

CPSC 340: Machine Learning and Data Mining

Non-Parametric Models

Fall 2019

Admin

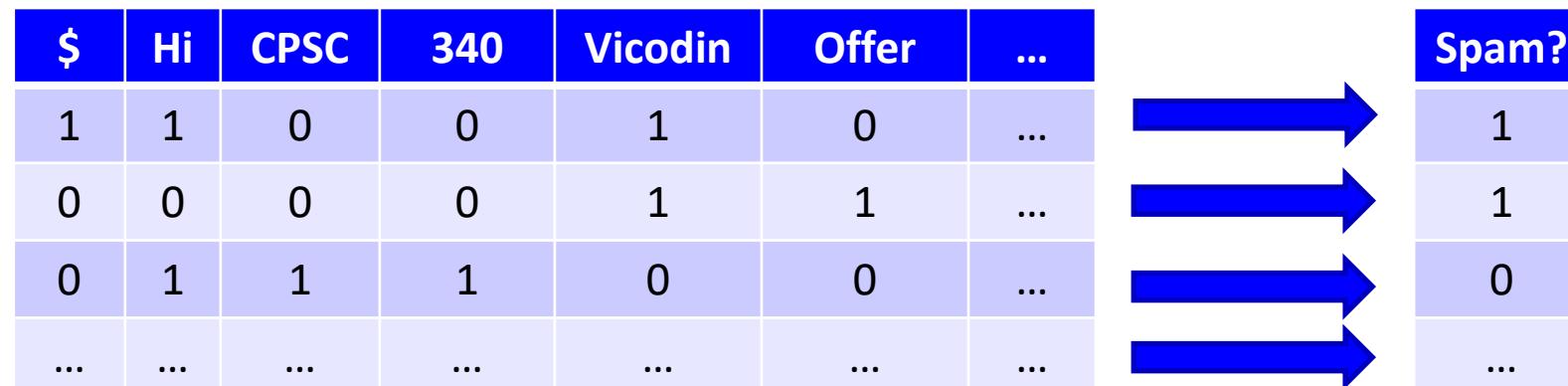
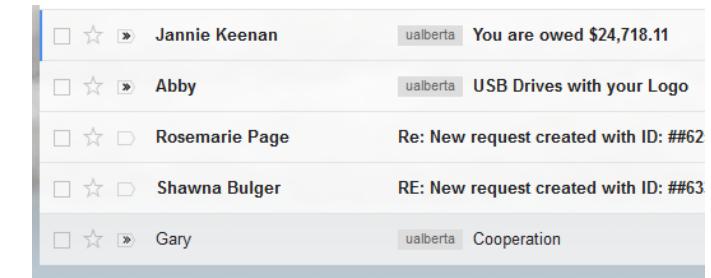
- Course webpage:
 - <https://www.cs.ubc.ca/~schmidtm/Courses/340-F19/>
- Assignment 1:
 - 1 late day to hand in tonight, 2 for Wednesday.
- Assignment 2 is out.
 - Due Friday of next week. It's long so start early.
- Add/drop deadline is tomorrow.
- Auditing/Exchange:
 - Bring your form to me after class.

Last Time: E-mail Spam Filtering

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

- We formulated as **supervised learning**:

- $(y_i = 1)$ if e-mail ‘ i ’ is spam, $(y_i = 0)$ if e-mail is not spam.
- $(x_{ij} = 1)$ if word/phrase ‘ j ’ is in e-mail ‘ i ’, $(x_{ij} = 0)$ if it is not.



Last Time: Naïve Bayes

- We considered spam filtering methods based on naïve Bayes:

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

- Makes conditional independence assumption to make learning practical:

$$p(\text{hello}=1, \text{vicodin}=0, \text{340}=1 | \text{spam}) \approx p(\text{hello}=1 | \text{spam}) p(\text{vicodin}=0 | \text{spam}) p(340=1 | \text{spam})$$

easy easy easy

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

Naïve Bayes

- Naïve Bayes formally:

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

$$\propto p(x_i | y_i) p(y_i) \quad ("denominator doesn't matter")$$

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

Only needs easy probabilities.

same for
all y_i
values

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

Laplace Smoothing

- Our estimate of $p(\text{'lactase'} = 1 | \text{'spam'})$ is:

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

- But there is a problem if you have no spam messages with lactase:
 - $p(\text{'lactase'} | \text{'spam'}) = 0$, so spam messages with lactase automatically get through.
then the final product formula will be 0
- 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.

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).

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

This is a “maximum a posteriori” (MAP) estimate of the probability. We’ll discuss MAP and how to derive this formula later.

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 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:

$$E \left[\underbrace{\text{cost}(\hat{y}_i, \tilde{y}_i)}_{\substack{\text{expectation of model} \\ \text{with respect to } \tilde{y}_i}} \right]$$

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.

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 $p(\tilde{y}_i = \text{"spam"} | \tilde{x}_i) = 0.6$, then:

$$\begin{aligned} \mathbb{E}[\text{cost}(\hat{y}_i = \text{"spam"}, \tilde{y}_i)] &= p(\tilde{y}_i = \text{"spam"} | \tilde{x}_i) \text{cost}(\hat{y}_i = \text{"spam"}, \tilde{y}_i = \text{"spam"}) \\ &\quad + p(\tilde{y}_i = \text{"not spam"} | \tilde{x}_i) \text{cost}(\hat{y}_i = \text{"spam"}, \tilde{y}_i = \text{"not spam"}) \\ &= (0.6)(0) + (0.4)(100) = 40 \end{aligned}$$

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

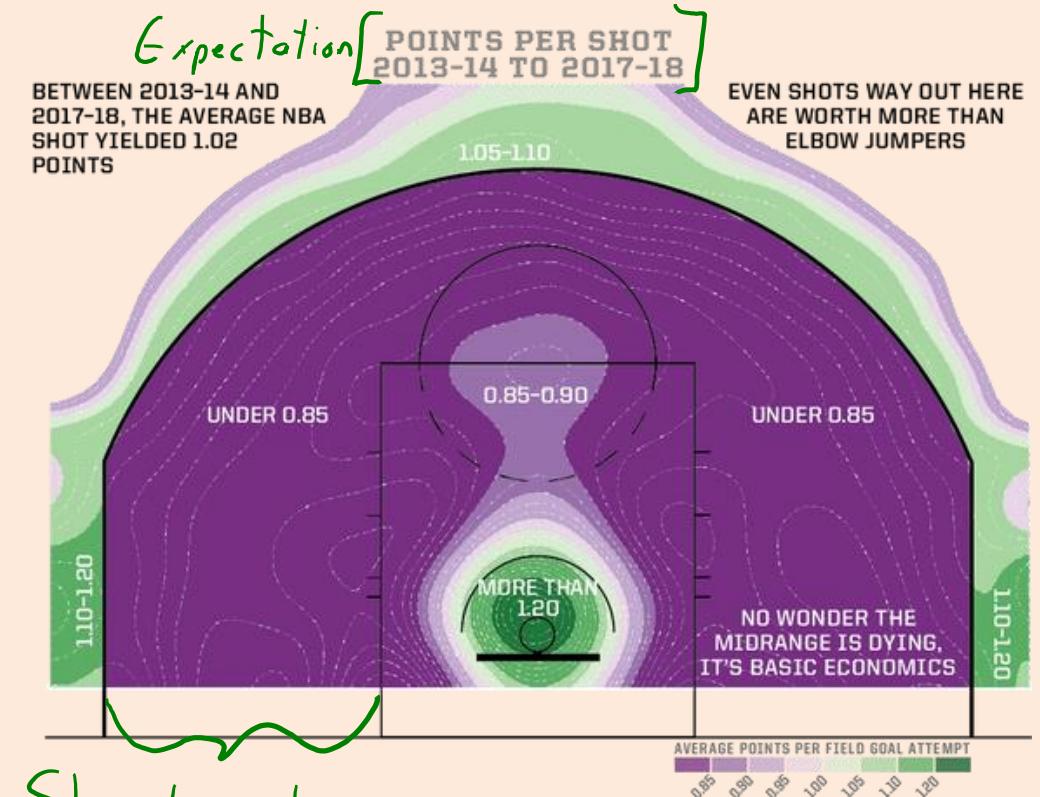
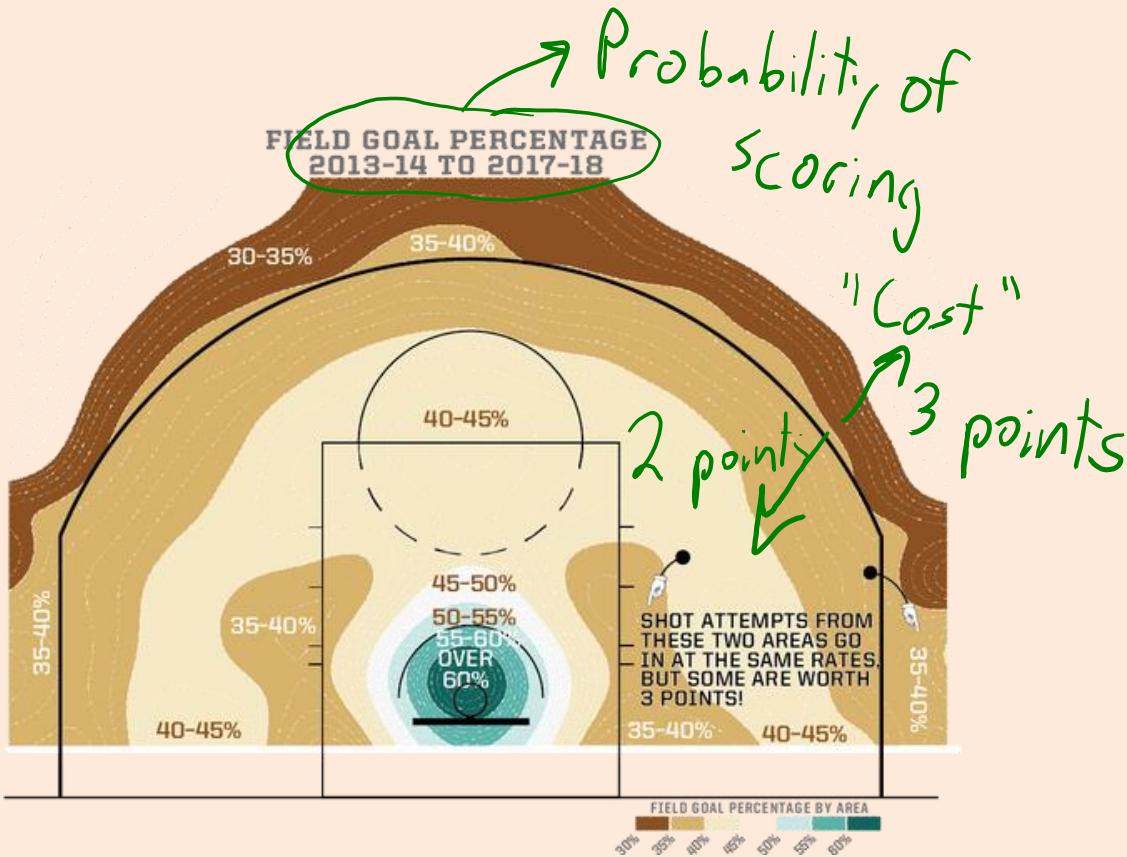
- Even though “spam” is more likely, we should predict “not spam”.

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 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”

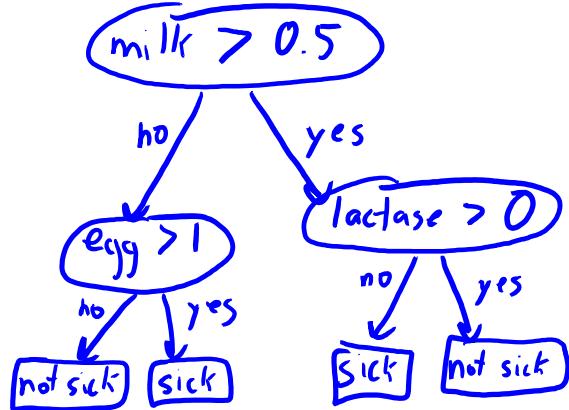


Shooting here is
a bad decision

(pause)

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.
cheap cost for training
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").

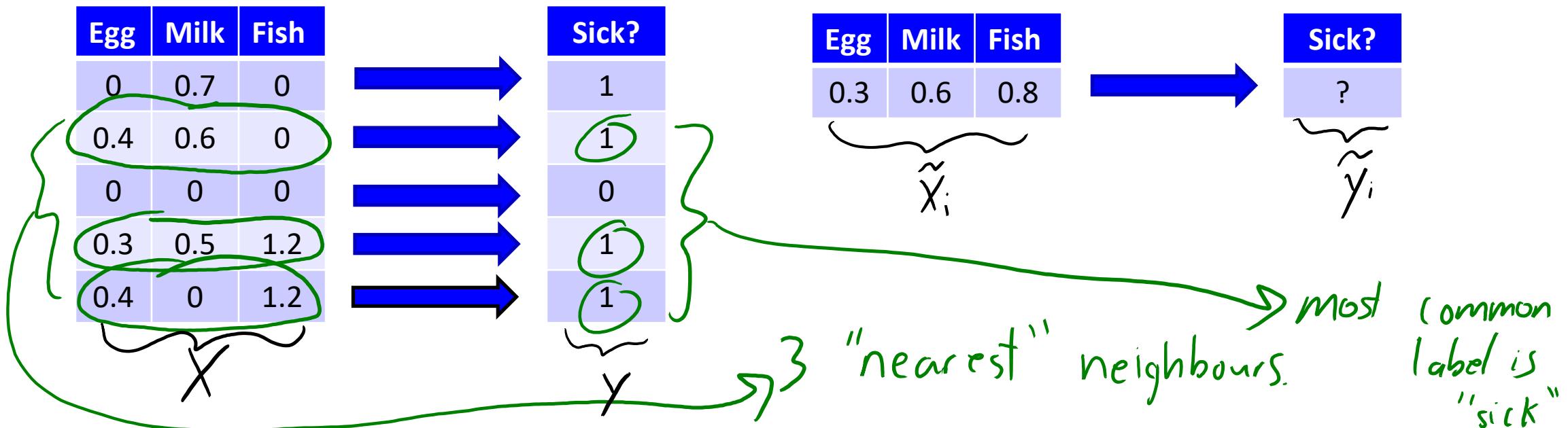
- Naïve Bayes:

$$\begin{aligned} & p(\text{sick} \mid \text{milk}, \text{egg}, \text{lactase}) \\ & \approx p(\text{milk} \mid \text{sick}) p(\text{egg} \mid \text{sick}) p(\text{lactase} \mid \text{sick}) p(\text{sick}) \end{aligned}$$

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 (text).
bag of words

K-Nearest Neighbours (KNN)

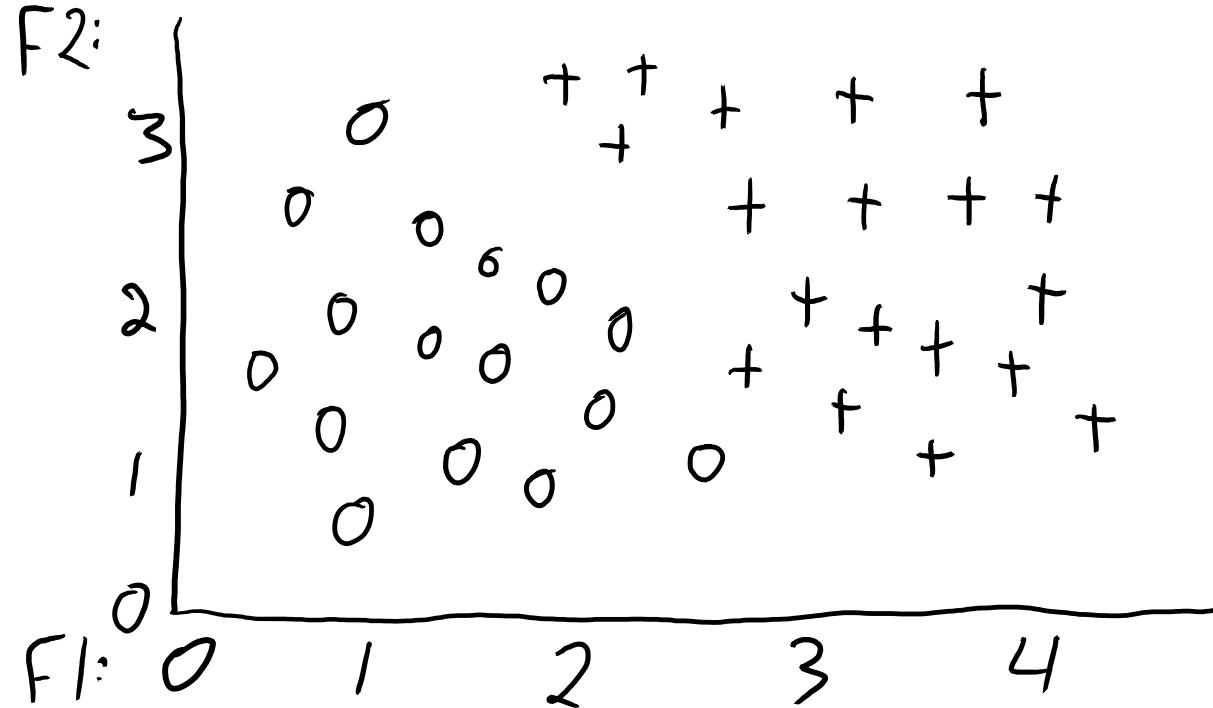
- An old/simple classifier: k-nearest neighbours (KNN).
- To classify an example \tilde{x}_i :
 1. Find the 'k' training examples x_i that are “nearest” to \tilde{x}_i .
 2. Classify using the most common label of “nearest” training examples.



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.

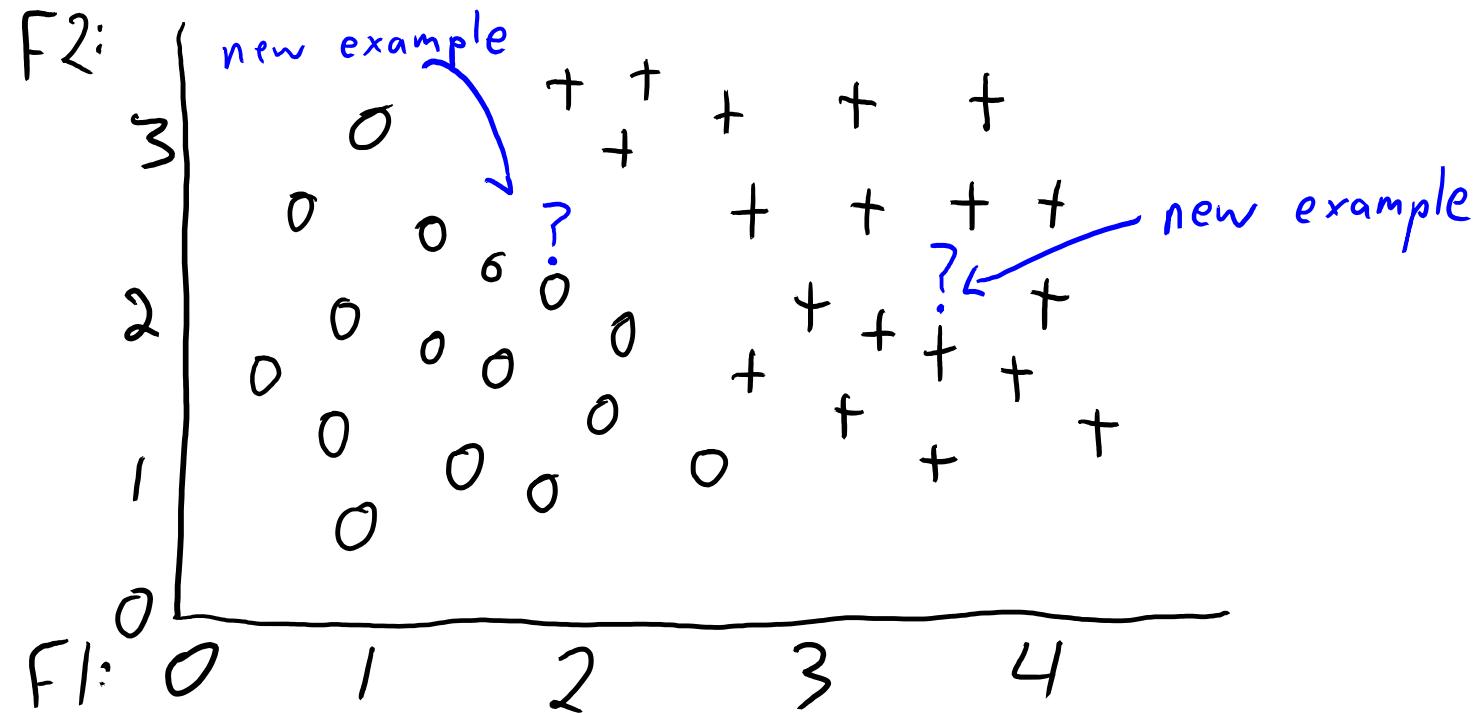
F1	F2	
Label		
1	3	0
2	3	+
3	2	+
2.5	1	0
3.5	1	+
...



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.

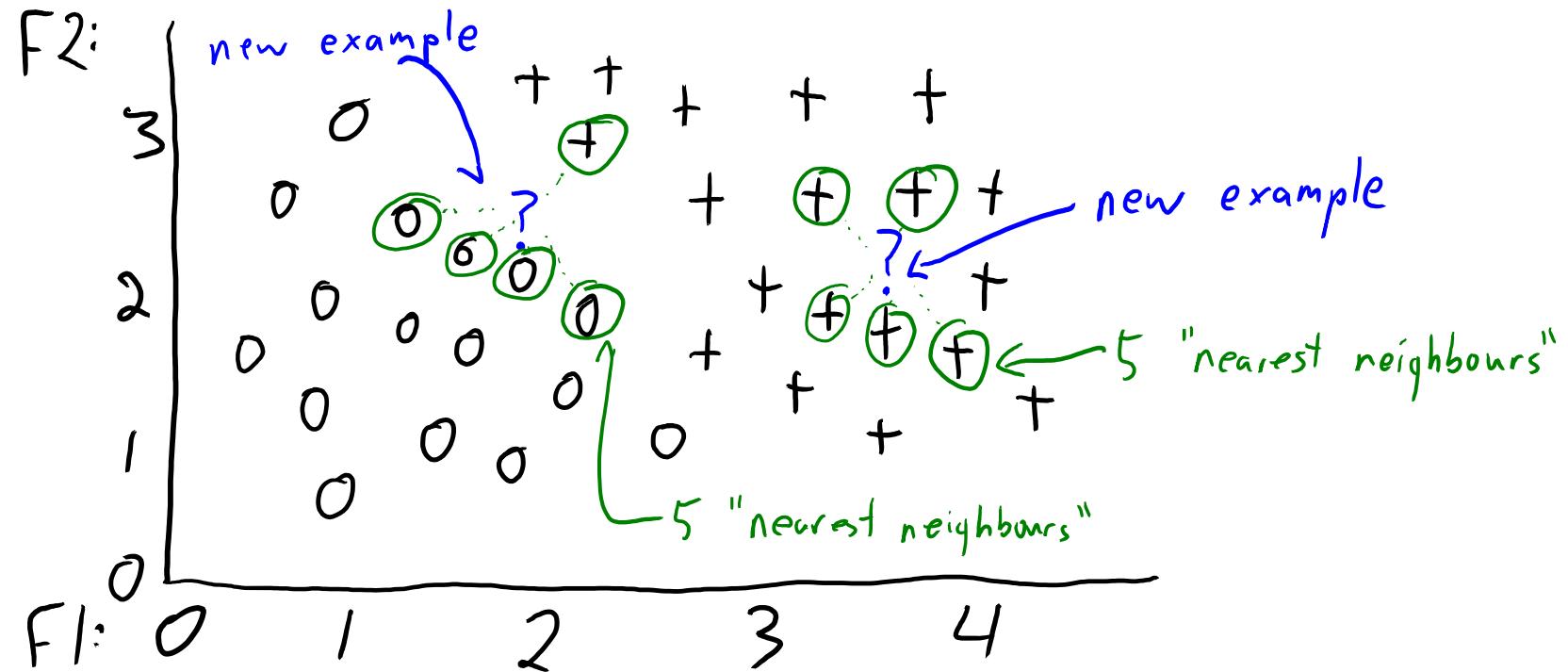
F1	F2	
1	3	→
2	3	→
3	2	→
2.5	1	→
3.5	1	→
...



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.

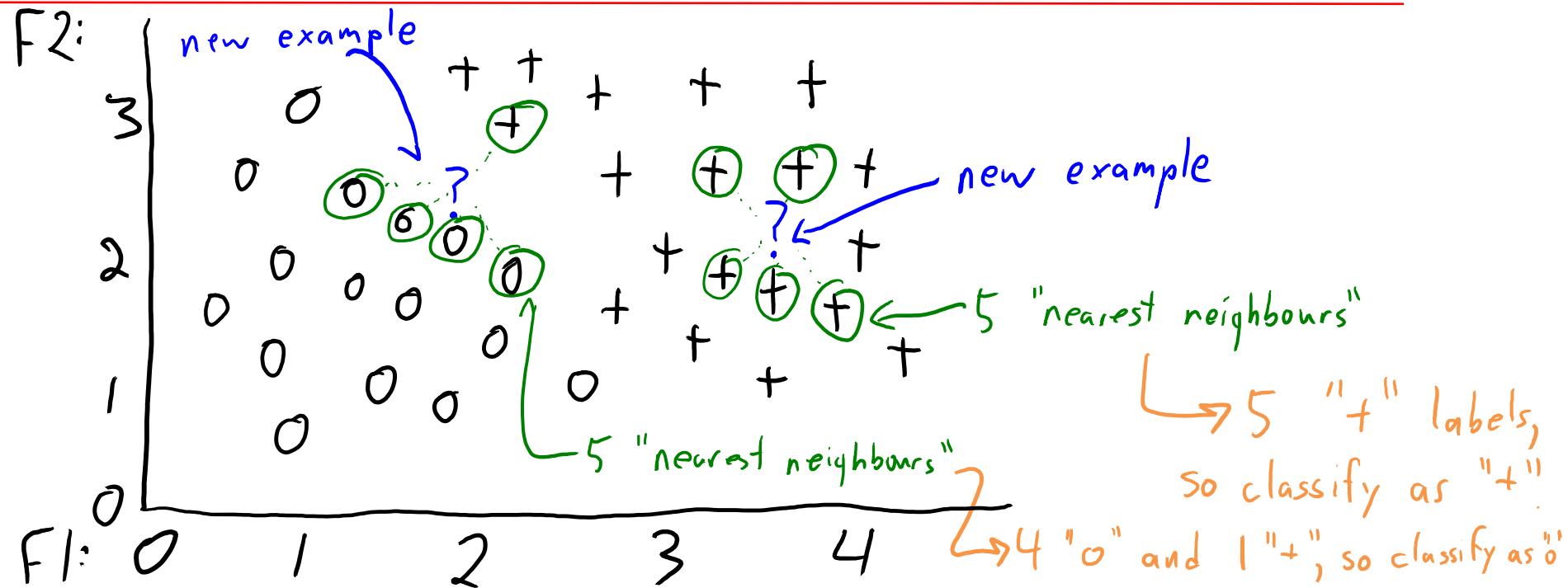
F1	F2	
1	3	→
2	3	→
3	2	→
2.5	1	→
3.5	1	→
...



K-Nearest Neighbours (KNN)

- An old/simple classifier: k-nearest neighbours (KNN).
 - To classify an example \tilde{x}_i :
standardize the data first
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	0
2	3	+
3	2	+
2.5	1	0
3.5	1	+
...



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:

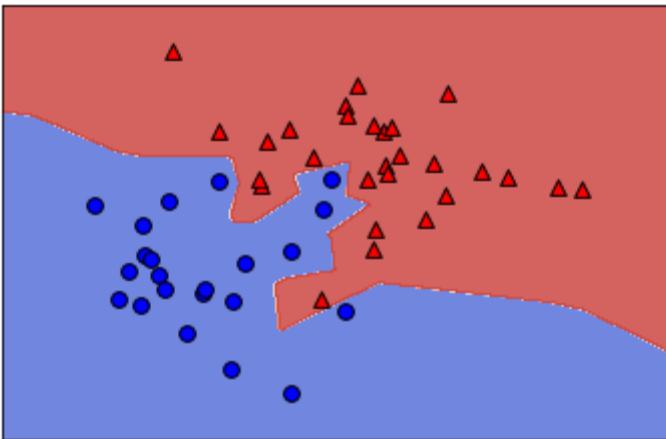
$$\|x_i - \tilde{x}_{\tilde{i}}\| = \sqrt{\sum_{j=1}^d (x_{ij} - \tilde{x}_{\tilde{i}j})^2}$$

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

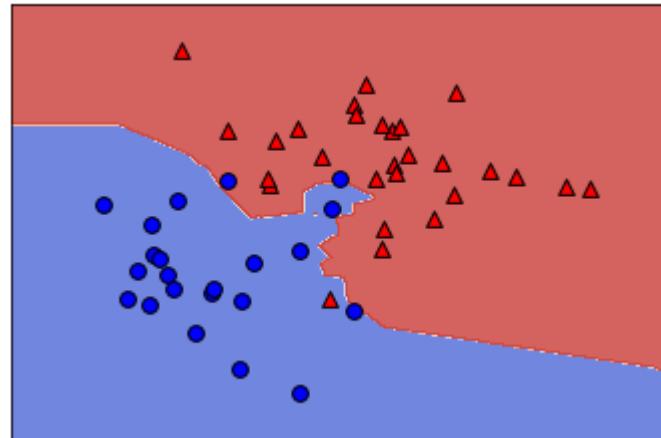
Effect of 'k' in KNN.

- 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.

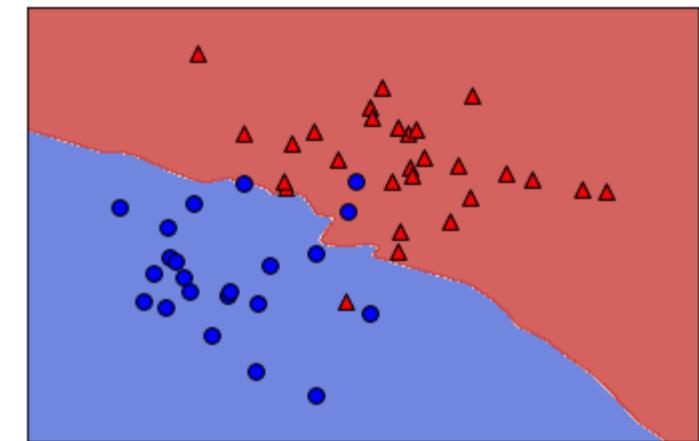
$k=1$



$k=3$



$k=10$



- Effect of 'k' on fundamental trade-off:
 - As 'k' grows, training error increase and approximation error decreases.

KNN Implementation

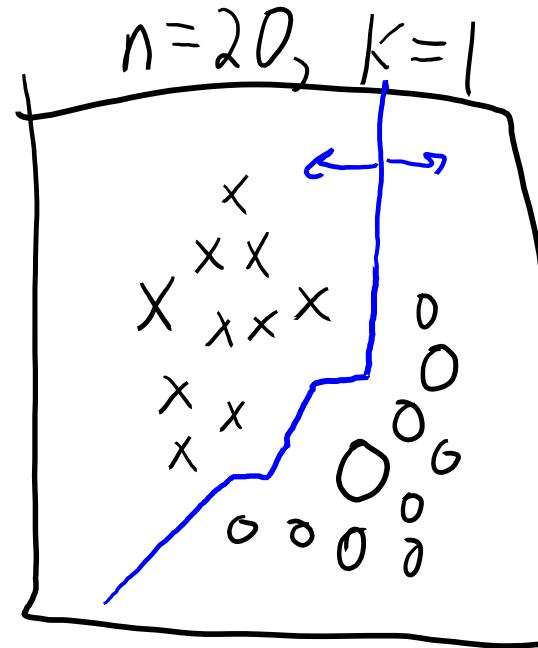
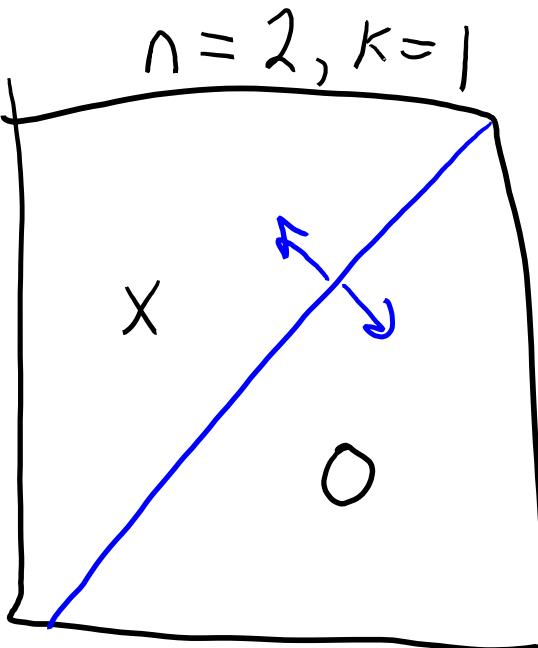
- There is **no training** phase in KNN (“lazy” learning).
 - You just store the training data.
 - Costs $O(1)$ if you use a pointer.
- But **predictions are expensive**: $O(nd)$ to classify 1 test example.
 - Need to do $O(d)$ distance calculation for all ‘n’ training examples.
 - So **prediction time grows with number of training examples**.
 - Tons of work on reducing this cost (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**.

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*.

Effect of 'n' in KNN.

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



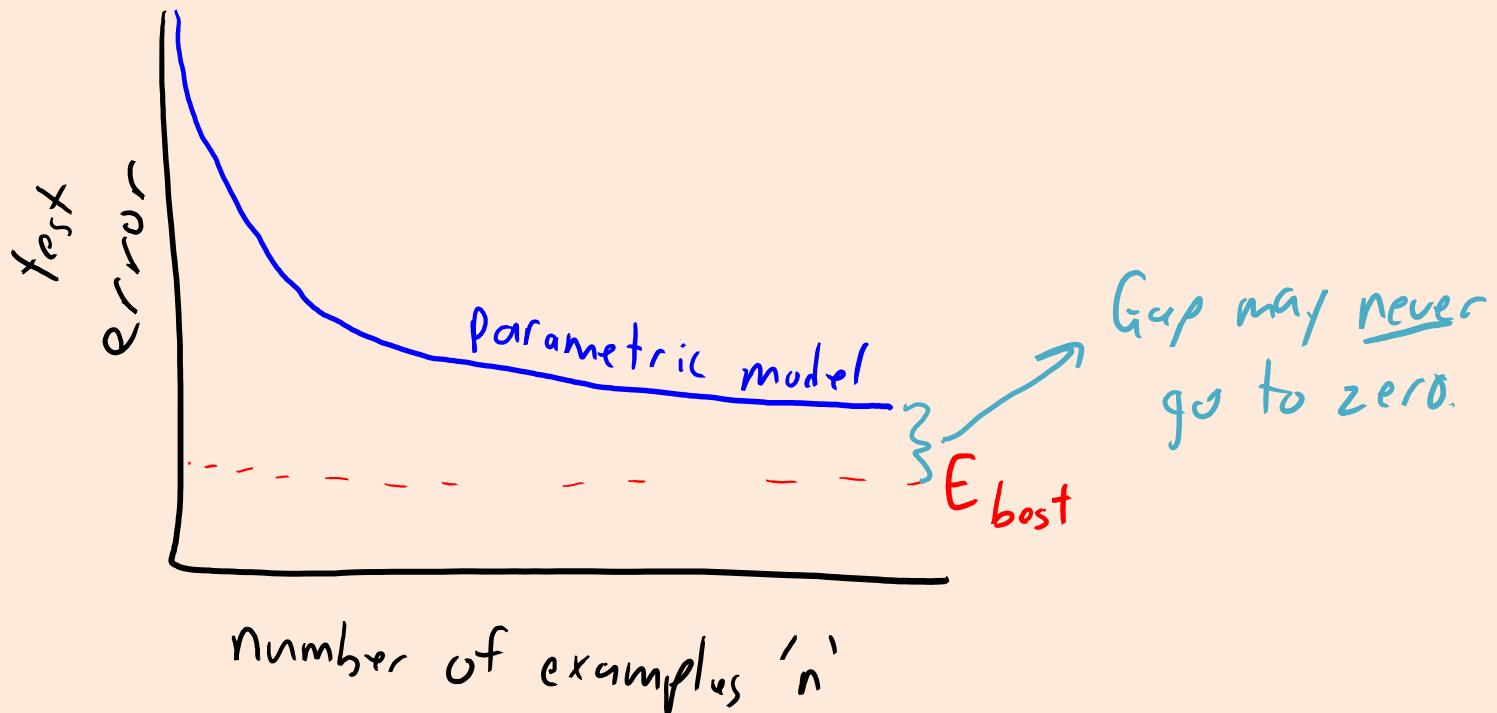
- Model gets more complicated as 'n' increases.
 - Requires more memory, but detects subtle differences between examples.

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**.
 - 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](#)").

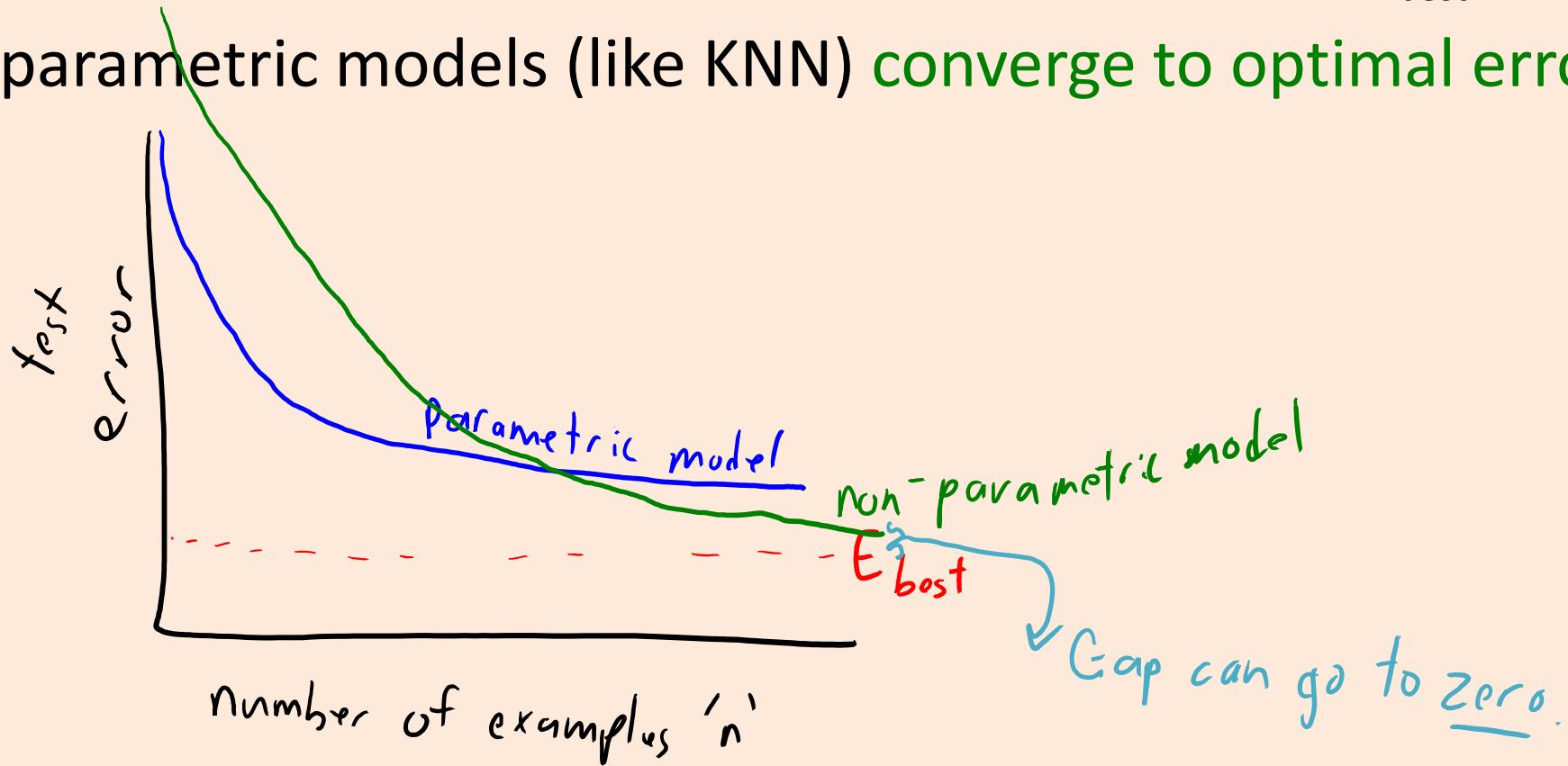
Parametric vs. Non-Parametric Models

- 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**.



Curse of Dimensionality

- “**Curse of dimensionality**”: problems with high-dimensional spaces.
 - Volume of space grows **exponentially** with dimension.
 - Circle has area $O(r^2)$, sphere has area $O(r^3)$, 4d hyper-sphere has area $O(r^4),\dots$
 - Need **exponentially more points** to ‘fill’ a high-dimensional volume.
 - “Nearest” neighbours might be really far even with large ‘n’.
- KNN is also problematic if features have very different scales.
- Nevertheless, KNN is really easy to use and often hard to beat!

Defining “Distance” with “Norms”

- A common way to define the “distance” between examples:
 - Take the “norm” of the difference between feature vectors.

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

train example test example "L₂-norm"

- Norms are a way to measure the “length” of a vector.
 - The most common norm is the “L2-norm” (or “Euclidean norm”):

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

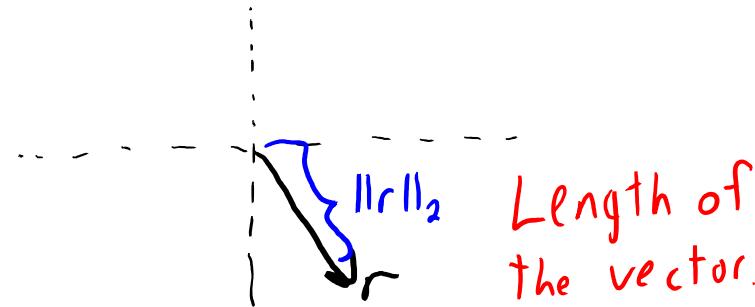
- Here, the “norm” of the difference is the standard Euclidean distance.

L₂-norm, L₁-norm, and L _{∞} -Norms.

- The three most common norms: L₂-norm, L₁-norm, and L _{∞} -norm.
 - Definitions of these norms with two-dimensions:

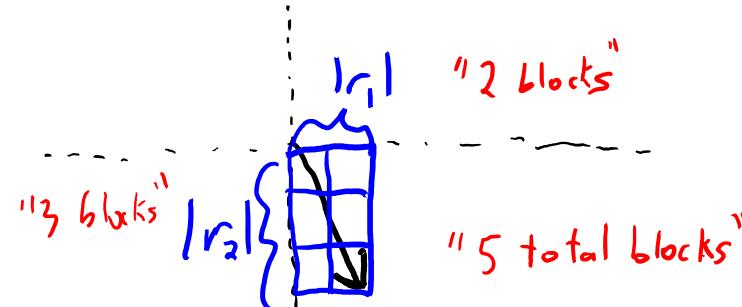
L₂ or "Euclidean" norm.

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



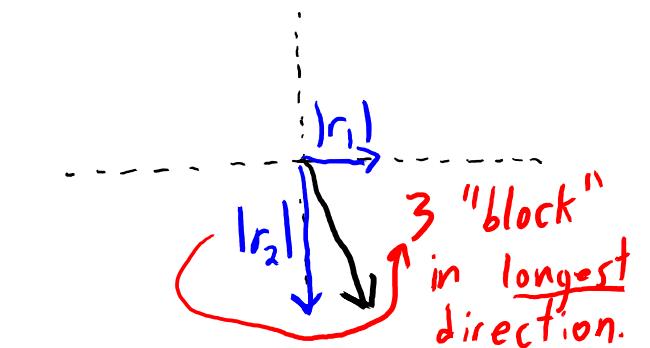
L₁ or "Manhattan" norm:

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



L _{∞} or "max" norm:

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



- Notation: we often leave out the "2" for the L₂-norm:

$$\|r\| = \|r\|_2$$

Norms in d-Dimensions

- We can generalize these common norms to d-dimensional vectors:

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

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

$$L_\infty: \max_j \{|r_j|\}$$

E.g., in 3-dimensions:

$$\|r\|_2 = \sqrt{r_1^2 + r_2^2 + r_3^2}$$

in 4-dimensions:

$$\|r\|_2 = \sqrt{r_1^2 + r_2^2 + r_3^2 + r_4^2}$$

- These norms place different “weights” on large values:

- L_1 : all values are equal.

- L_2 : bigger values are more important (because of squaring).

- L_∞ : only biggest value is important.

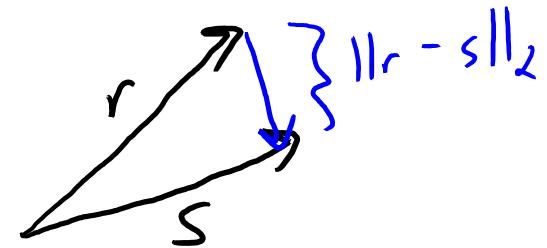
Notation: $\|r\|_2^2 = (\|r\|_2)^2$
 $= (\sqrt{\sum_{j=1}^d r_j^2})^2$
 $= \sum_{j=1}^d r_j^2$
 $= r^T r$

Different ways
to write the same thing

Norms as Measures of Distance

- By taking norm of difference, we get a “distance” between vectors:

$$\begin{aligned}\|r - s\|_2 &= \sqrt{(r_1 - s_1)^2 + (r_2 - s_2)^2} \\ &= \|r - s\| \text{ "Euclidean distance"}$$



$$\|r - s\|_1 = |r_1 - s_1| + |r_2 - s_2|$$

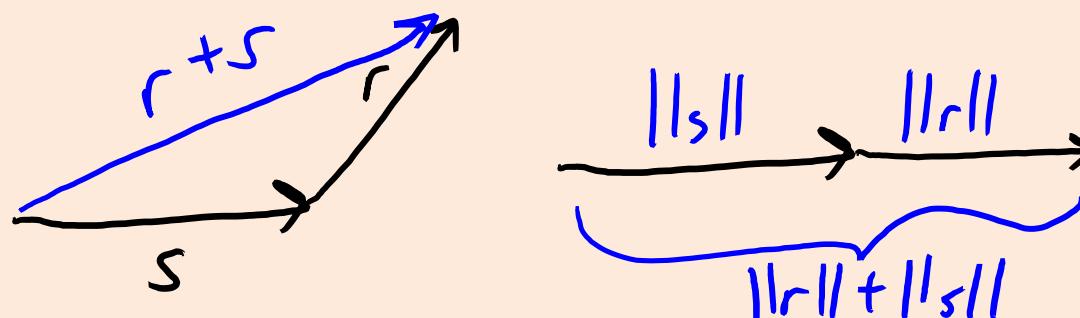
"Number of blocks you need to walk to get from r to s."

$$\|r - s\|_\infty = \max \{|r_1 - s_1|, |r_2 - s_2|\}$$

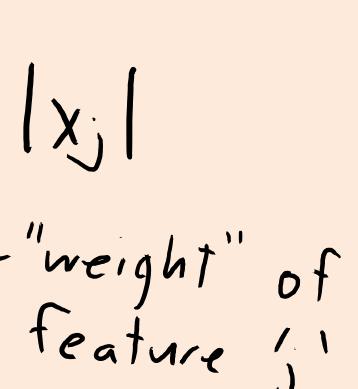
"Most number of blocks in any direction you would have to walk."

3 Defining Properties of Norms

- A “norm” is any function satisfying the following 3 properties:
 1. Only ‘0’ has a ‘length’ of zero.
 2. Multiplying ‘r’ by constant ‘ α ’ multiplies length by $|\alpha|$
 - “If r will twice as long if you multiply by 2”: $\| \alpha r \| = |\alpha| \cdot \| r \|$.
 - Implication is that norms cannot be negative.
 3. Length of ‘ $r+s$ ’ is not more than length of ‘ r ’ plus length of ‘ s ’:
 - “You can’t get there faster by a detour”.
 - “Triangle inequality”: $\| r + s \| \leq \| r \| + \| s \|$.



KNN Distance Functions

- Most common KNN distance functions: $\text{norm}(x_i - x_j)$.
 - L1-, L2-, and L^∞ -norm.
 - Weighted norms (if some features are more important): $\sum_{j=1}^d v_j |x_j|$

 - “Mahalanobis” distance (takes into account correlations).
- But we can consider other distance/similarity functions:
 - Jaccard similarity (if x_i are sets).
 - Edit distance (if x_i are strings).
 - Metric learning (*learn* the best distance function).

Summary

- **Decision theory** allows us to consider costs of predictions.
- **Non-parametric models** grow with number of training examples.
- **K-Nearest Neighbours:** simple non-parametric classifier.
 - Appealing “consistency” properties.
 - Suffers from high prediction cost and curse of dimensionality.
- 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 ($y_i = c$).

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

$n_1 = 6$



$n_0 = 4$



Naïve Bayes Training Phase

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

- Training a naïve Bayes model:

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

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

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

Naïve Bayes Training Phase

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

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

0	1	1
1	1	1
0	0	1
1	1	1
1	1	1
0	0	1
1	0	0
1	0	0
1	1	0
1	0	0

, $y =$

$n_{121} = 4$

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

Naïve Bayes Training Phase

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

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

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

0	1	1
1	1	1
0	0	1
1	1	1
1	1	1
0	0	1
1	0	0
1	0	0
1	1	0
1	0	0

, $y =$

$n_{121} = 4$

$$p(x_{12} = 1, y_i = 1) = \frac{4}{10}$$

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

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

$$\begin{aligned} 5. \text{ Use that } p(x_{ij} = k | y_i = c) &= \frac{p(x_{ij} = k, y_i = c)}{p(y_i = c)} \\ &= \frac{n_{cjk}/n}{n_c/n} = \frac{n_{cjk}}{n_c} \end{aligned}$$

$p(x_{12} = 1 | y_i = 1) = \frac{4}{6} = \frac{2}{3}$

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

0	1	1	1
1	1		1
0	0		1
1	1		1
1	1		1
0	0	,	1
1	0		0
1	0		0
1	1		0
1	0		0

$$n_{121} = 4$$

$$n_0 = 4$$

$$p(x_{12} = 1, y_i = 1) = \frac{4}{10}$$

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

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 $p(\tilde{x}_i | \hat{y}_i = c)$

Under the naive Bayes assumption we can maximize:

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

Naïve Bayes Prediction Phase

- Prediction in a naïve Bayes model:

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

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

Naïve Bayes Prediction Phase

- Prediction in a naïve Bayes model:

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

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

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

Naïve Bayes Prediction Phase

- Prediction in a naïve Bayes model:

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

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

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

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

Naïve Bayes Prediction Phase

- Prediction in a naïve Bayes model:

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

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

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

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

Since $p(\tilde{y}_i=1 | \tilde{x}_i)$ is bigger than $p(\tilde{y}_i=0 | \tilde{x}_i)$, naïve 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:

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

- 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 :

$$\beta \exp(-1^2) + \beta \exp(-2^2) + \beta \exp(-3^2) + \beta \exp(-4^2) = 1$$
$$\Leftrightarrow \beta [\underbrace{\exp(-1^2) + \exp(-2^2) + \exp(-3^2) + \exp(-4^2)}_1] = 1$$

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

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:

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

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

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

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

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

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

Less-Naïve Bayes

- Given features $\{x_1, x_2, x_3, \dots, x_d\}$, naïve Bayes approximates $p(y|x)$ as:

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

- 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:

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

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

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

These are the quantities
we compute during training

Gaussian Discriminant Analysis

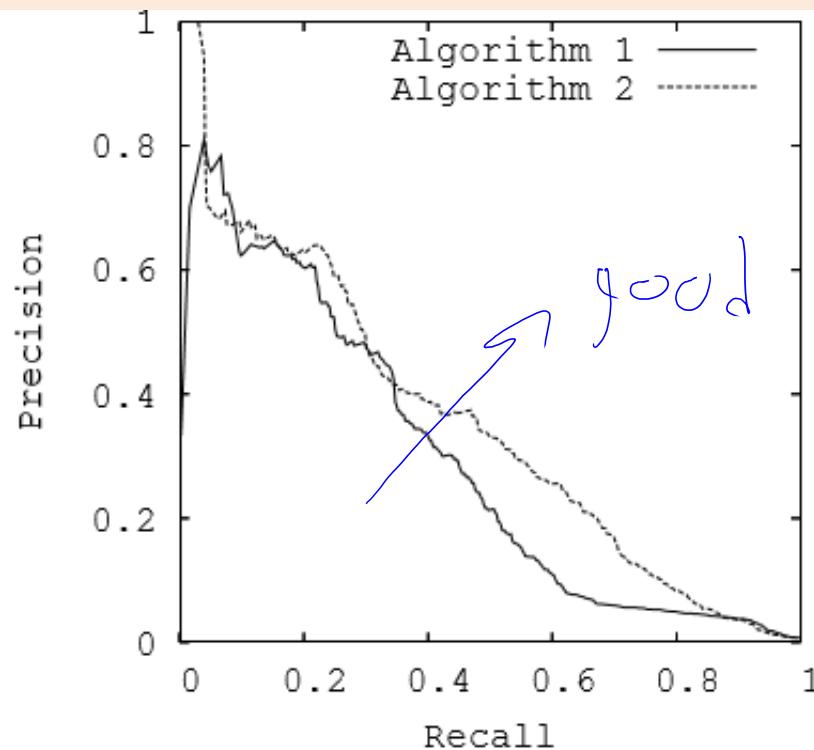
- Classifiers based on Bayes rule are called **generative classifier**:
 - They often work well when you have **tons of features**.
 - But they **need to know $p(x_i | y_i)$** , probability of features given the class.
 - How to “generate” features, based on the class label.
- To fit generative models,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

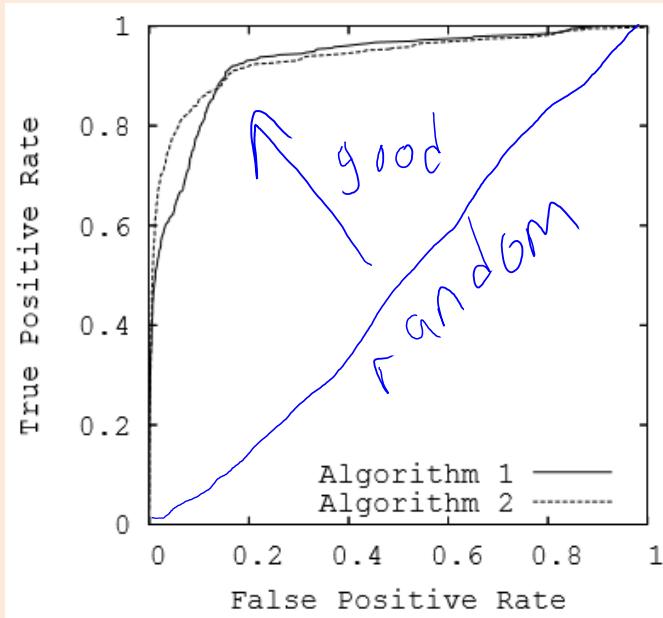
Precision-Recall Curve

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



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.

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:
$$\|x_i - y_j\|^2 = \|x_i\|^2 - 2x_i^T x_j + \|x_j\|^2$$
where ‘i’ is a training example and ‘j’ is a test example.
 - We can compute D in Julia using:

```
x1.^2*ones(d, t) .+ ones(n, d) * (x2').^2 .- 2x1*x2'
```

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

Squared/Euclidean-Norm Notation

We're using the following conventions:

The subscript after the norm is used to denote the p-norm, as in these examples:

$$\|x\|_2 = \sqrt{\sum_{j=1}^d w_j^2}.$$

$$\|x\|_1 = \sum_{j=1}^d |w_j|.$$

If the subscript is omitted, we mean the 2-norm:

$$\|x\| = \|x\|_2.$$

If we want to talk about the *squared* value of the norm we use a superscript of "2":

$$\|x\|_2^2 = \sum_{j=1}^d w_j^2.$$

$$\|x\|_1^2 = \left(\sum_{j=1}^d |w_j| \right)^2.$$

If we omit the subscript and have a superscript of "2", we're taking about the squared L2-norm:

$$\|x\|^2 = \sum_{j=1}^d w_j^2.$$

L_p-norms

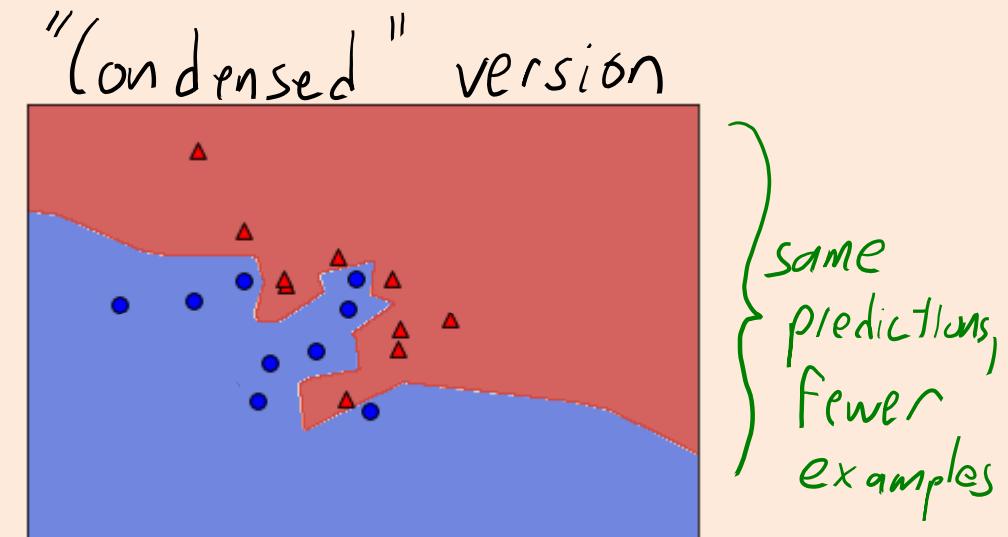
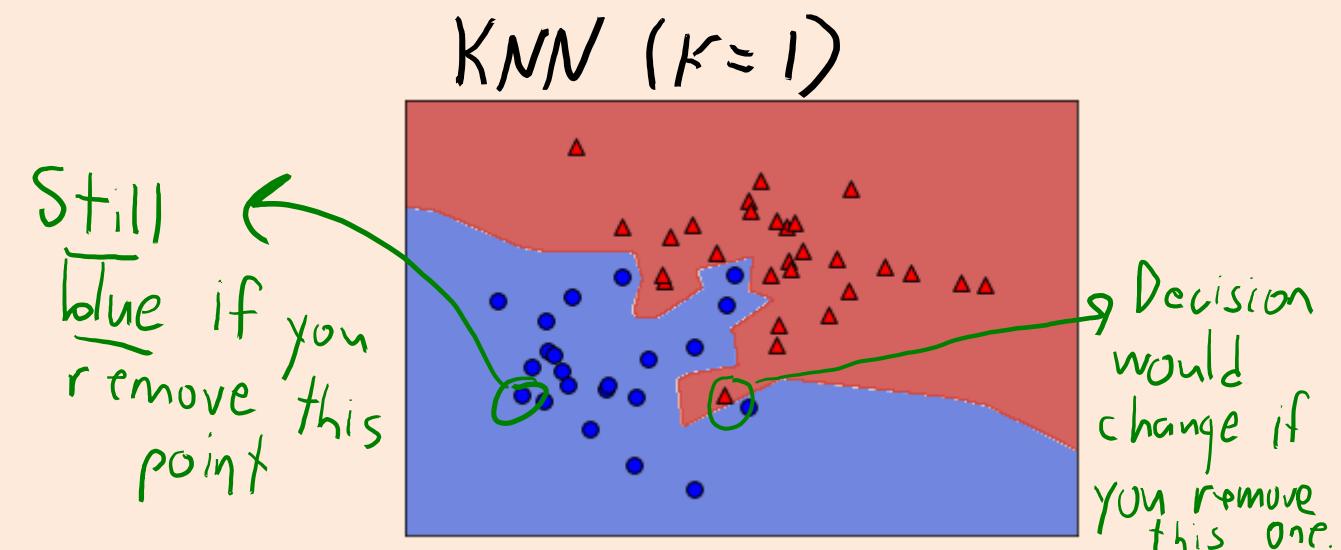
- The L₁-, L₂-, and L_∞-norms are special cases of L_p-norms:

$$\|x\|_p = \left(\sum_{j=1}^d x_j^p \right)^{1/p}$$

- This gives a norm for any (real-valued) $p \geq 1$.
 - The L_∞-norm is limit as 'p' goes to ∞ .
- For $p < 1$, not a norm because triangle inequality not satisfied.

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 $O(md)$.



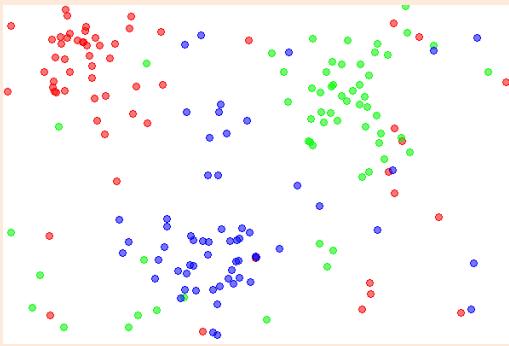
Condensed Nearest Neighbours

- 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”).

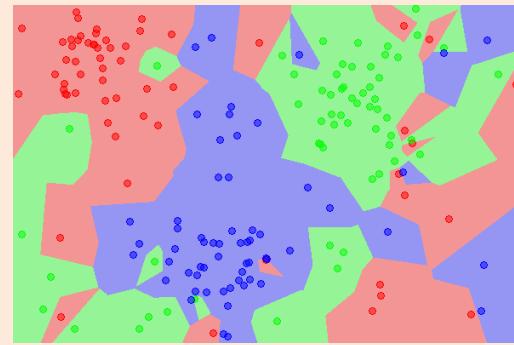
Condensed Nearest Neighbours

- Classic condensed nearest neighbours:

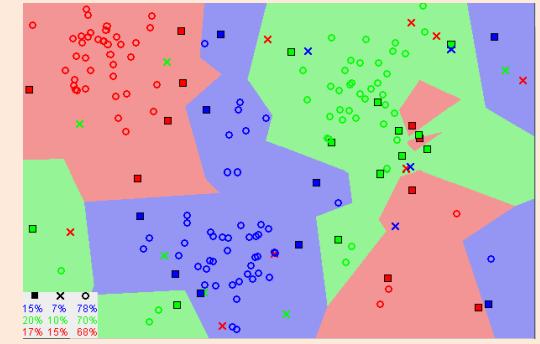
Data:



1NN



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



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