

CPSC 340: Machine Learning and Data Mining

Non-Parametric Models

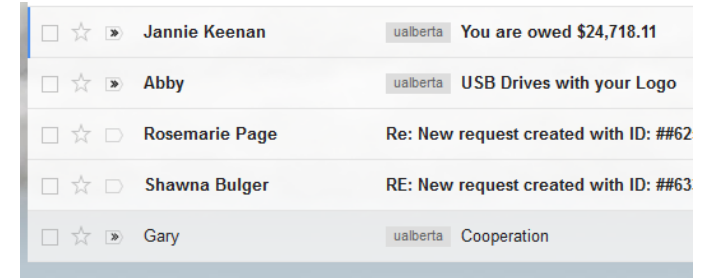
Fall 2018

Admin

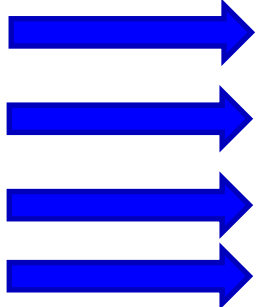
- Course webpage: www.ugrad.cs.ubc.ca/~cs340
- **Assignment 2** is out.
 - Due Friday of next week. It's long so start early.
- **Add/drop deadline** is tomorrow.
- **Auditing**: message me on Piazza if you want to audit.
 - Bring your form to me after class.

Last Time: E-mail Spam Filtering

- Want a build a system that filters spam e-mails:
- We formulated as supervised learning:
 - $(y_i = 1)$ if e-mail 'i' is spam, $(y_i = 0)$ if e-mail is not spam.
 - $(x_{ij} = 1)$ if word/phrase 'j' is in e-mail 'i', $(x_{ij} = 0)$ if it is not.



\$	Hi	CPSC	340	Vicodin	Offer	...	Spam?
1	1	0	0	1	0	...	1
0	0	0	0	1	1	...	1
0	1	1	1	0	0	...	0
...



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:

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

- 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.

Laplace Smoothing

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

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

- But there is a problem if you have **no spam messages with lactase**:

- $p(\text{'lactase'} \mid \text{'spam'}) = 0$, so spam messages with lactase automatically get through.

- Common fix is **Laplace smoothing**:

- **Add 1 to numerator**,
and 2 to denominator (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 sums to 1).

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

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 what **minimizing expected cost**:

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

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 higher cost.

Decision Theory Example

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

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

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

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

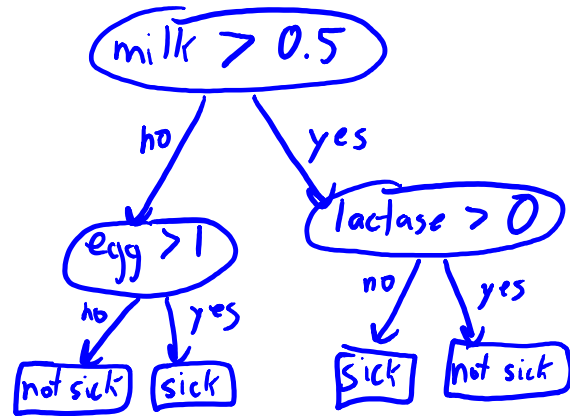
- Even though "spam" is more likely, we should predict "not spam".

Decision Theory Discussion

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

- 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 (text).

Parametric vs. Non-Parametric

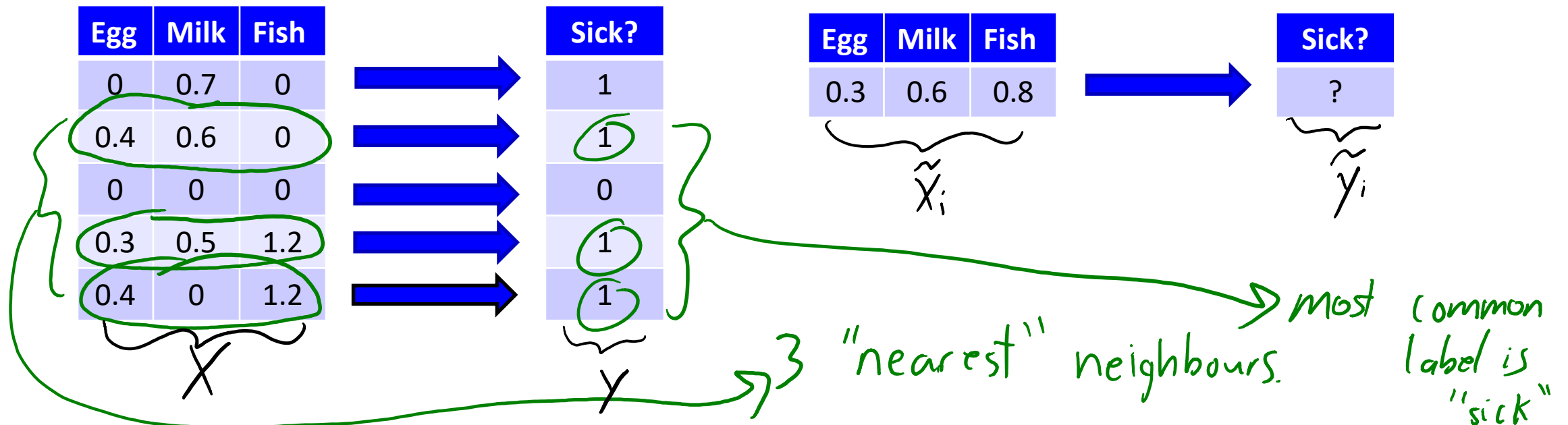
- Decision trees and naïve Bayes are often **not very accurate**.
 - Greedy rules or conditional independence might be bad assumptions.
 - They are also **parametric** models.

Parametric vs. Non-Parametric

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

K-Nearest Neighbours (KNN)

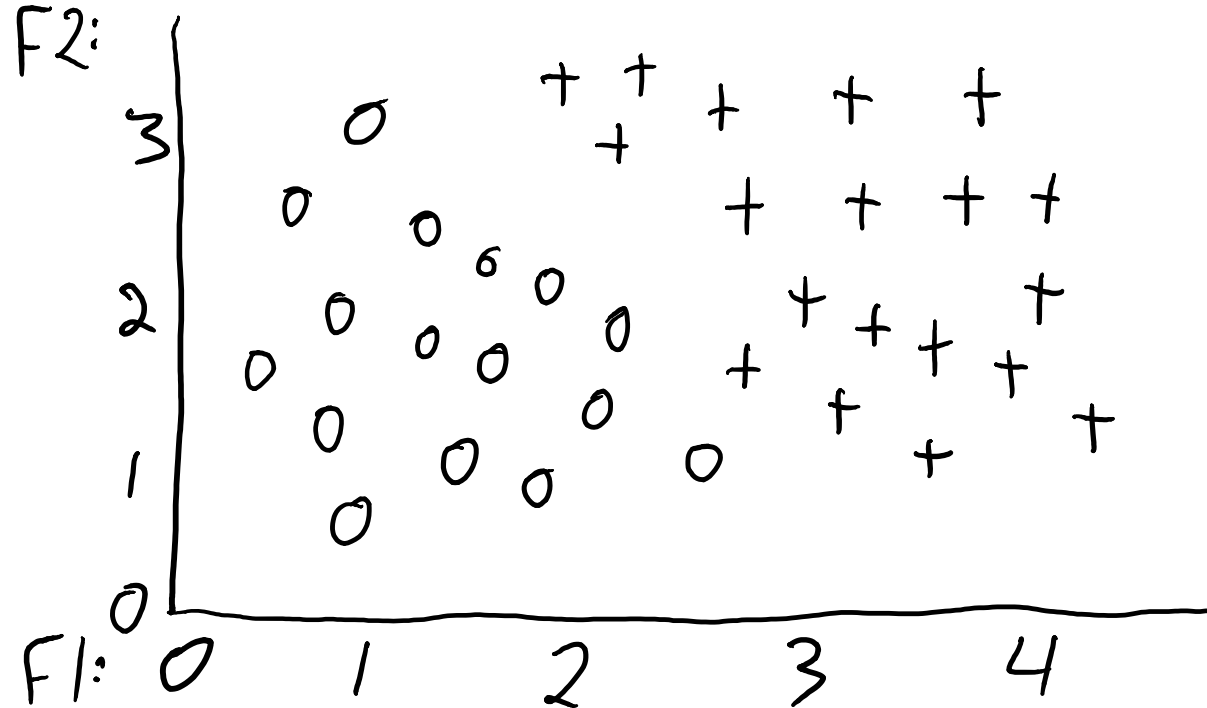
- Classical non-parametric classifier is **k-nearest neighbours (KNN)**.
- To classify an example \tilde{x}_i :
 1. Find the 'k' training examples x_i that are "nearest" to \tilde{x}_i .
 2. Classify using the **most common label** of "nearest" examples.



K-Nearest Neighbours (KNN)

- Classical non-parametric classifier is **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” examples.

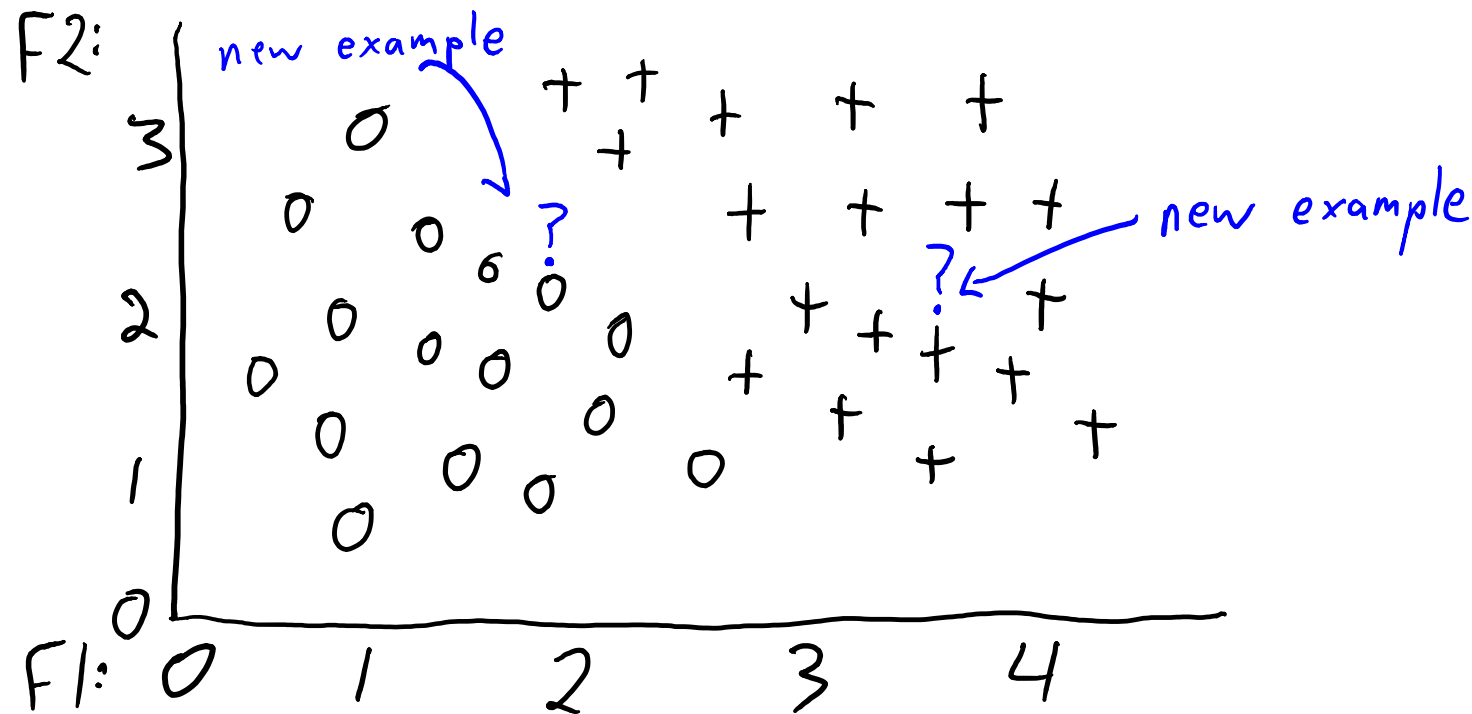
F1	F2	Label
1	3	0
2	3	+
3	2	+
2.5	1	0
3.5	1	+
...



K-Nearest Neighbours (KNN)

- Classical non-parametric classifier is **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" examples.

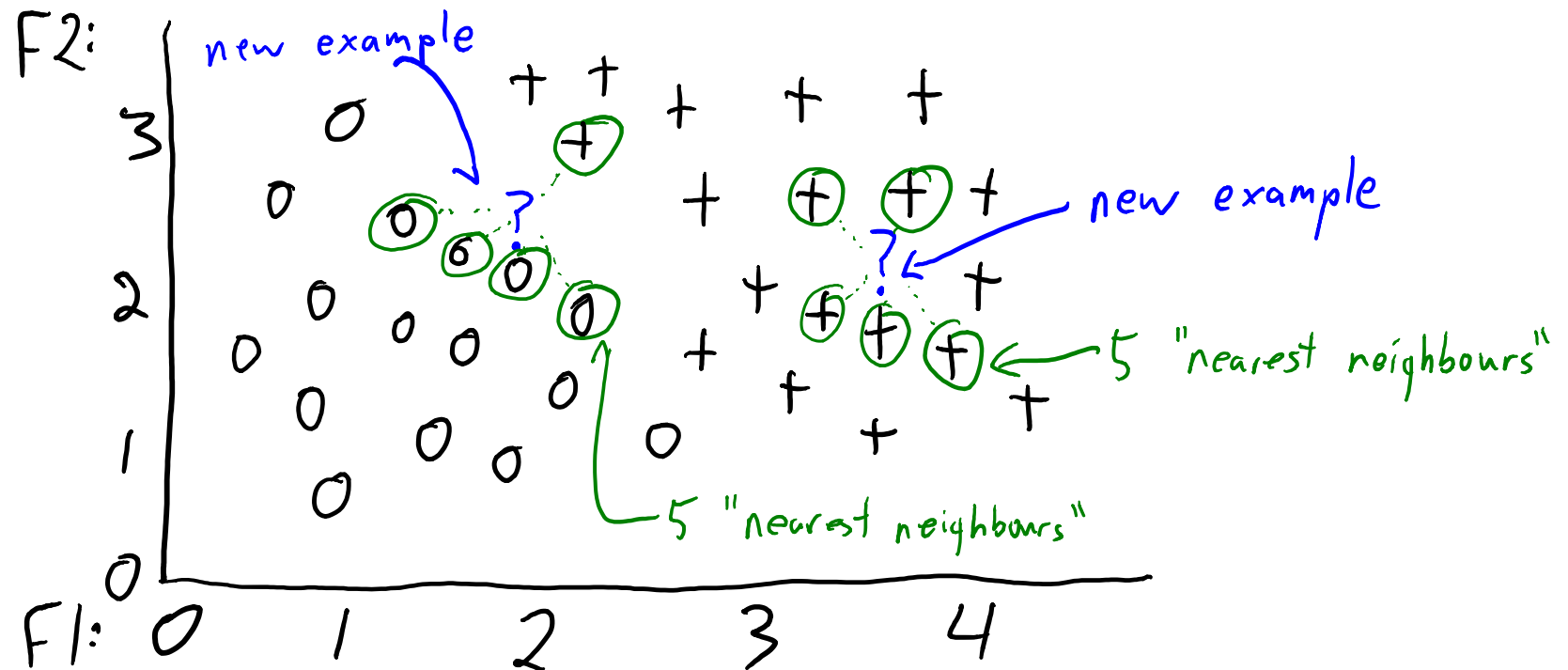
F1	F2	Label
1	3	0
2	3	+
3	2	+
2.5	1	0
3.5	1	+
...



K-Nearest Neighbours (KNN)

- Classical non-parametric classifier is **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" examples.

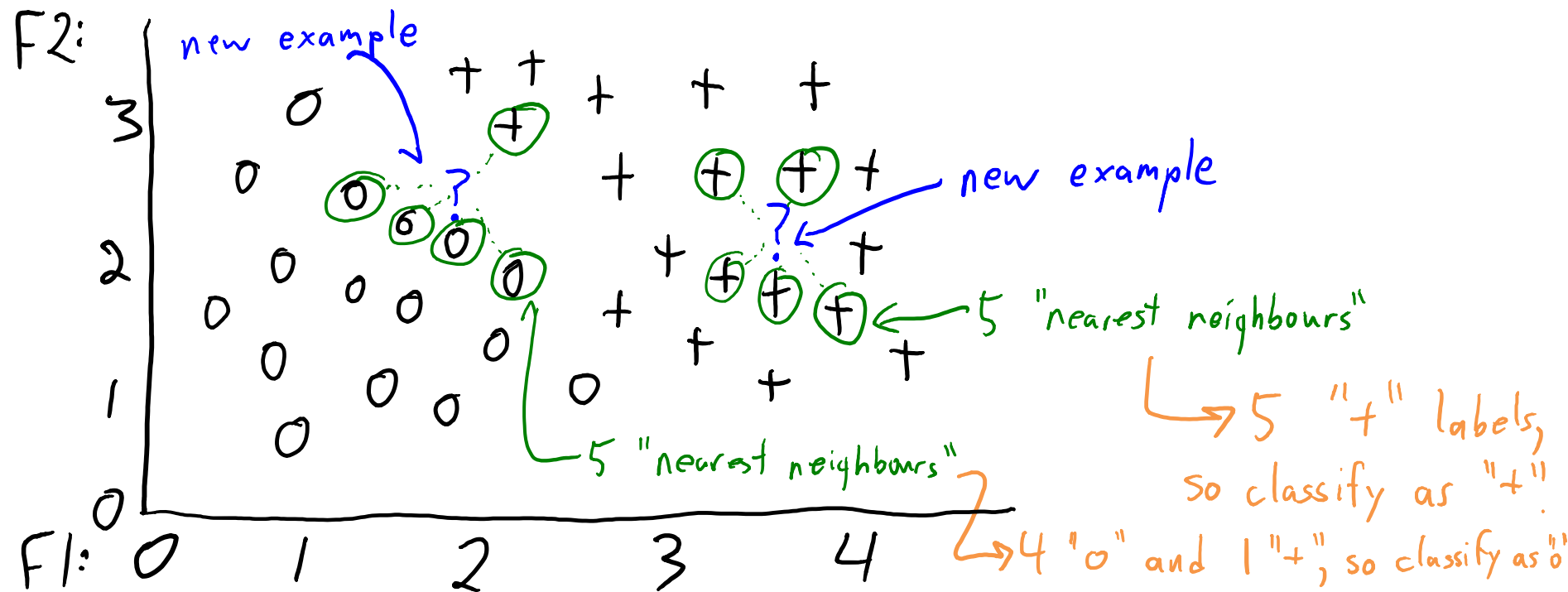
F1	F2	Label
1	3	0
2	3	+
3	2	+
2.5	1	0
3.5	1	+
...



K-Nearest Neighbours (KNN)

- Classical non-parametric classifier is **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" 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.
- Most common distance function is **Euclidean distance**:

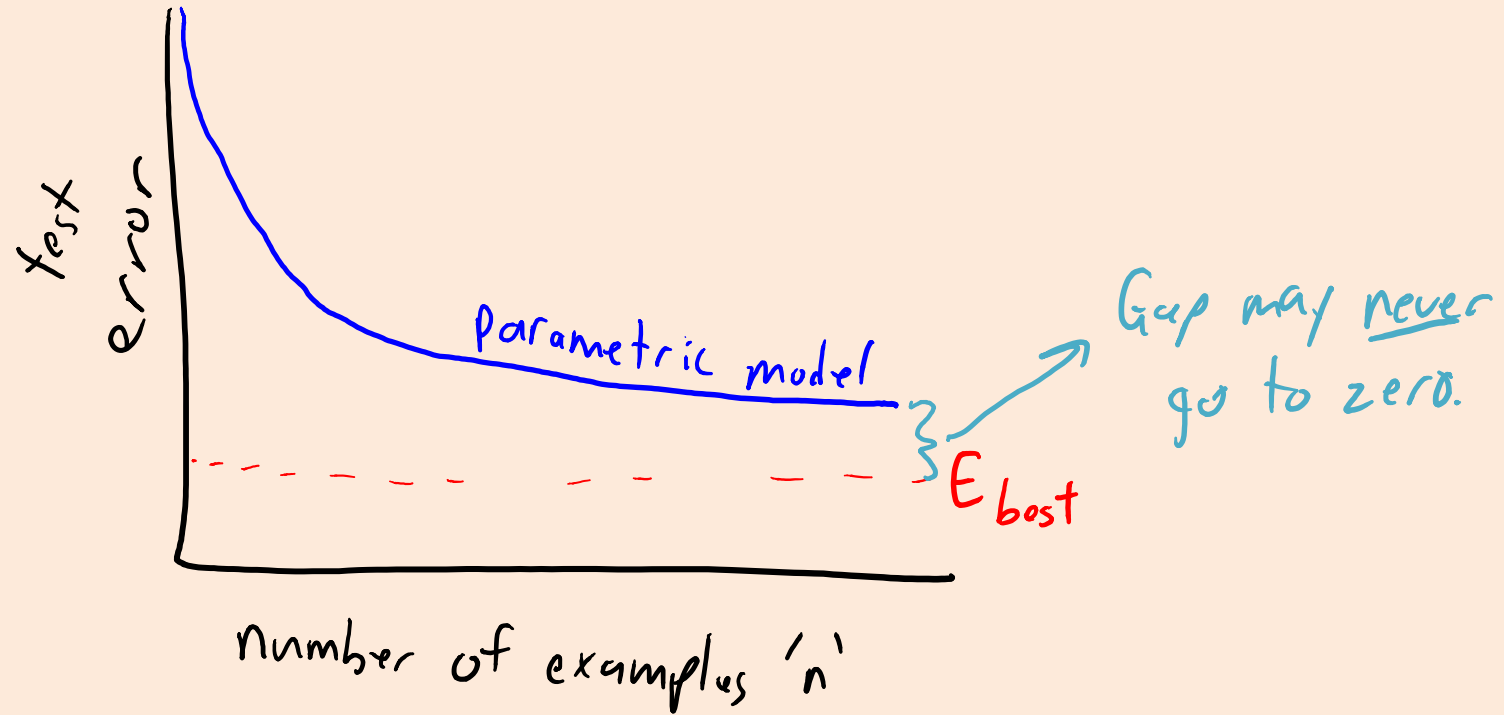
$$\|x_i - \tilde{x}_{\tilde{i}}\| = \sqrt{\sum_{j=1}^d (x_{ij} - \tilde{x}_{\tilde{i}j})^2}$$

- x_i is features of training example 'i', and $\tilde{x}_{\tilde{i}}$ is features of test example ' \tilde{i} '.
- With a small 'n', KNN model will be very simple.
- Model **gets more complicated as 'n' increases**.
 - Starts to detect subtle differences between examples.

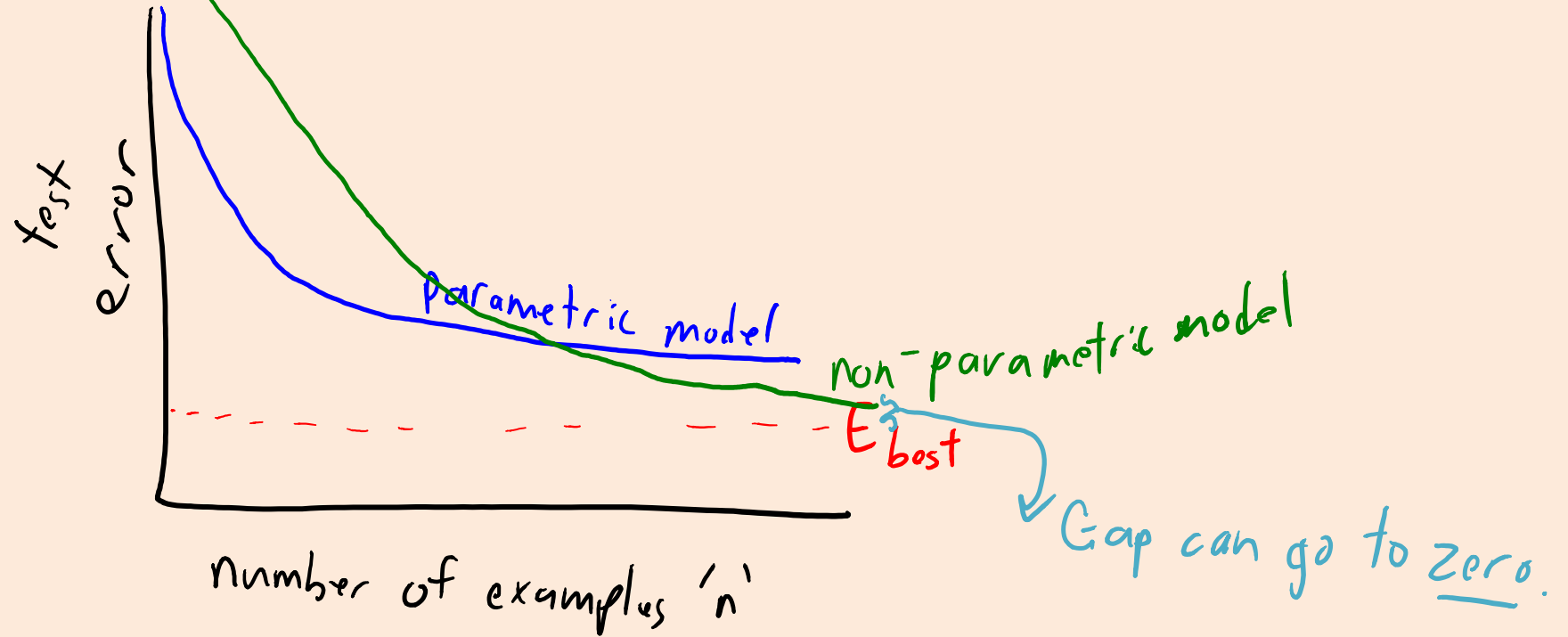
Consistency of KNN

- 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



Parametric vs. Non-Parametric Models



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 $O(r^4)$,...
 - 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!**

KNN Implementation

- There is **no training** phase in KNN (“lazy” learning).
 - You just store the training data.
 - **Non-parametric** because the **size of the model is $O(nd)$** , the size of ‘X’.
- But **predictions are expensive**: $O(nd)$ to classify 1 test example.
 - Tons of work on reducing this cost (we’ll discuss this later).
- There are also alternatives to Euclidean distance...

Defining “Distance” with “Norms”

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

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

example *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”):

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

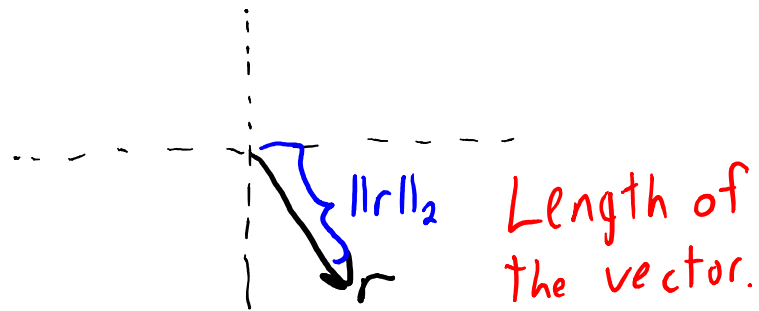
- Here, the “norm” of the difference is the standard Euclidean distance.

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:

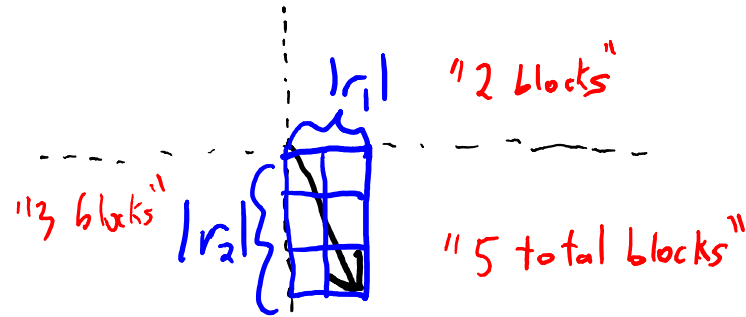
L₂ or "Euclidean" norm.

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



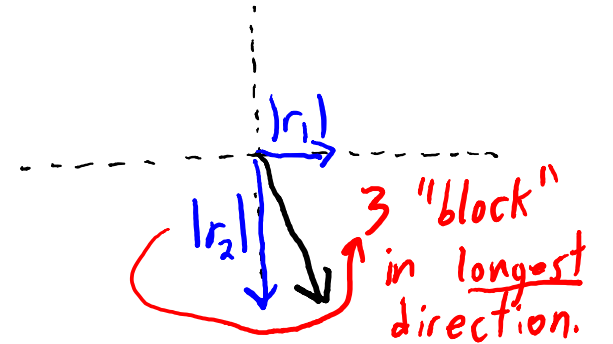
L₁ or "Manhattan" norm:

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



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:

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

$$\begin{aligned} \text{Notation: } \|r\|_2^2 &= (\|r\|_2)^2 \\ &= \left(\sqrt{\sum_{j=1}^d r_j^2}\right)^2 \\ &= \sum_{j=1}^d r_j^2 \\ &= r^T r \end{aligned}$$

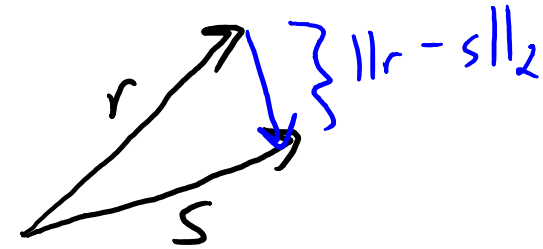
- These norms place different “weights” on large values:
 - L_1 : all values are equal.
 - L_2 : bigger values are more important (because of squaring).
 - L_∞ : only biggest value is important.

Different ways
to write the
same thing

Norms as Measures of Distance

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

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



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

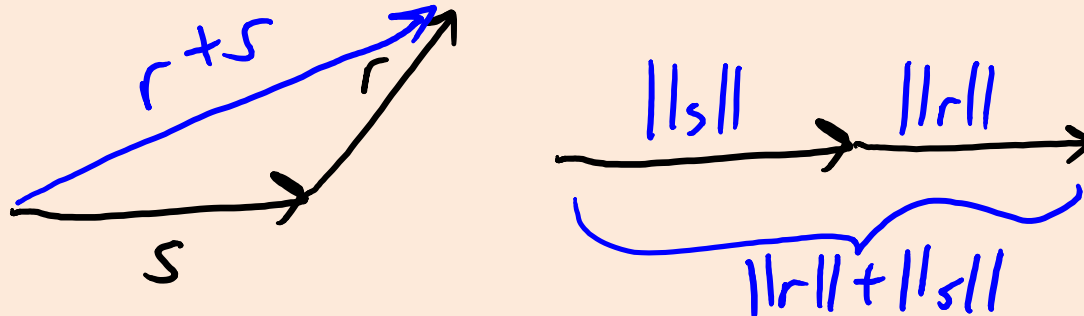
"Number of blocks you need to walk to get from r to s."

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

"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 be 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 Linf-norm.

- Weighted norms (if some features are more important):

- “Mahalanobis” distance (takes into account correlations).

$$\sum_{j=1}^d v_j |x_j|$$

↑ "weight" of feature j

- 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

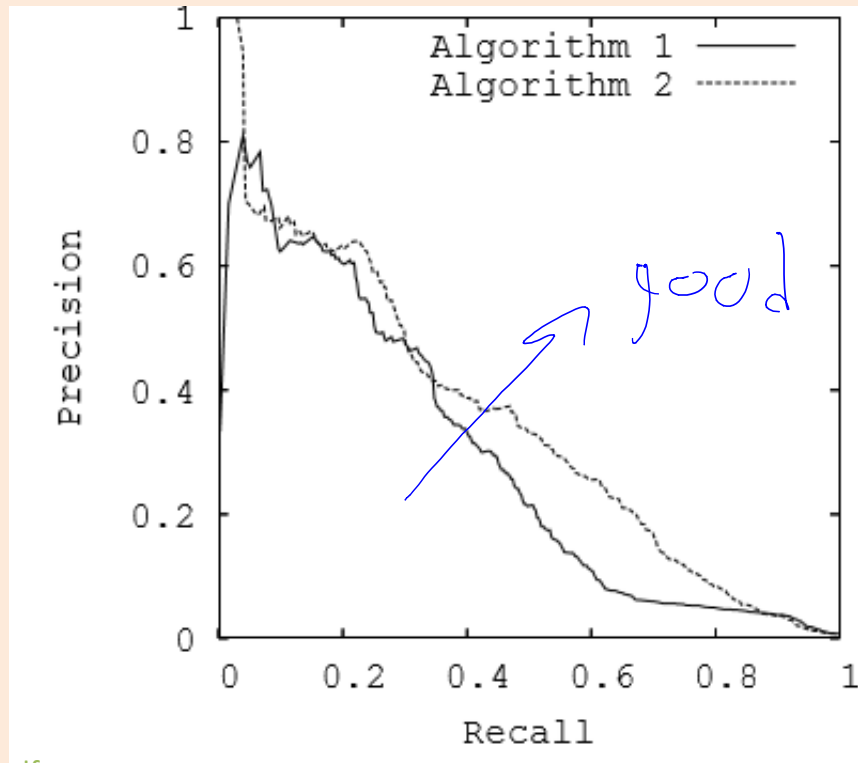
- **Decision theory** allows us to consider costs of predictions.
- **Non-parametric models** grow with number of training examples.
- **K-Nearest Neighbours**: simple non-parametric classifier.
 - Appealing “consistency” properties.
 - Suffers from high prediction cost and curse of dimensionality.
- **Next Time:**
 - Fighting the fundamental trade-off and Microsoft Kinect.

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.

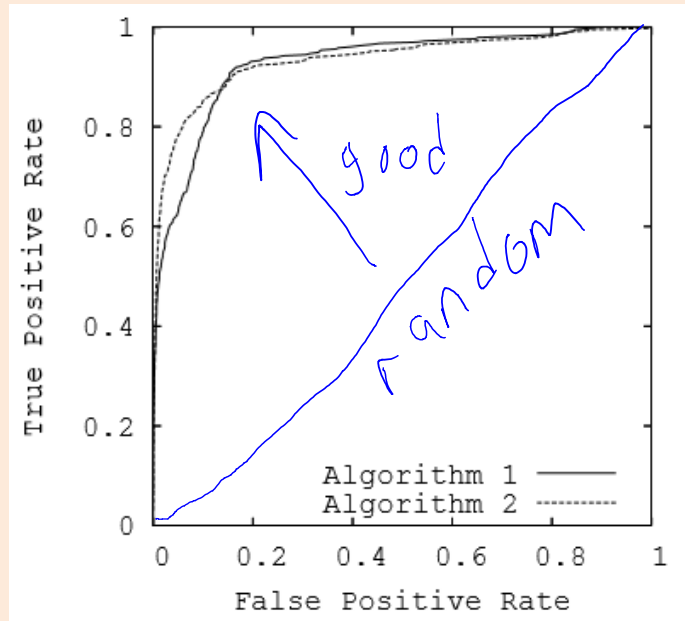
Precision-Recall Curve

- 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.



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:

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

where 'i' is a training example and 'j' is a test example.

- We can compute D in Julia using:

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

- 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 talking 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.