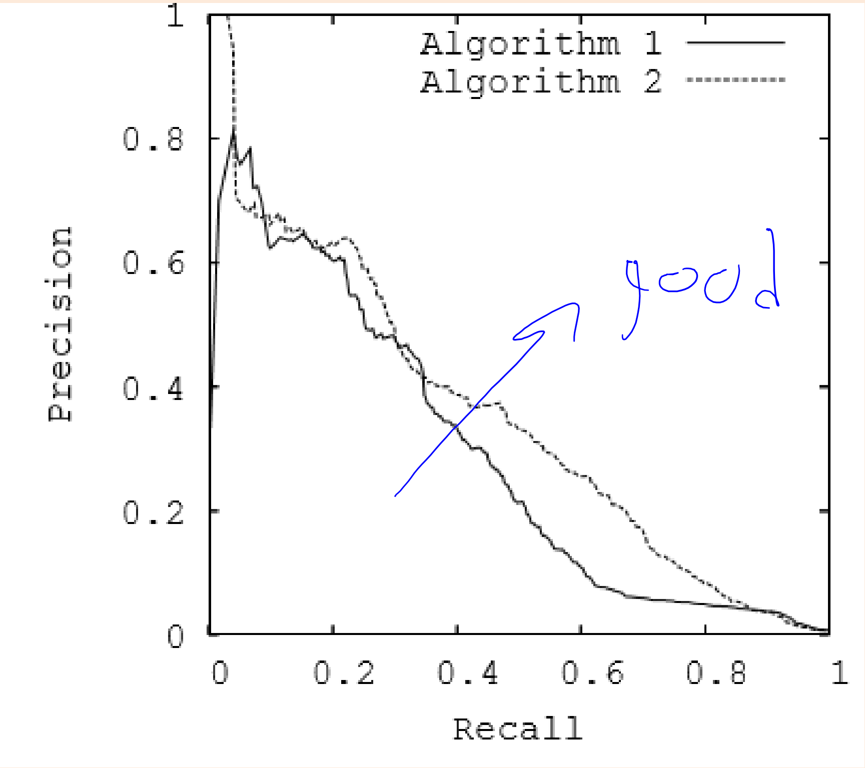
Other Performance Measures

- 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.

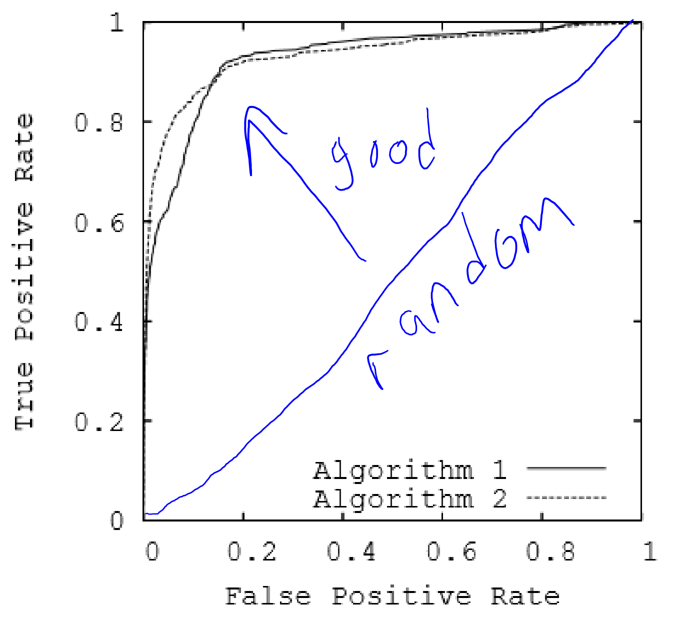


http://pages.cs.wisc.edu/~jdavis/davisgoadrichcamera2.pdf

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.

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

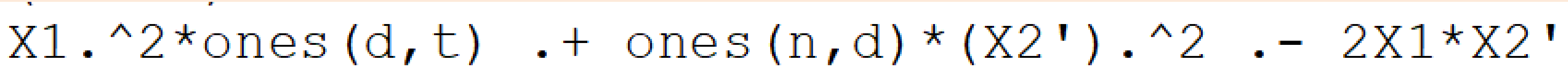
Vectorized Distance Calculation

- 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:

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

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



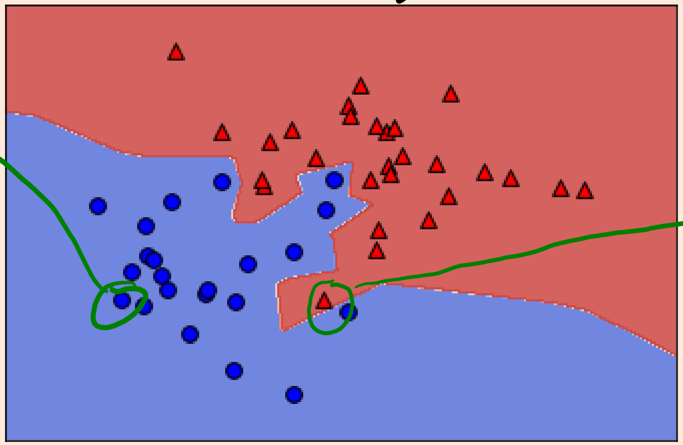
```
X1.^2*ones(d,t) .+ ones(n,d)*(X2').^2 .- 2X1*X2'
```

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

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).

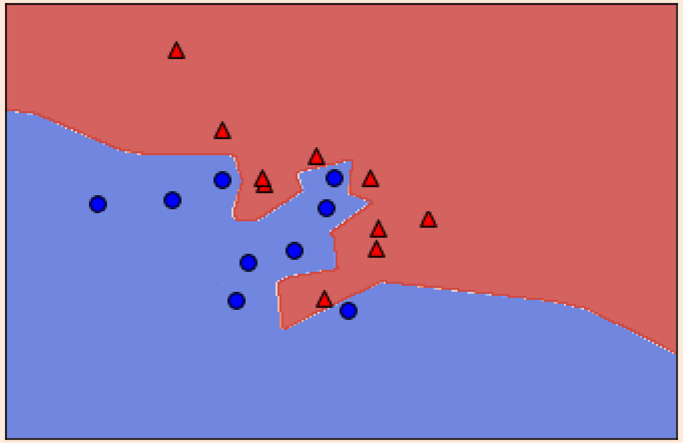
KNN (k=1)



Still blue if you remove this point

Decision would change if you remove this one.

"Condensed" version



same predictions, fewer examples

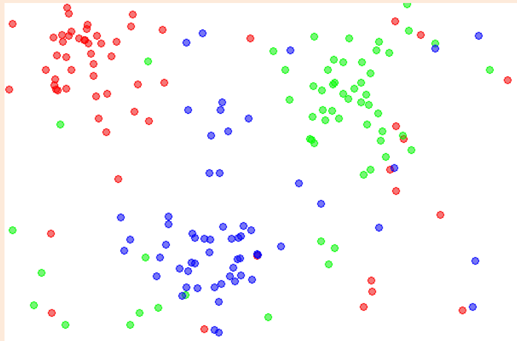
Condensed Nearest Neighbours

- 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’).

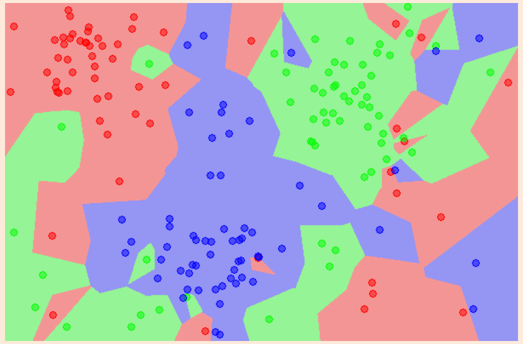
Condensed Nearest Neighbours

- Classic condensed nearest neighbours:

Data:



1NN



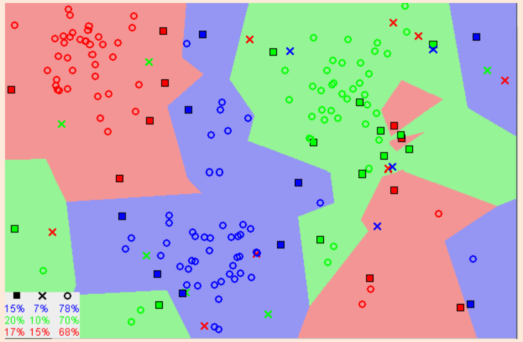
CNN:

□: 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}:

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

- 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).