CPSC 340:
Machine Learning and Data Mining

Fundamentals of learning (continued)
and
the k-nearest neighbours classifier

Original version of these slides by Mark Schmidt, with modifications by Mike Gelbart.

Admin

- Assignment 1 is out:
  - Due Wednesday.
  - Fairly representative of workload in this course, but difficultly will increase.

- Add/drop deadline is Wednesday.
  - Good news: we may be expanding this section by a few seats… stay tuned.

Last Time: Training, Testing, and Validation

- Training step:

  Input: set of 'n' training examples x_i with labels y_i

  Output: a model that maps from arbitrary x_i to a y_i

- Prediction step:

  Input: set of 't' testing examples $\tilde{x}_i$ and a model.

  Output: predictions $\hat{y}_i$ for the testing examples.

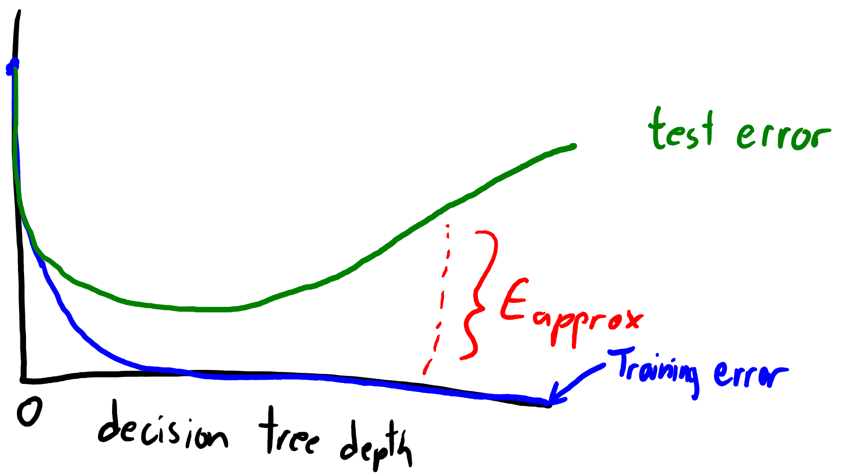
- What we are interested in is the test error:
  - Error made by prediction step on new data.

Last Time: Fundamental Trade-Off

- We decomposed test error to get a fundamental trade-off:

$$\underbrace{E_{test}}_{\text{"test error"}} = \underbrace{E_{approx}}_{\text{"approximation error"}} + \underbrace{E_{train}}_{\text{"training error"}}$$

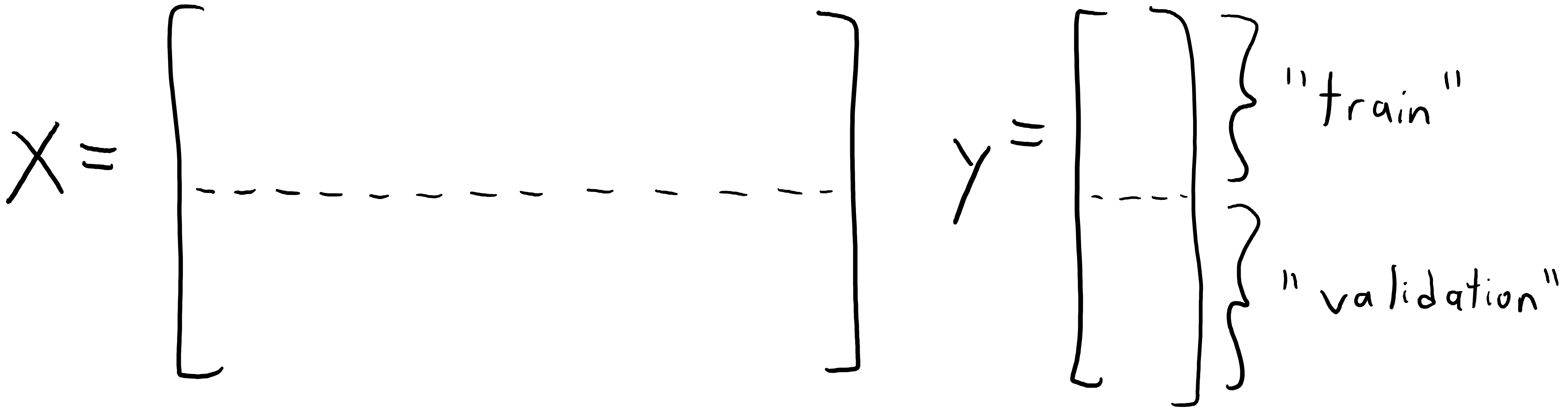
  – Where $E_{approx} = (E_{test} - E_{train})$.



- E_{train} goes down as model gets complicated:
  – Training error goes down as a decision tree gets deeper.
- But $E_{approxr}$ goes up as model gets complicated:
  – Training error becomes a worse approximation of test error.

Last Time: Validation Error

- Golden rule: we can't look at test data during training.
- But we can approximate E_{test} with a validation error:
  - Error on a set of training examples we "hid" during training.



  - Find the decision tree based on the "train" rows.
  - Validation error is the error of the decision tree on the "validation" rows.

Notation: Parameters and Hyperparameters

- The decision tree rule values are called "parameters".
  - Parameters control how well we fit a dataset.
  - We "train" a model by trying to find the best parameters on training data.

- The decision tree depth is a called a "hyperparameter".
  - Hyper-parameters control how complex our model is.
  - We can't "train" a hyperparameter.
    - You can always fit training data better by making the model more complicated.
  - We "validate" a hyperparameter using a validation score.

Choosing Hyper-Parameters with Validation Set

- So to choose a good value of depth (“hyperparameter”), we could:
  - Try a depth-1 decision tree, compute validation error.
  - Try a depth-2 decision tree, compute validation error.
  - Try a depth-3 decision tree, compute validation error.
  - …
  - Try a depth-20 decision tree, compute validation error.
  - Return the depth with the lowest validation error.

- After you choose the hyper-parameter, we usually re-train on the full training set with the chosen hyper-parameter.

Choosing Hyper-Parameters with Validation Set

- This leads to much less overfitting than using the training error.
  - We optimize the validation error over 20 values of “depth”.
  - Unlike training error, where we optimize over tons of decision trees.

- But it can still overfit (very common in practice):
  - Validation error is only an unbiased approximation if you use it once.
  - If you minimize it to choose a model, introduces optimization bias:
    - If you try lots of models, one might get a low validation error by chance.

- Remember, our goal is still to do well on the test set (new data), not the validation set (where we already know the labels).

Should you trust them?

- Scenario 1:
  - "I built a model based on the data you gave me."
  - "It classified your data with 98% accuracy."
  - "It should get 98% accuracy on the rest of your data."

- Probably not:
  - They are reporting training error.
  - This might have nothing to do with test error.
  - E.g., they could have fit a very deep decision tree.
- Why 'probably'?
  - If they only tried a few very simple models, the 98% might be reliable.
  - E.g., they only considered decision stumps with simple 1-variable rules.

Should you trust them?

- Scenario 2:
  - "I built a model based on half of the data you gave me."
  - "It classified the other half of the data with 98% accuracy."
  - "It should get 98% accuracy on the rest of your data."
- Probably:
  - They computed the validation error once.
  - This is an unbiased approximation of the test error.
  - Trust them if you believe they didn't violate the golden rule.

Should you trust them?

- Scenario 3:
  - "I built 10 models based on half of the data you gave me."
  - "One of them classified the other half of the data with 98% accuracy."
  - "It should get 98% accuracy on the rest of your data."

- Probably:
  - They computed the validation error a small number of times.
  - Maximizing over these errors is a biased approximation of test error.
  - But they only maximized it over 10 models, so bias is probably small.
  - They probably know about the golden rule.

Should you trust them?

- Scenario 4:
  - "I built 1 billion models based on half of the data you gave me."
  - "One of them classified the other half of the data with 98% accuracy."
  - "It should get 98% accuracy on the rest of your data."

- Probably not:
  - They computed the validation error a huge number of times.
  - Maximizing over these errors is a biased approximation of test error.
  - They tried so many models, one of them is likely to work by chance.
- Why 'probably'?
  - If the 1 billion models were all extremely-simple, 98% might be reliable.

Should you trust them?

- Scenario 5:
  - "I built 1 billion models based on the first third of the data you gave me."
  - "One of them classified the second third of the data with 98% accuracy."
  - "It also classified the last third of the data with 98% accuracy."
  - "It should get 98% accuracy on the rest of your data."

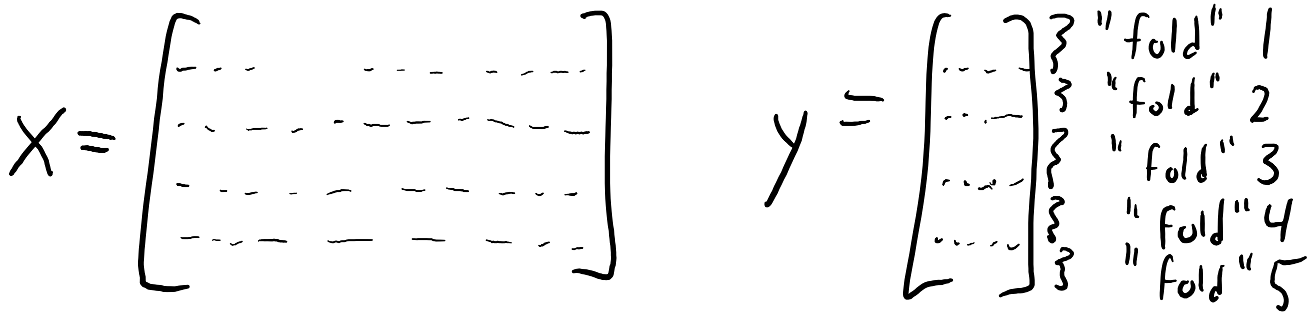
- Probably:
  - They computed the first validation error a huge number of times.
  - But they had a second validation set that they only looked at once.
  - The second validation set gives unbiased test error approximation.
  - This is ideal, as long as they didn't violate golden rule on the last third.
  - And assuming you are using IID data in the first place.

Validation Error and Optimization Bias

- Optimization bias is small if you only compare a few models:
  - Best decision tree on the training set among depths, 1, 2, 3,…, 10.
  - Risk of overfitting to validation set is low if we try 10 things.
- Optimization bias is large if you compare a lot of models:
  - All possible decision trees of depth 10 or less.
  - Here we're using the validation set to pick between a billion+ models:
    - Risk of overfitting to validation set is high: could have low validation error by chance.
  - If you did this, you might want a second validation set to detect overfitting.

Cross-Validation (CV)

- Isn't it wasteful to only use part of your data?
- 5-fold cross-validation:
  - Train on 80% of the data, validate on the other 20%.
  - Repeat this 5 more times with different splits, and average the score.



1. Train on folds {1, 2, 3, 4}, compute error on fold 5.
2. Train on folds {1, 2, 3, 5}, compute error on fold 4.
3. Train on folds {1, 2, 4, 5}, compute error on fold 3.

⋮

6. Take average of the 5 errors as approximation of test error

Cross-Validation (CV)

- You can take this idea further:
  - 10-fold cross-validation: train on 90% of data and validate on 10%.
    - Repeat 10 times and average.
  - Leave-one-out cross-validation: train on all but one training example.
    - Repeat n times and average.
    - This is the same as n-fold cross validation.
- Gets more accurate but more expensive with more folds.
  - To choose depth we compute the cross-validation score for each depth.
- As before, if data is ordered then folds should be random splits.
  - Randomize first, then split into fixed folds.

(pause)

The “Best” Machine Learning Model

- Decision trees are not always most accurate on test error.
- What is the “best” machine learning model?
- First we need to define generalization error:
  - Test error restricted to new feature combinations (no x_i from train set).
- No free lunch theorem:
  - There is **no** “best” model achieving the best generalization error for every problem.
  - If model A generalizes better to new data than model B on one dataset, there is another dataset where model B works better.
- This question is like asking which is “best” among “rock”, “paper”, and “scissors”.

The “Best” Machine Learning Model

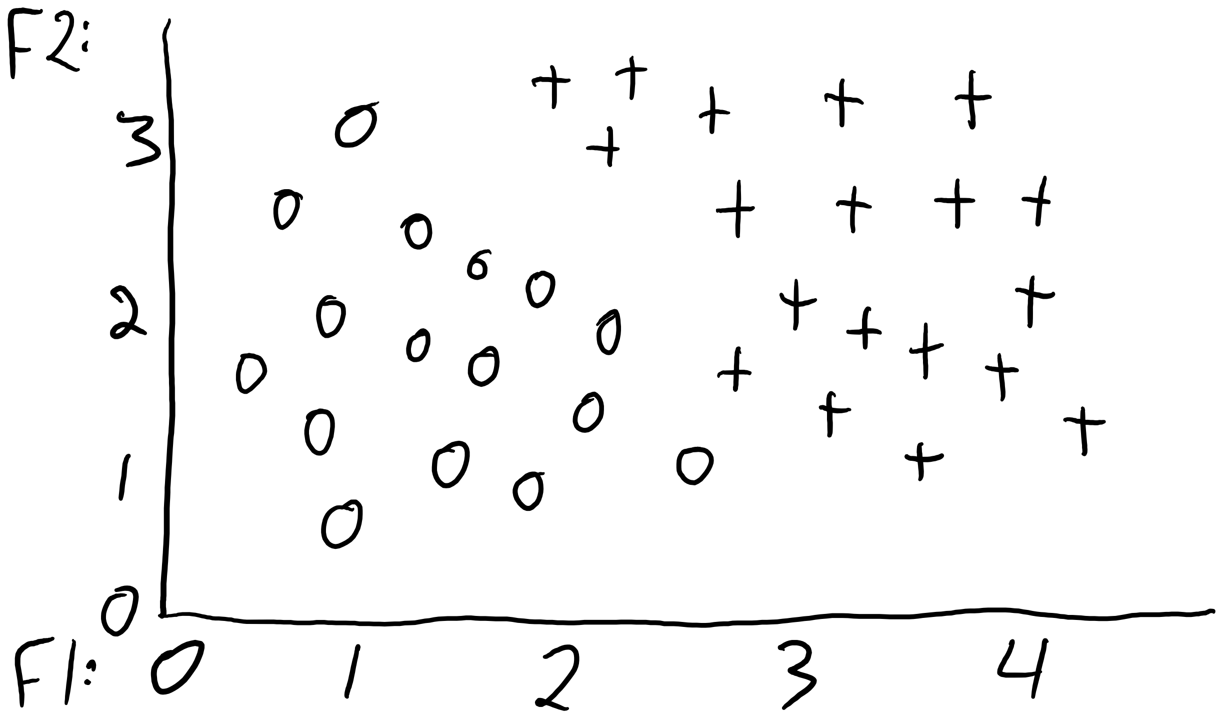
- Implications of the lack of a “best” model:
  - We need to learn about and try out multiple models.
- So which ones to study in CPSC 340?
  - We’ll usually motivate each method by a specific application.
  - But we’re focusing on models that have been effective in many applications.

- Caveat of no free lunch (NFL) theorem:
  - The world is very structured.
  - Some datasets are more likely than others.
  - Model A really could be better than model B on every real dataset in practice.
- Machine learning research:
  - Large focus on models that are useful across many applications.

K-Nearest Neighbours (KNN)

- To classify an object $\tilde{x}_i$:
  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.

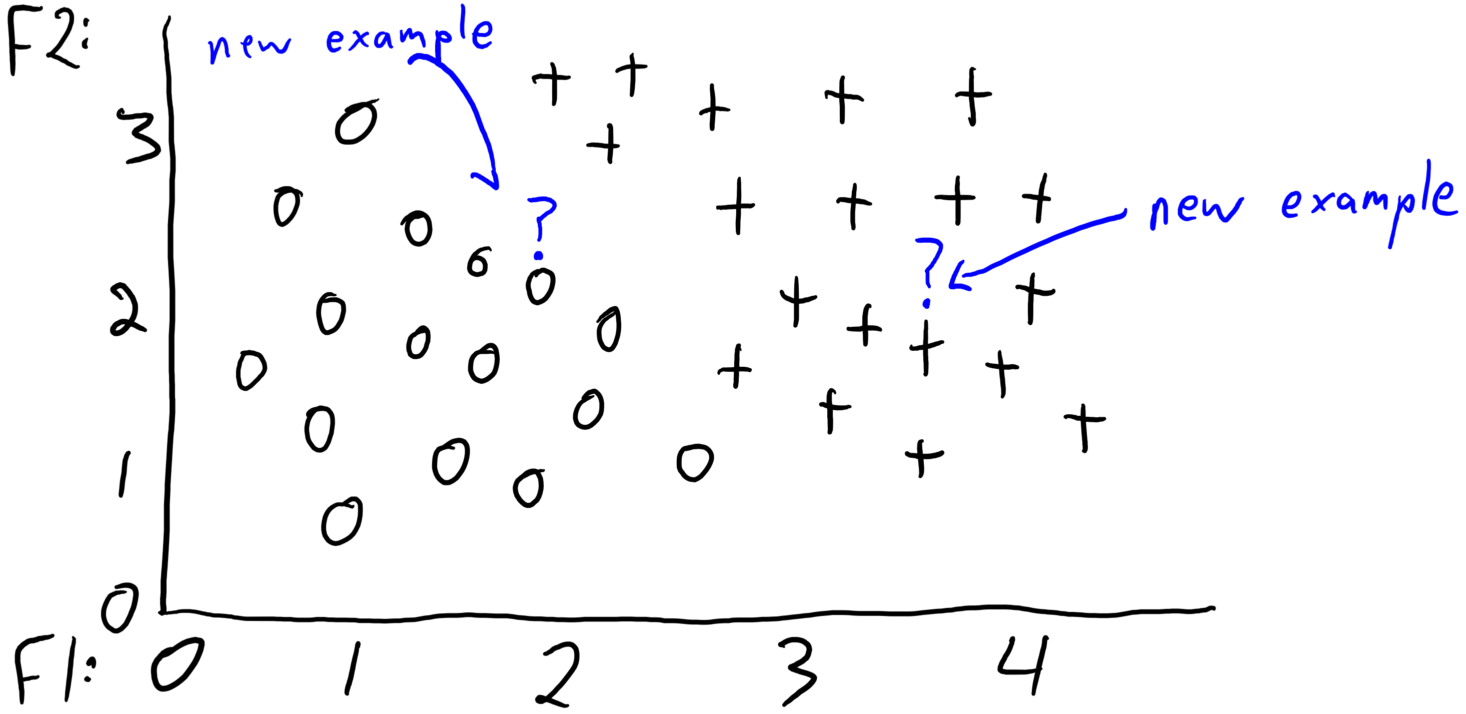
F1	F2		Label
1	3	→	O
2	3	→	+
3	2	→	+
2.5	1	→	O
3.5	1	→	+
…	…		…



K-Nearest Neighbours (KNN)

- To classify an object $\tilde{x}_i$:
  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	O
2	3	+
3	2	+
2.5	1	O
3.5	1	+
…	…	…

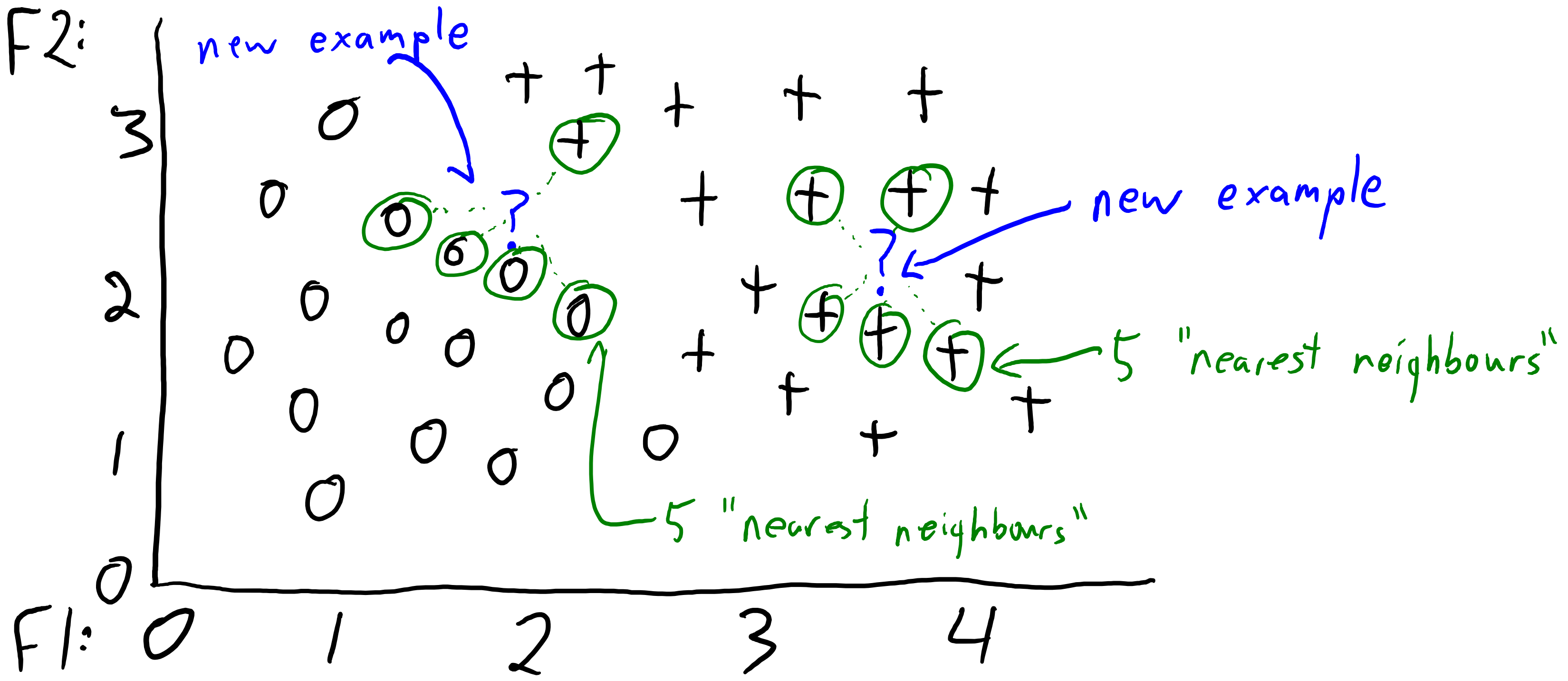


K-Nearest Neighbours (KNN)

- To classify an object $\tilde{x}_i$:
  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
1	3
2	3
3	2
2.5	1
3.5	1
…	…

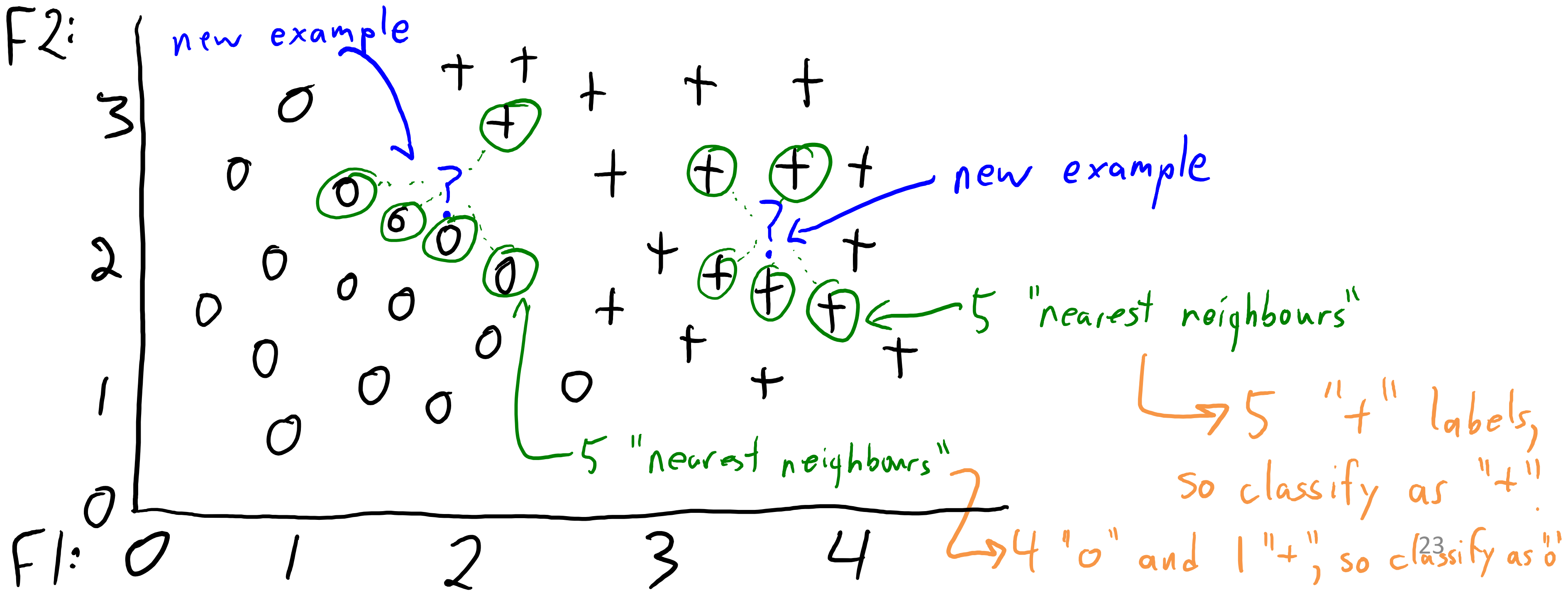
Label
O
+
+
O
+
…



K-Nearest Neighbours (KNN)

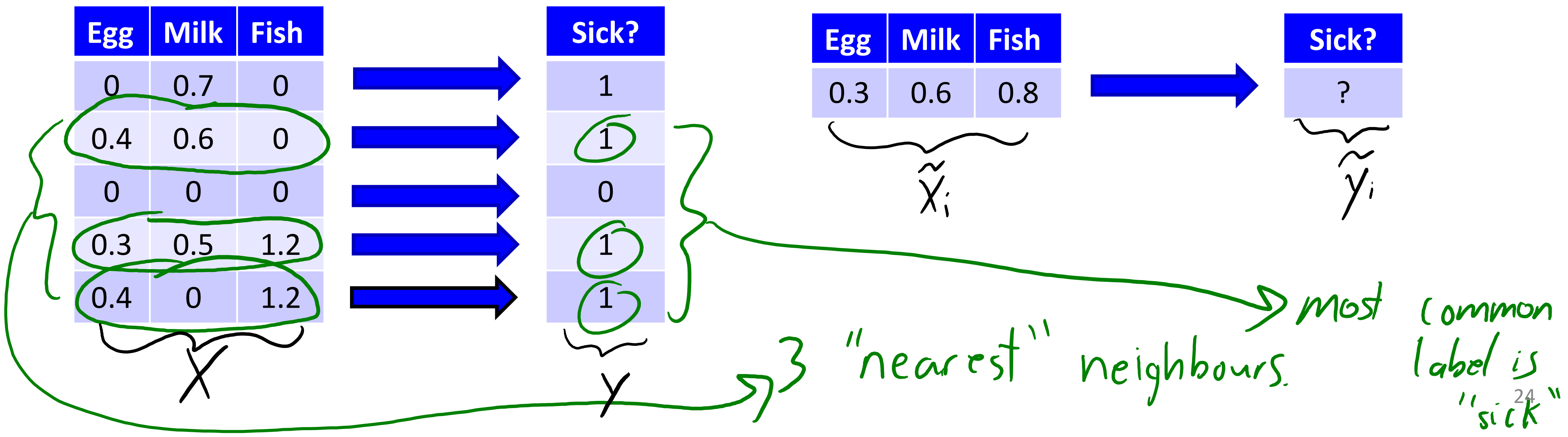
- To classify an object $\tilde{x}_i$:
  1. Find the ‘k’ training examples x_i that are “nearest” to $\tilde{x}_i$.
  2. Classify using the most common label of “nearest” examples.

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



K-Nearest Neighbours (KNN)

- To classify an object $\tilde{x}_i$:
  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.



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?
?

K-Nearest Neighbours (KNN)

- Most common distance function is Euclidean distance:

$$d(v,w) = \sqrt{\sum_{j=1}^{d}(v_j - w_j)^2}$$

  – Compute this distance between a test point and **all** training points.

- Assumption:
  – Objects with similar features likely have similar labels.

KNN Implementation

- There is no training phase in KNN (“lazy” learning).
  - You just store the training data.
- But predictions are expensive: O(nd) to classify 1 test object.
  - Tons of work on reducing this cost.
- There are also alternatives to Euclidean distance.

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.
    - You might not have any training points "near" a test point.
- KNN is also problematic if features have very different scales.
  - A feature with a big scale can dominate all the distances
  - A feature with a small scale can be neglected
- Nevertheless, KNN is really easy to use and often hard to beat!

Parametric vs. Non-Parametric

- Parametric models:
  - Have a fixed number of parameters: size of "model" is O(1) in terms 'n'.
    - E.g., fixed-depth decision tree just stores rules.
  - 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'.
    - E.g., with KNN we need to store O(nd) information.
  - Model gets more complicated as you get more data.
- (IMO decision trees are an ambiguous case, but it's usually clear.)

Non-parametric models

- With a small ‘n’, KNN model will be very simple.
- Model gets more complicated as ‘n’ increases.
  - Starts to detect subtle differences between examples.
- We say “the complexity grows with the amount of data”.

Norms (abridged)

- The notation ||x|| refers to the norm (like the size) of a vector x.

$$||x||_2^2 = \sum_{i=1}^{n} x_i^2$$

- The 2 in the subscript is the type of norm: “L2 norm”
  - The L1 norm is the sum of the absolute valued.
- The 2 in the superscript is just regular squaring
- A norm operates on ONE vector
- A distance function operates on TWO vectors, e.g. d(x,y)
- However, we can represent distances as norms, as in

$$d_{Euclidean}(x,y) = ||x-y||_2 = ||x-y||$$

- Later in the course we’ll see other types of norms, like L1, L0, etc.
  - Surprisingly, some of the key ideas in this course pertain to changing norm types.

Summary

- Hyperparameters: high-level choices that control model complexity
  - E.g., tree depth for decision trees, ‘k’ for KNN
- Optimization bias: unwittingly overfitting your validation set
- Cross-validation: many train/validation splits from one data set
  - More accurate but requires training more models (slower)
- K-Nearest Neighbours: simple non-parametric classifier.
  - Appealing “consistency” properties.
  - Suffers from high prediction cost and curse of dimensionality.
- Non-parametric models grow with number of training examples.
- Norms measure the size of a vector (“distance from the origin”).

Back to Decision Trees

- Instead of validation set, you can use CV to select tree depth.

- But you can also use these to decide whether to split:
  - Don’t split if validation/CV error doesn’t improve.
  - Different parts of the tree will have different depths.
- Or fit deep decision tree and use CV to prune:
  - Remove leaf nodes that don’t improve CV error.

- Popular implementations that have these tricks and others.

Cross-Validation Theory

- Does CV give unbiased estimate of test error?
  - Yes!
    - Since each data point is only used once in validation, expected validation error on each data point is test error.
  - But again, if you CV to select among models then it is no longer unbiased.

- What about variance of CV?
  - Hard to characterize.
  - CV variance on ‘n’ data points is worse than with a validation set of size ‘n’.
    - But we believe it is close.

KNN Distance Functions

- Most common KNN distance functions: norm($x_i - x_j$).
  - L1-, L2-, and Linf-norm.
  - Weighted norms (if some features are more important): $\sum_{j=1}^{d} v_j |x_j|$
  - "Mahalanobis" distance (takes into account correlations).

$\uparrow$ "weight" of feature 'j'

- But we can consider other distance/similarity functions:
  - Hamming distance.
  - Jaccard similarity (if x_i are sets).
  - Edit distance (if x_i are strings).
  - Metric learning (*learn* the best distance function).

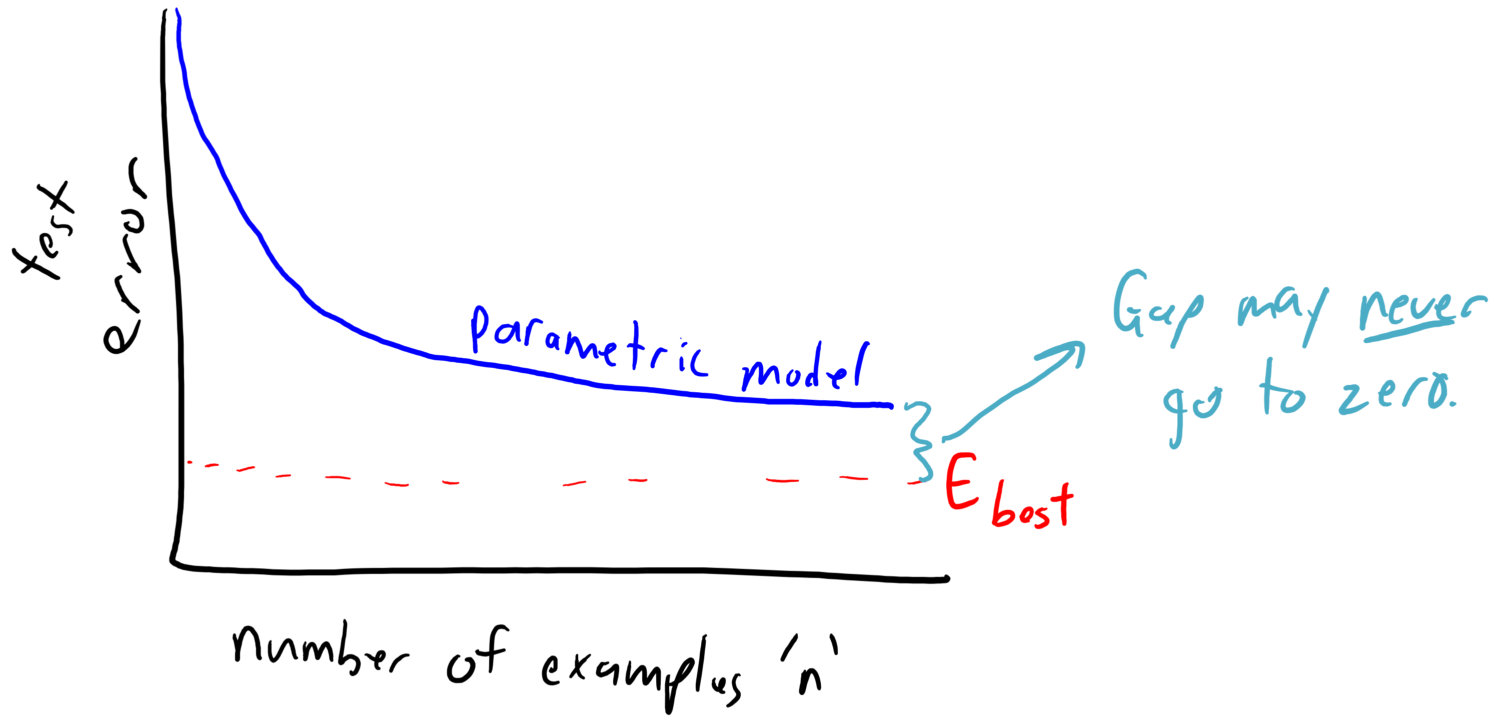
Consistency of KNN

- 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).

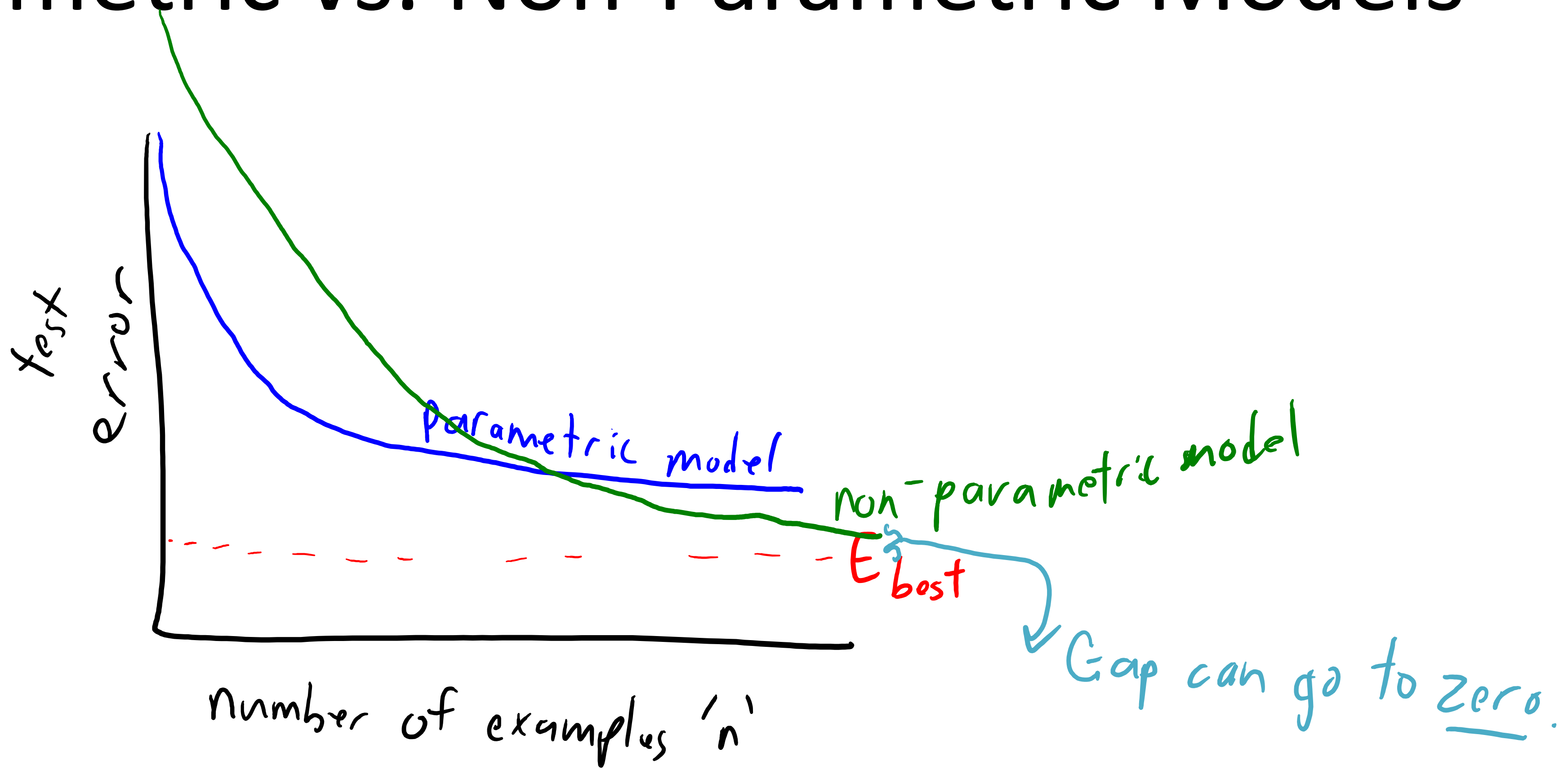
- Stone's Theorem: KNN is "universally consistent".
  - If k/n goes to zero and 'k' goes to ∞, converges to the best possible error.
    - First algorithm shown to have this property.

- 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



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:

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

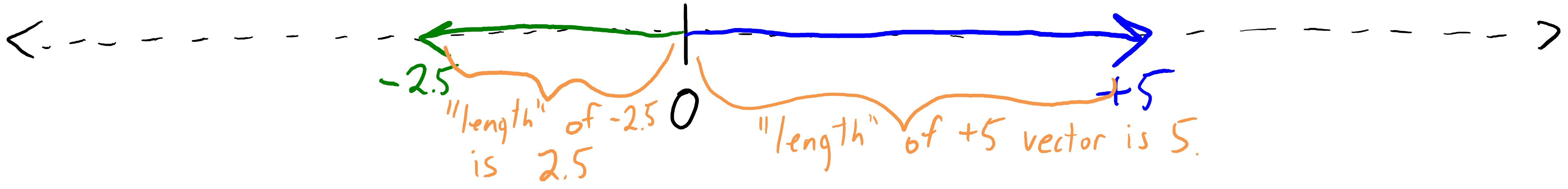
    where ‘i’ is a training example and ‘j’ is a test example.
  - We can compute D in Julia using:

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

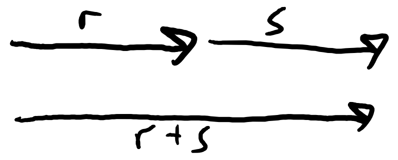
  - And you get an extra boost because Julia uses multiple cores.
    - Something similar exists in Python

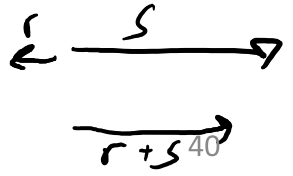
Norms in 1-Dimension

- We can view absolute value, |r|, as 'size' or 'length' of a number 'r':



- It satisfies three intuitive properties of 'length':
  1. Only '0' has a 'length' of zero.
  2. Multiplying 'r' by constant 'α' multiplies length by |α|: |αr| = |α||r|.
     - "If be will twice as long if you multiply by 2".
  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| <= |r| + |s|.





Norms in 2-Dimensions

- In 1-dimension, only scaled absolute values satisfy the 3 properties.
- In 2-dimensions, there is no unique function satisfying them.
- We call any function satisfying them a norm:
  - Measures of "size" or "length" in 2-dimensions.
- Three most common examples:

L_2 or "Euclidean" norm.

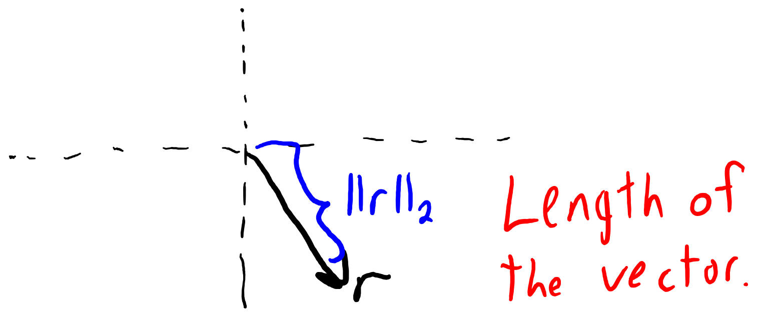
$$\|r\|_2 = \sqrt{r_1^2 + r_2^2}$$

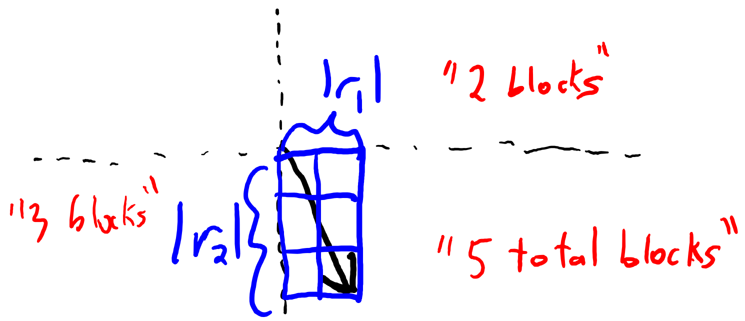
L_1 or "Manhattan" norm:

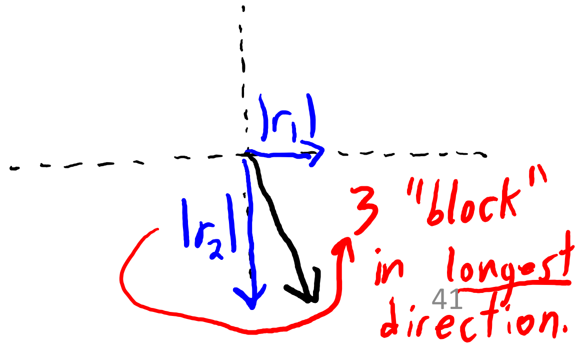
$$\|r\|_1 = |r_1| + |r_2|$$

L_∞ or "max" norm:

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





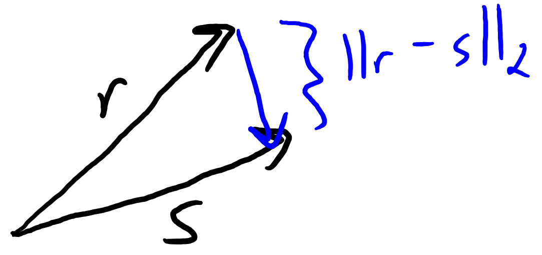


Norms as Measures of Distance

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

$$\|r-s\|_2 = \sqrt{(r_1-s_1)^2+(r_2-s_2)^2}$$

$= \|r-s\|$ "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."

Infinite Series Video

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_∞: $\max_j \{|r_j|\}$

E.g., in 3-dimensions:

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

in 4-dimensions:

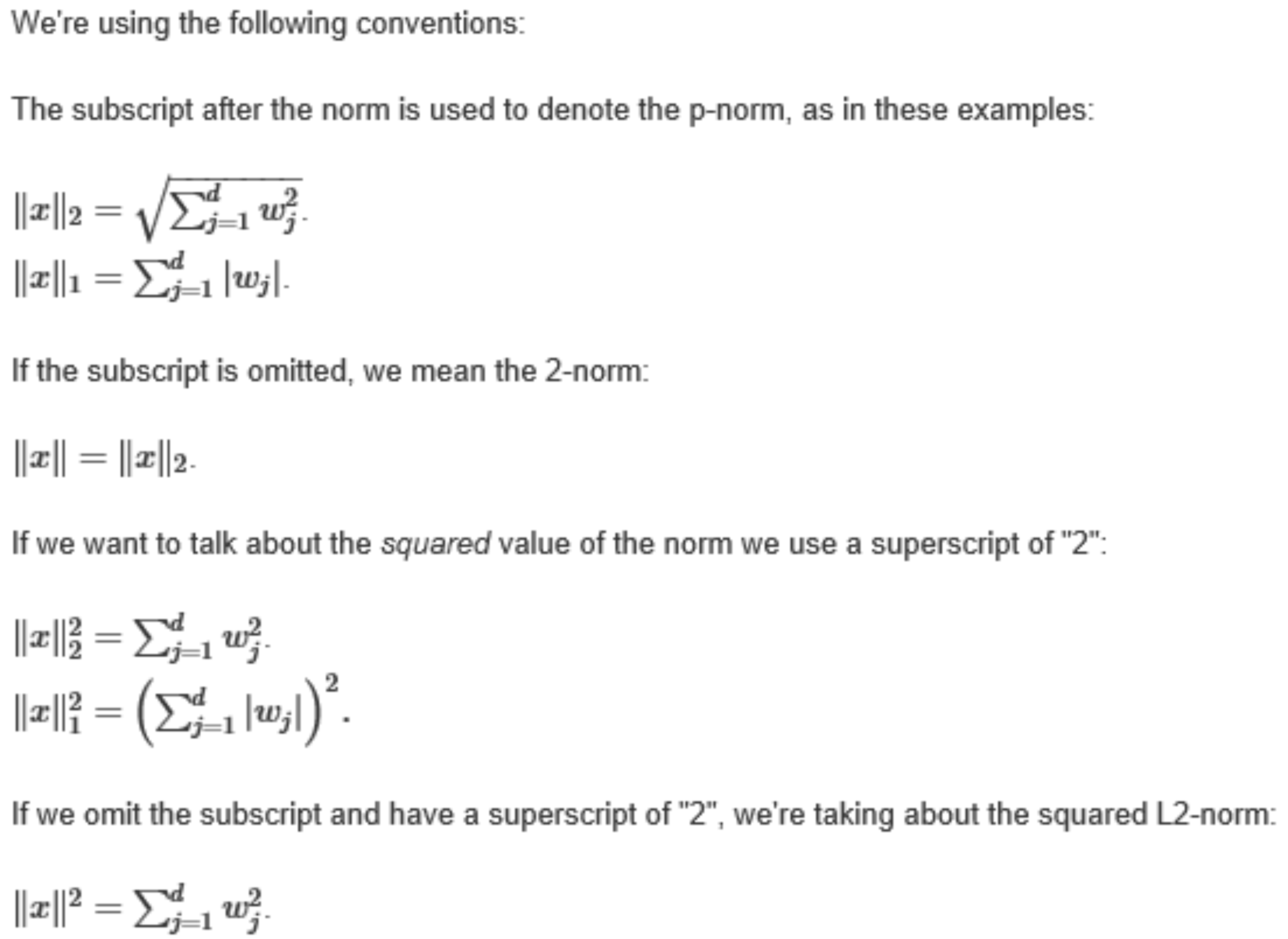
$$\|r\|_2 = \sqrt{r_1^2 + r_2^2 + r_3^2 + r_4^2}$$

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

Different ways to write the same thing

- 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.

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:

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

Lp-norms

- The L_1-, L_2-, and L_∞-norms are special cases of Lp-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.