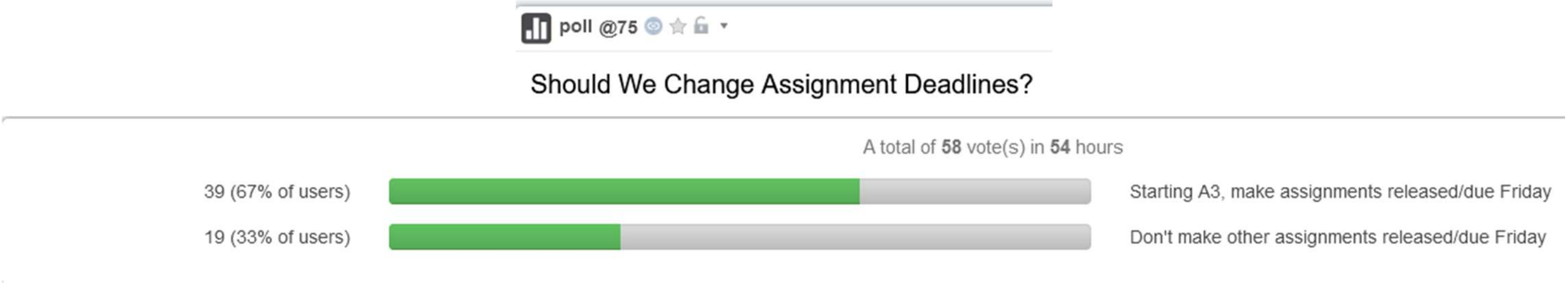
CPSC 340: Machine Learning and Data Mining

Data Augmentation

Summer 2021

Admin

- Assignment 2 is out.
  - Due Monday of next week.
- Assignment 3 is out Friday
- Midterm in Tuesday, June 1, 2021



poll @75

Should We Change Assignment Deadlines?

A total of **58** vote(s) in **54** hours

39 (67% of users) — Starting A3, make assignments released/due Friday

19 (33% of users) — Don't make other assignments released/due Friday

Notes on Programming

- Remember: this is a 300-level computer science course.
  - I'm assuming that you know how to:
    - Associate plainly described algorithms to lines we wrote in Python
    - Debug and test your code
  - Or that you can pick up these skills as you go

- Please ask more efficient programming questions:
  - Bad: "Here's my code and it doesn't work."
  - Good: "Here's my code, and output of each variable in it.
    My understanding of the algorithm is <blah>. Where did I go wrong?"

Ask “Upstream” Questions

- Assumption: if your understanding is correct and your logic is good, then assignment questions should be straightforward.

- Don’t ask: “Here’s my solution. Is this correct?”
- Ask: “I understand that <blah>, and I’m following this logic: <bleh>. Am I going in the right direction?”

Live Demo:

1. ipdb
2. using a runbook

Pro Tip: Keep a “Runbook”

- Python is general-purpose scripting language
  - Different from Java or TypeScript

- We’ll be using main.py as if it’s a function on its own
  - Has an argument called “q” and we specify question number

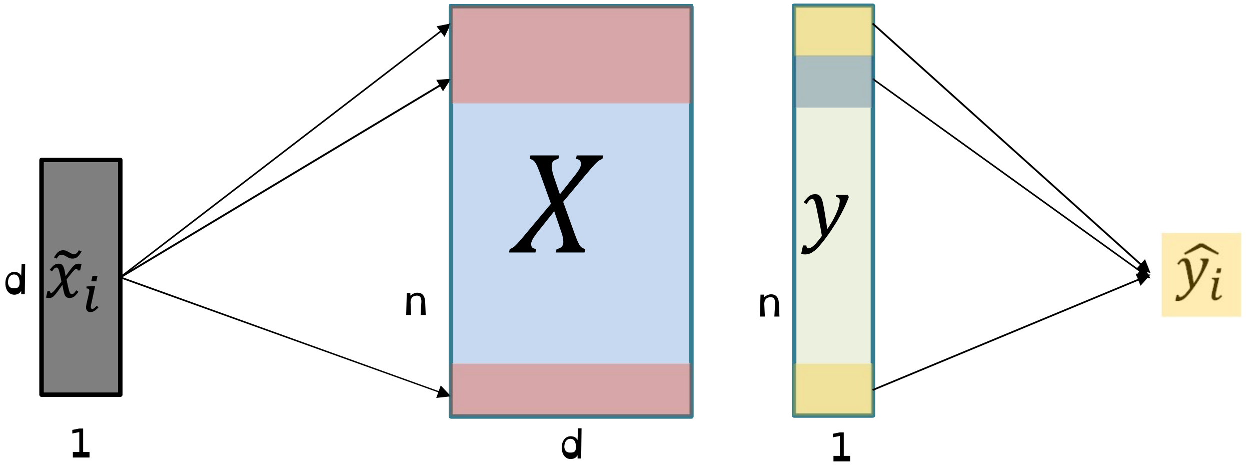
- Useful to copy-and-paste commands from somewhere instead of having to remember exact commands
  - Optionally, debug commands (not really needed if using IDE and setting up run configs)

In This Lecture

- Nonparametric Models (15 minutes)
- Data Augmentation (20 minutes)
- Ensemble Methods (15 minutes)

Last Time: K-Nearest Neighbours

- 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.



L2-norm, L1-norm, and L∞-Norms.

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

L_2 or "Euclidean" norm.

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

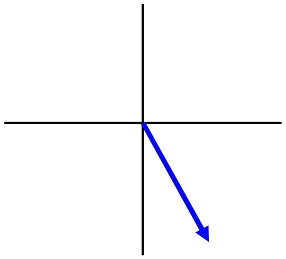
L_1 or "Manhattan" norm:

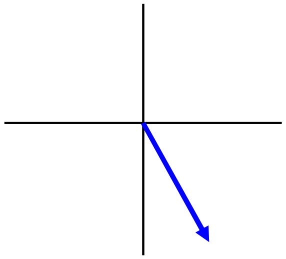
$$\|r\|_1 = |r_1| + |r_2|$$

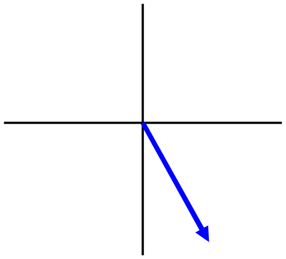
L_∞ or "max" norm:

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

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







  - Definitions of these norms in d-dimensions.

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

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

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

Infinite Series Video

KNN Distance Functions

- Most common KNN distance functions: norm($x_i - x_j$).
  - L1-, L2-, and L∞-norm.
  - Weighted norms (if some features are more important):

    $$\sum_{j=1}^{d} \underbrace{v_j}_{\text{"weight" of feature 'j'}} |x_j|$$

  - "Mahalanobis" distance (incorporate correlations).
    - See bonus slide for what functions define a "norm".

- 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).

When you train KNN

Coming Up Next

NON-PARAMETRIC MODELS

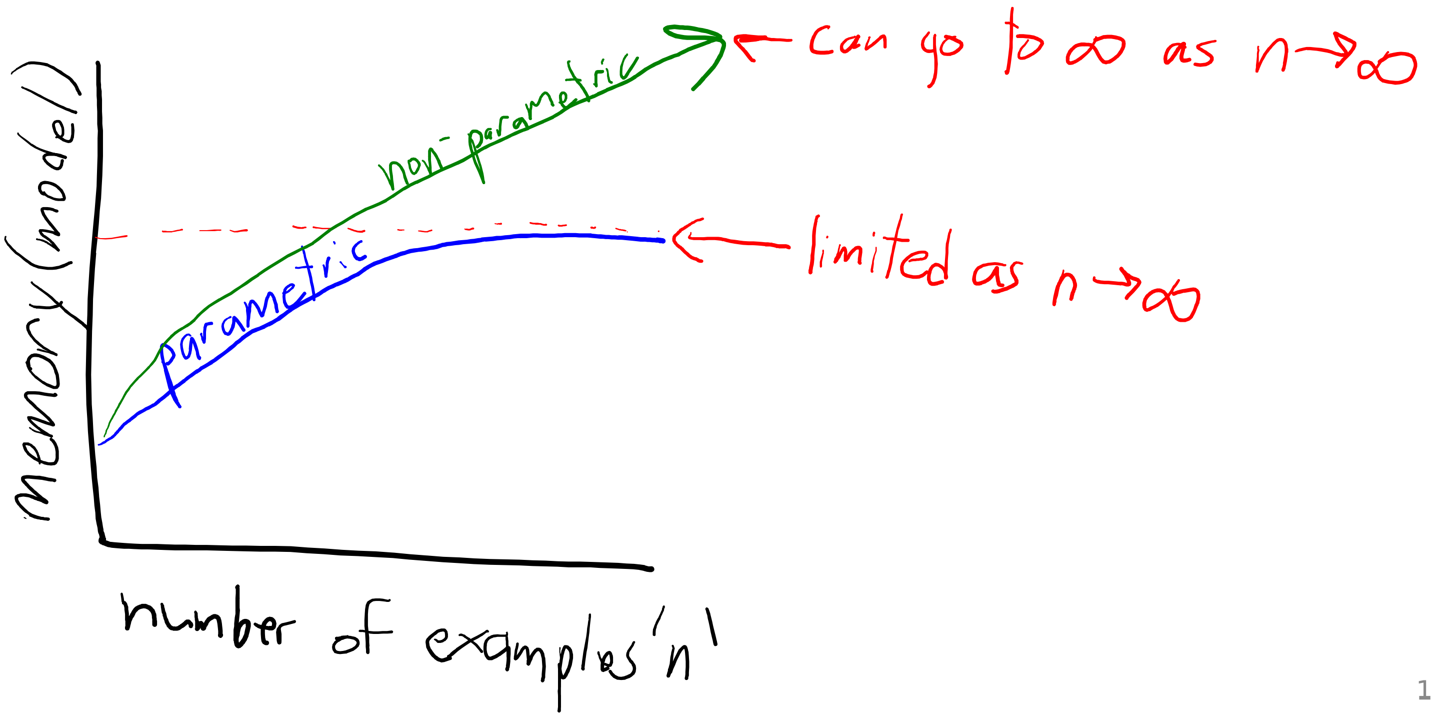
Parametric vs. Non-Parametric

- Parametric models:
  - Have fixed number of parameters: trained "model" size is O(1) in terms 'n'.
    - E.g., naïve Bayes just stores counts.
    - E.g., fixed-depth decision tree just stores rules for that depth.
  - You can estimate the fixed parameters more accurately with more data.
  - But eventually more data doesn't help: model is too simple.

- Non-parametric models:
  - Number of parameters grows with 'n': size of "model" depends on 'n'.
  - Model gets more complicated as you get more data.
    - E.g., KNN stores all the training data, so size of "model" is O(nd).
    - E.g., decision tree whose depth *grows with the number of examples*.

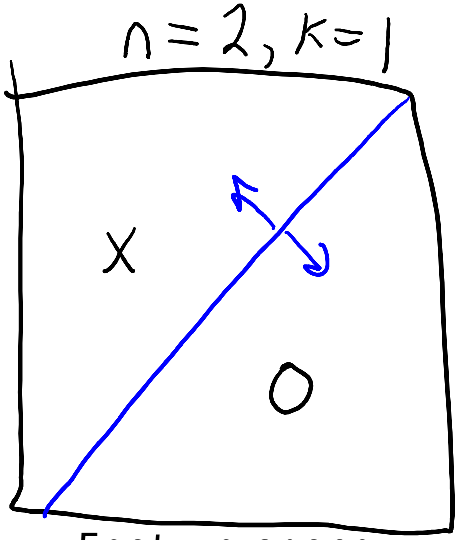
Parametric vs. Non-Parametric Models

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

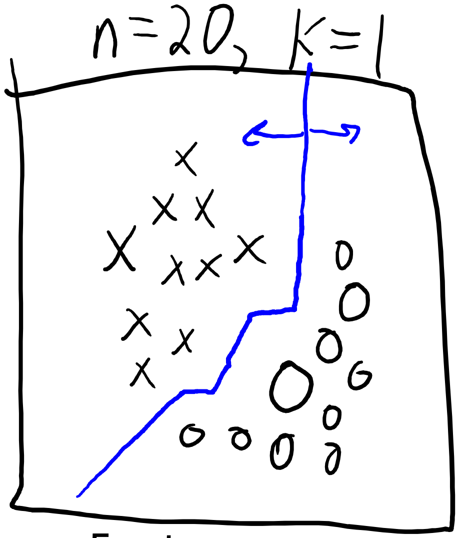


Effect of ‘n’ in KNN.

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



Feature space



Feature space

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

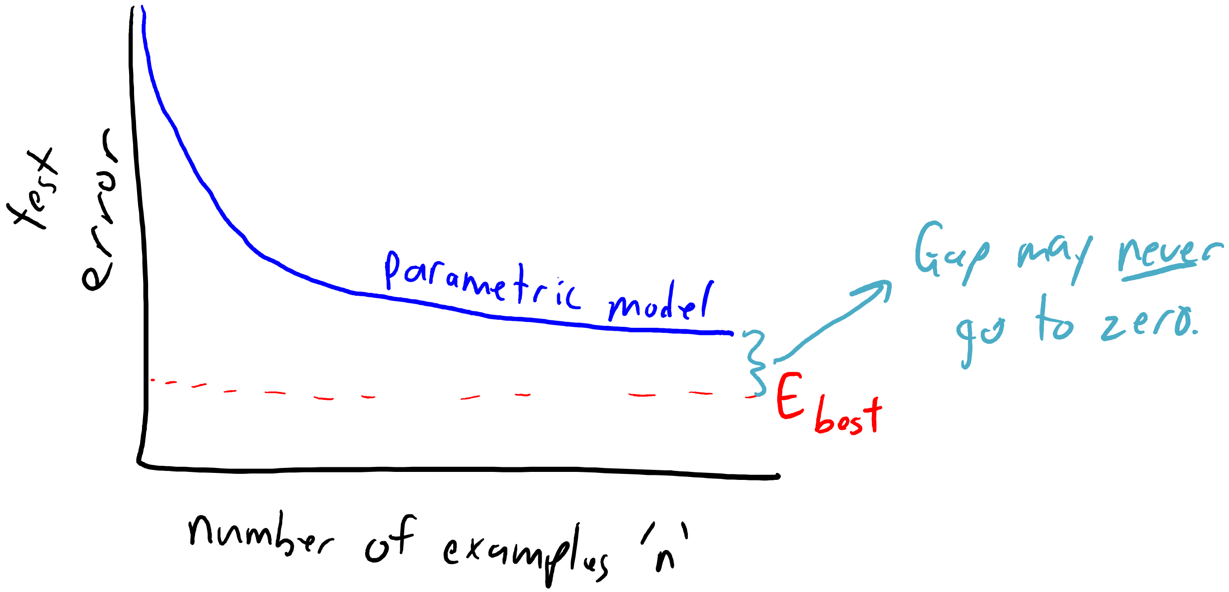
Q: Does that mean we overfit with large n?

Consistency of KNN (‘n’ going to ‘∞’)

- KNN has appealing consistency properties:
  - As ‘n’ goes to ∞, KNN $E_{test} < 2 * E_{best}$.
    - E_{best} := best test error possible
    - 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 E_{best}.
    - For example, k = log(n).
    - First algorithm shown to have this property.
- Does Stone’s Theorem violate the no free lunch theorem?
  - No: it requires a continuity assumption on the labels.
  - Consistency says nothing about finite ‘n’ (see "Dont Trust Asymptotics”).

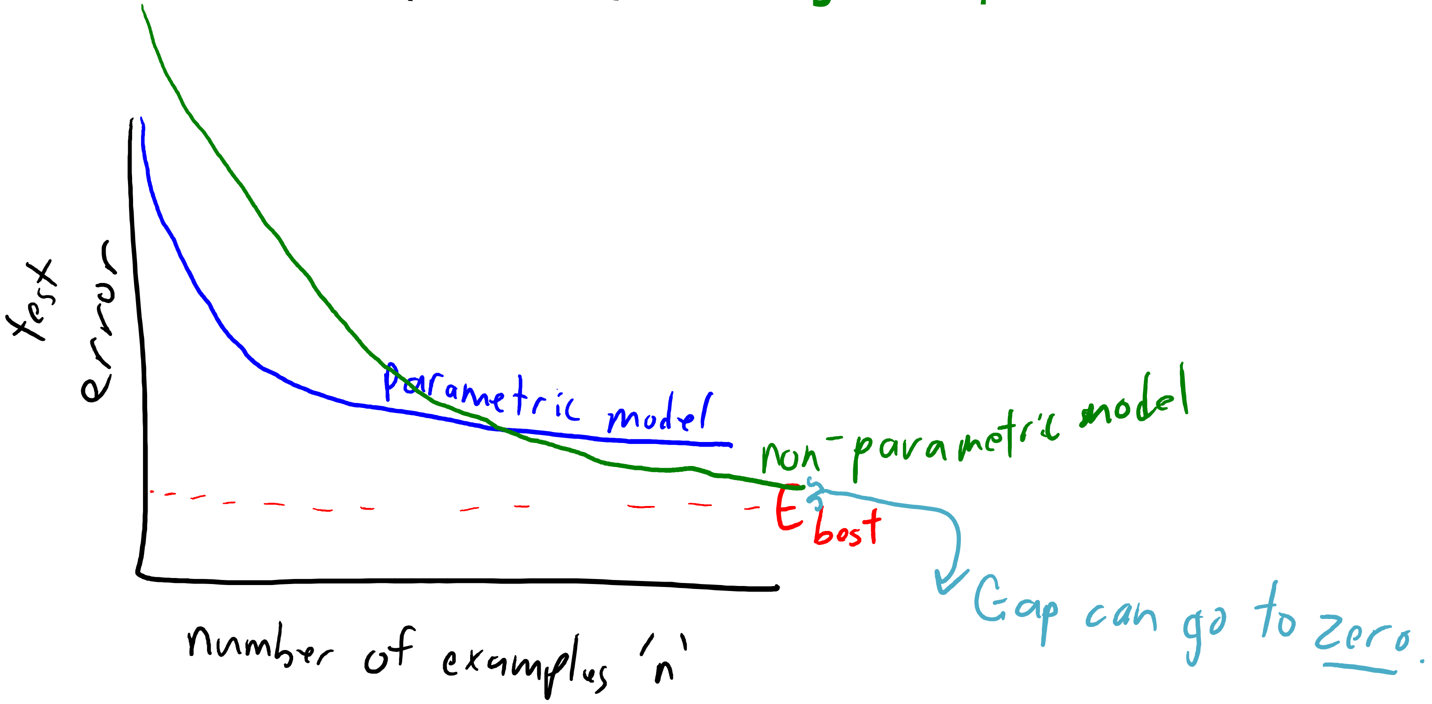
Parametric vs. Non-Parametric Models

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



Parametric vs. Non-Parametric Models

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

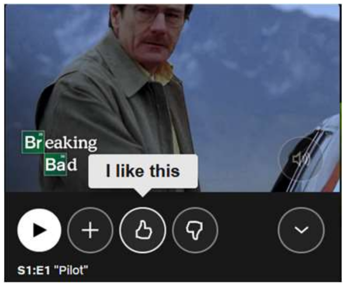


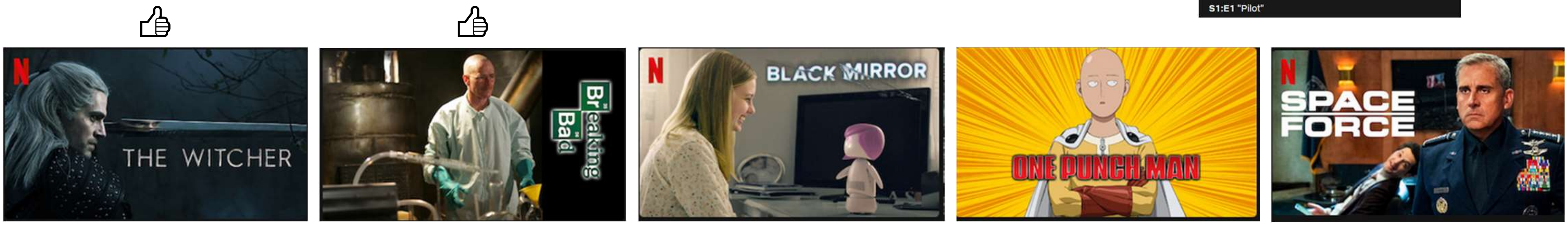
Coming Up Next

CURSE OF DIMENSIONALITY

Application: Netflix Show Recommendation

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





All shows on Netflix

Should I recommend this show?

Application: Netflix Show Recommendation

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

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

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

Your preference	1	1	0	0

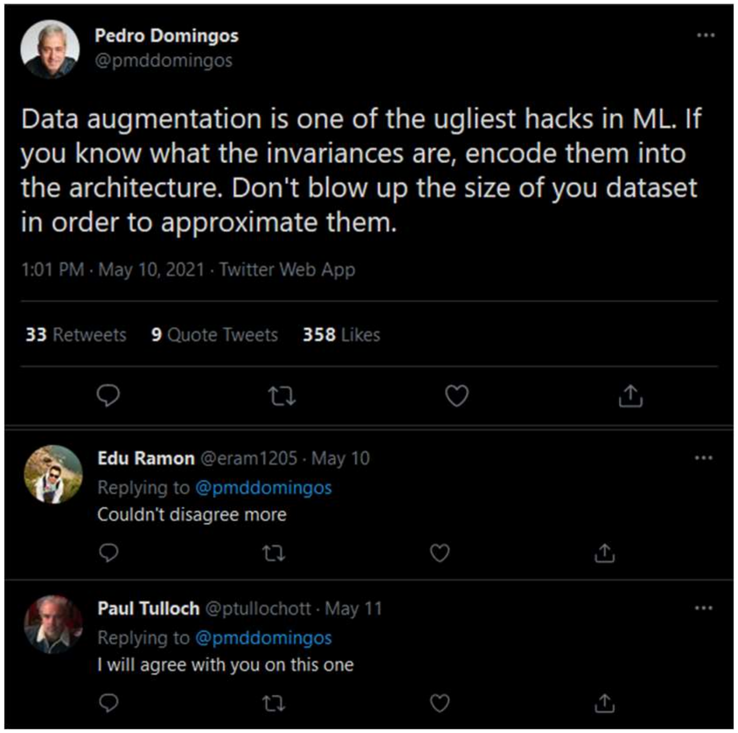
Q: According to KNN with k=1,
Should I recommend Space Force?

Curse of Dimensionality

- What if I have n=5 users and d=10000 shows?
  - Much less likely that nearest neighbours have "perfect match"
  - In fact, not very likely to have similar preferences at all.
  - "Curse of dimensionality": problems with high-dimensional spaces.
    - We saw a similar case is Naïve Bayes, where we needed O(2^d) examples
  - For each additional show, we need exponentially more users to preserve the usefulness of nearest neighbours

- KNN is also problematic if features have very different scales.
  - What if feature 1 is binary and feature 2 is continuous and can be huge?

- Nevertheless, KNN is really easy to use and often hard to beat!



Pedro Domingos
@pmddomingos

Data augmentation is one of the ugliest hacks in ML. If you know what the invariances are, encode them into the architecture. Don't blow up the size of you dataset in order to approximate them.

1:01 PM · May 10, 2021 · Twitter Web App

33 Retweets 9 Quote Tweets 358 Likes

Edu Ramon @eram1205 · May 10
Replying to @pmddomingos
Couldn't disagree more

Paul Tulloch @ptullochott · May 11
Replying to @pmddomingos
I will agree with you on this one

Coming Up Next

DATA AUGMENTATION

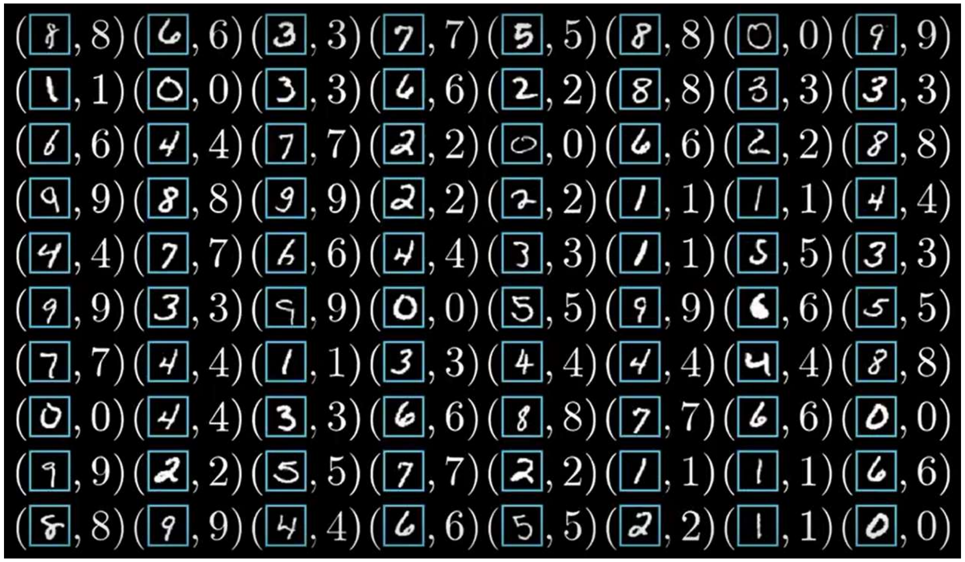
Application: Optical Character Recognition

Jane Doe
1095 Venture Dr.
Forest, VA 24551
John Q. Sample
123 Main Street
Anytown, US 12345

Q: How can we convert handwritten letters and digits into corresponding strings?

Application: Optical Character Recognition

- To scan documents, we want to turn images into characters:
  - "Optical character recognition" (OCR).



Q: How can we make this a supervised learning problem?

https://www.youtube.com/watch?v=IHZwWFHWa-w

Recall: Representing Images

m x n image

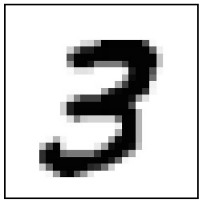
grayscale intensity

(1,1)	(2,1)	(3,1)	…	(m,1)	…	(m,n)
45	44	43	…	12	…	35

mn x 1 vector

Application: Optical Character Recognition

- Turning this into a supervised learning problem



28x28 image

“3”

String (or char)

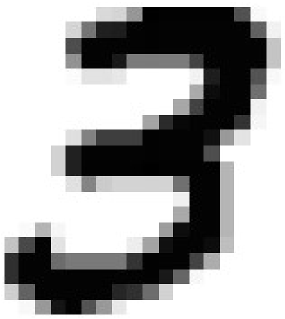
(1,1)	(2,1)	(3,1)	…	(28,1)	(1,2)	(2,2)	…	(14,14)	…	(28,28)
0	0	0		0	0	0		1		0
0	0	0		0	0	0		1		0
0	0	0		0	0	0		0		0
0	0	0		0	0	0		1		0

char
3
6
0
9

Human vs. Machine Perception

- There is huge difference between what we see and what computer sees:

What we see:



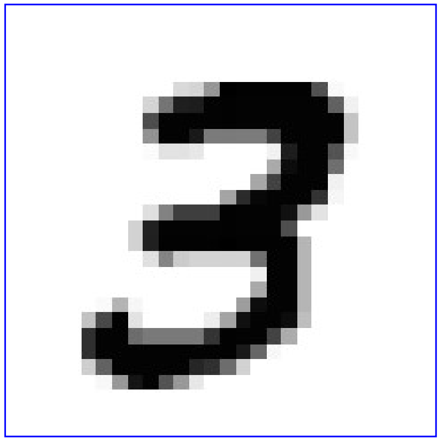
What the computer “sees”:

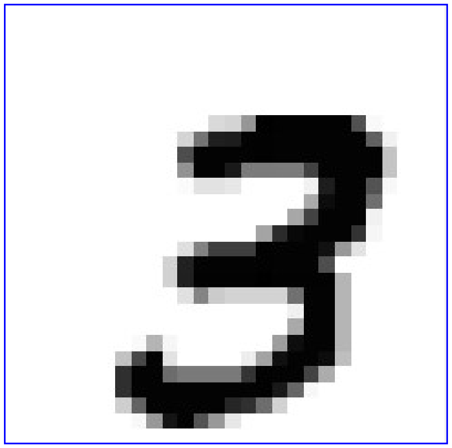
Impenetrable sea of numbers

Actually, it’s worse:

What the Computer Sees

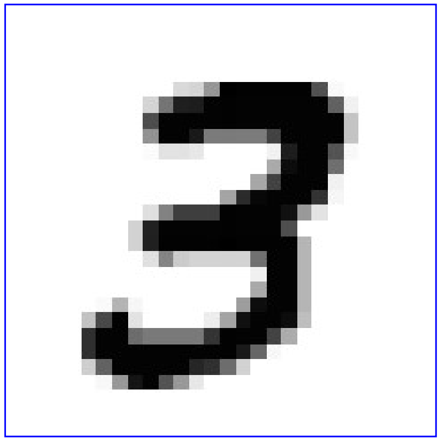
- Are these two images “similar”?

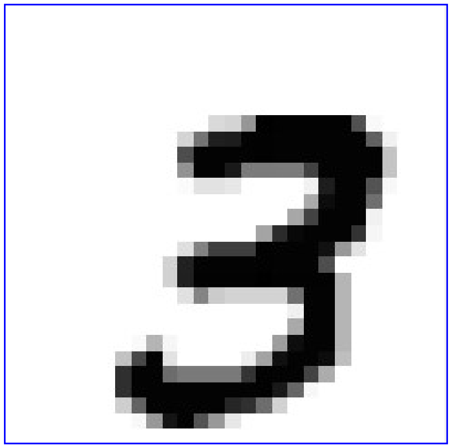




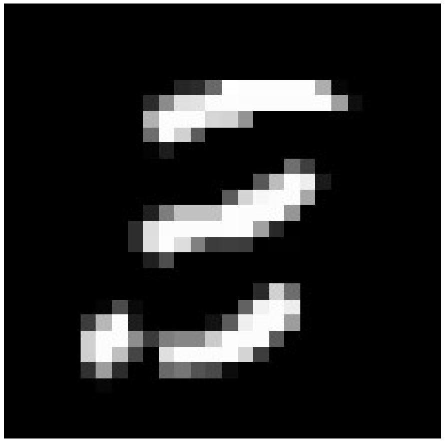
What the Computer Sees

- Are these two images “similar”?



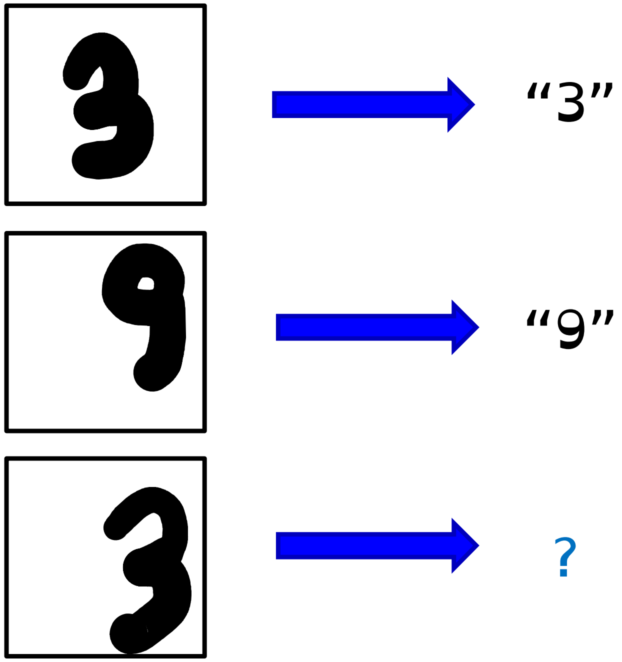


Difference:



Q: How would this make KNN fail?

Failure Due to Translation



"3"

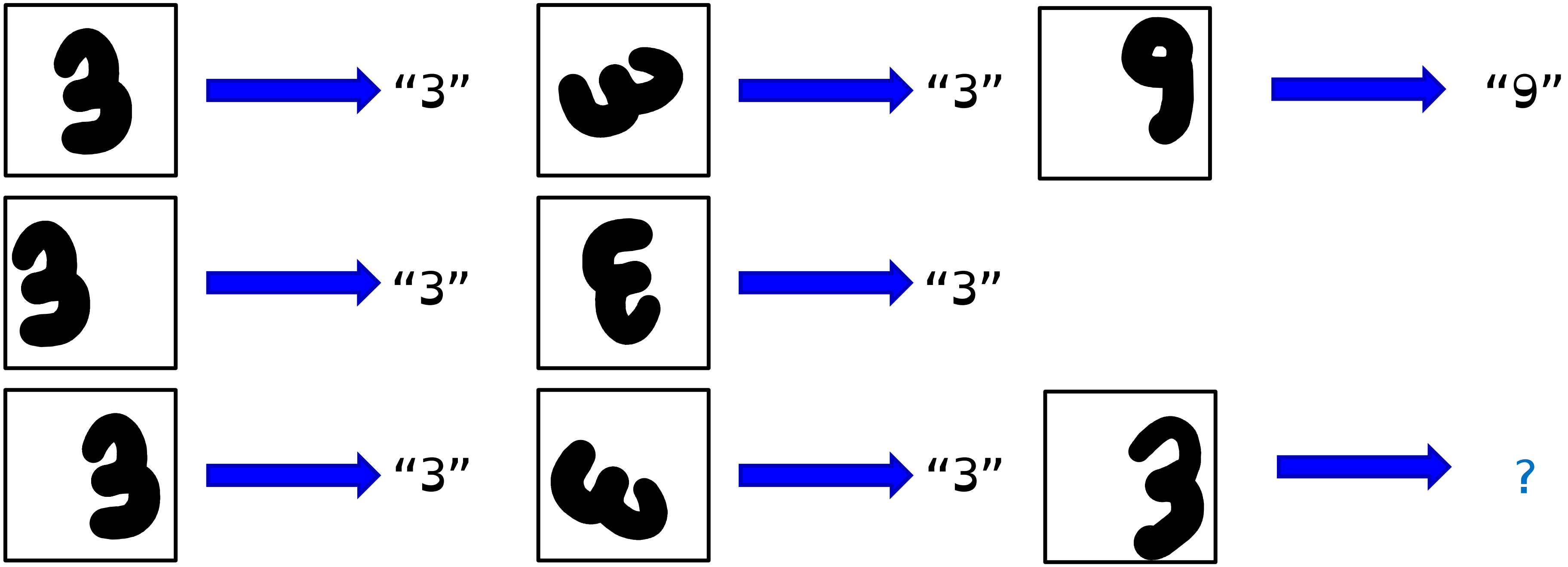
"9"

?

Q: How do we fix this?

Encouraging “Invariance"

- Idea: put translated images in training data



“Invariance”

- Invariance := recognizing exact same information in different-looking things
- E.g.
  - “3” on the right-hand corner is exactly the same thing as “3” at the center.
  - Sound effect at volume 5 is exactly the same thing as sound effect at volume 6
- Features “look” different but they correspond to the exact same signals.

Data Augmentation

- May want classifier to be invariant to certain feature transforms.
  - Images: translations, small rotations, changes in size, mild warping,

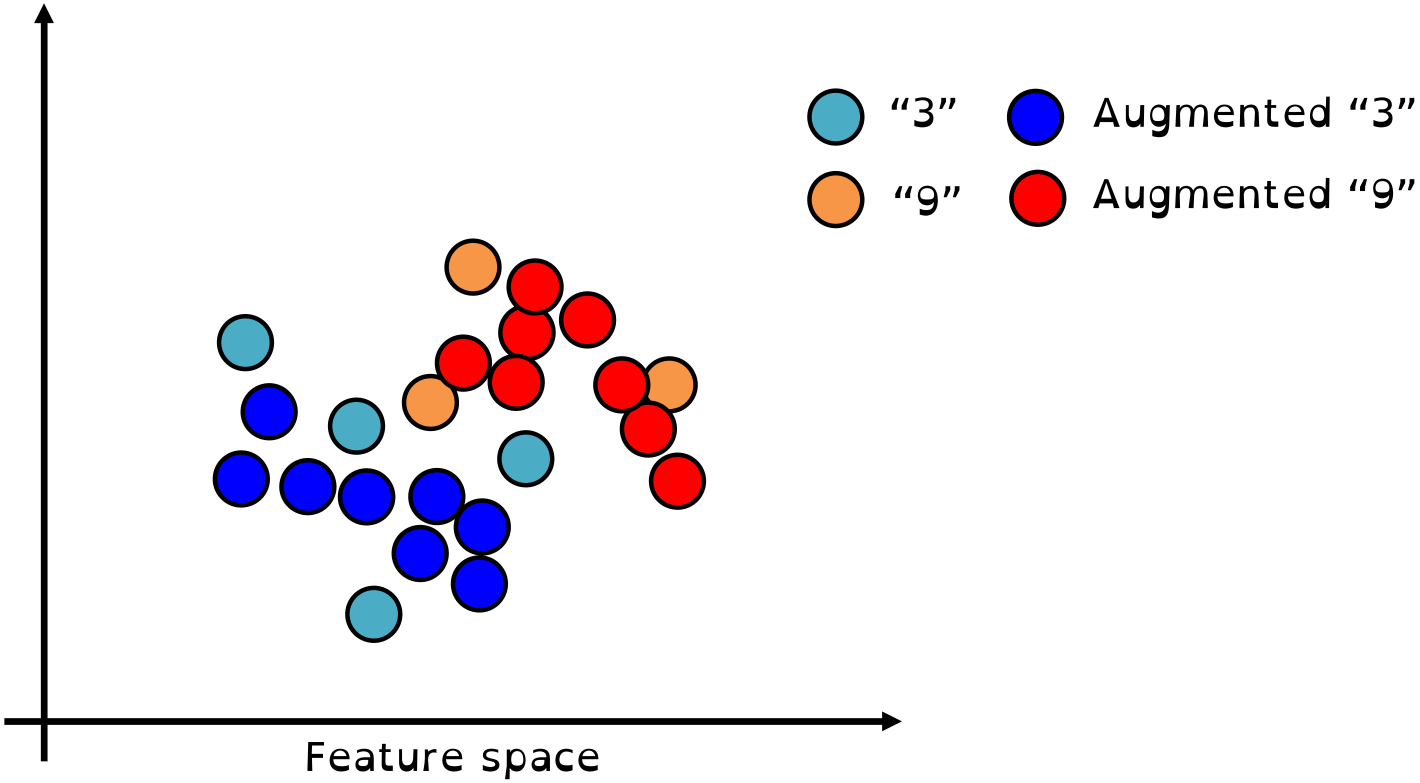
- The hard/slow way is to modify your distance function:
  - Find neighbours that require the "smallest" transformation of image.

- The easy/fast way is to just add transformed data during training:
  - Add translated/rotate/resized/warped versions of training images.
  - Crucial part of many successful vision systems.
  - Also important for sound (translate, change volume, and so on).

Visualizing Data Augmentation

$$X = \begin{array}{cc} n & \begin{bmatrix} \end{bmatrix} \\ & d \end{array} \quad y = \begin{array}{cc} n & \begin{bmatrix} \end{bmatrix} \\ & 1 \end{array} \rightarrow X_{aug} = \begin{array}{cc} n & \left[\begin{array}{c} X \\ \hdashline X_+ \end{array}\right] \\ n_+ & \\ & d \end{array} \quad y = \begin{array}{cc} n & \left[\begin{array}{c} y \\ \hdashline y_+ \end{array}\right] \\ n_+ & \\ & 1 \end{array}$$

Visualizing Data Augmentation



Q: Did we actually gain any information?

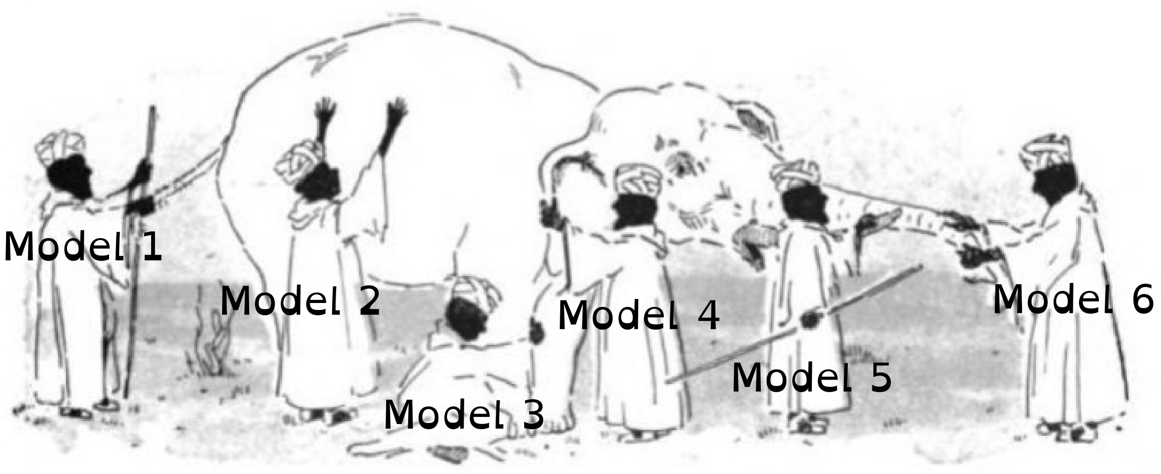
Why Data Augmentation?

- We (humans) did not gain any additional information
  - All the transformed “3”s already look like 3 anyways

- However, data augmentation improves model performance
  - We made the signal “thicker” by introducing more examples
  - We communicated to the model the invariances we want

“Online” Data Augmentation

- Online := introduce augmentation as you train the model
  - As opposed to offline, where you compute X_{aug} and y_{aug} first
  - E.g. while fitting a decision tree, augment (X_{yes}, y_{yes}) and (X_{no}, y_{no})
- Random transformations can be applied:
  - E.g. translate image in the X direction by some number between 0 and 28
- Online augmentation improves training by enhancing the distribution of training data (more on this later in course)

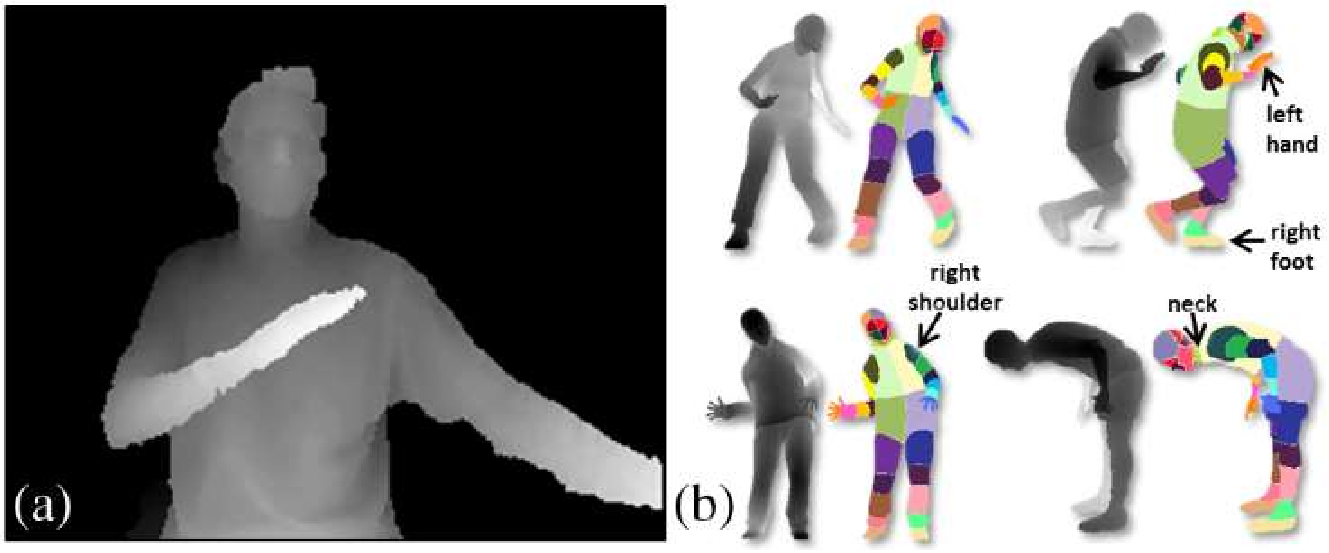


Coming Up Next

ENSEMBLE METHODS

Application: Body-Part Recognition

- Microsoft Kinect:
  - Real-time recognition of 31 body parts from laser depth data.

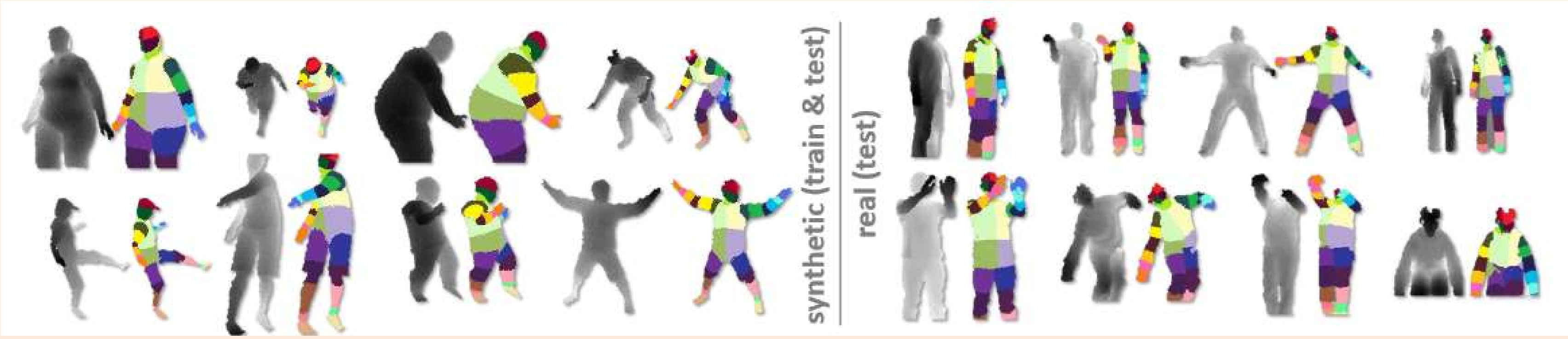


Q: How can we make this a supervised learning problem?

http://research.microsoft.com/pubs/158806/CriminisiForests_FoundTrends_2011.pdf

Some Ingredients of Kinect

1. Collect hundreds of thousands of labeled images (motion capture).
   – Variety of pose, age, shape, clothing, and crop.
2. Build a simulator that fills space of images by making even more images.



3. Extract features of each location, that are cheap enough for real-time calculation (depth differences between pixel and pixels nearby.)
4. Learn to classify body part of a pixel.
5. Run classifier in parallel on all pixels using graphical processing unit (GPU).

http://research.microsoft.com/pubs/145347/BodyPartRecognition.pdf

Supervised Learning Step

- ALL steps are important, but we'll focus on the learning step.

- Do we have any classifiers that are accurate and run in real time?
  - Decision trees and naïve Bayes are fast, but often not very accurate.
  - KNN is often accurate, but not very fast.

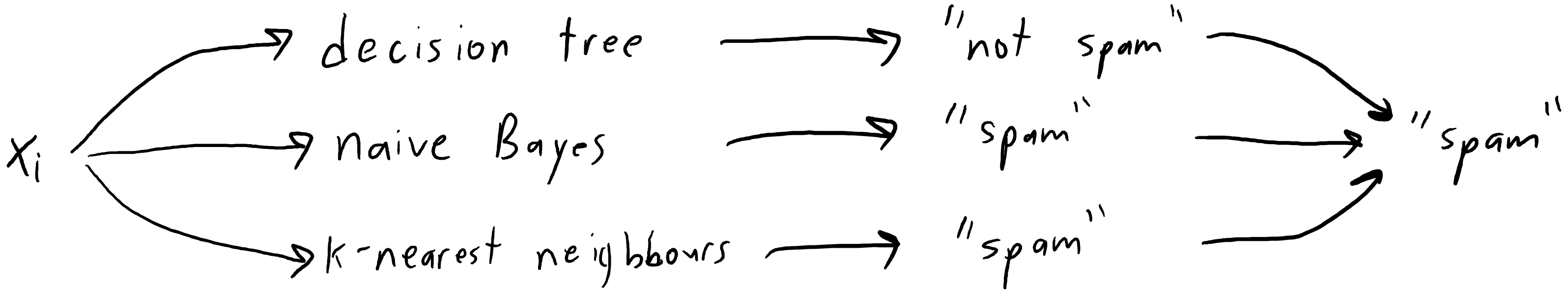
- Kinect deploys an ensemble method called random forests.

Ensemble Classifier

- Ensemble classifiers are classifiers that have classifiers as input.
  - Also called “meta-learning”.
- They have the best names:
  - Averaging.
  - Blending.
  - Boosting.
  - Bootstrapping.
  - Bagging.
  - Cascading.
  - Random Forests.
  - Stacking.
  - Voting.
- Ensemble classifiers often have higher accuracy than input classifiers.

Ensemble Method Example: Voting

- Ensemble methods use predictions of a set of models.
  - For example, we could use:
    - Decision trees make one prediction.
    - Naïve Bayes makes another prediction.
    - KNN makes another prediction.
- One of the simplest ensemble methods is voting:
  - Take the mode of the predictions across the classifiers.



Why can Voting Work?

- Consider 3 binary classifiers, each independently correct with probability 0.80:

- With voting, ensemble prediction is correct if we have "at least 2 right":
  - P(all 3 right) = 0.8^3 = 0.512.
  - P(2 rights, 1 wrong) = $3*0.8^2(1-0.8)$ = 0.384.
  - P(1 right, 2 wrongs) = $3*(1-0.8)^2 0.8$ = 0.096.
  - P(all 3 wrong) = $(1-0.8)^3$ = 0.008.
  - So ensemble is right with probability 0.896 (which is 0.512+0.384).

- Notes:
  - For voting to work, errors of classifiers need to be at least somewhat independent.
  - You also want the probability of being right to be > 0.5, otherwise it can do much worse.
  - Probabilities also shouldn't be too different (otherwise, it might be better to take most accurate).

Why can Voting Work?

- Why can voting lead to better results?

- Consider classifiers that overfit (like deep decision trees):
  - If they all overfit in exactly the same way, voting does nothing.

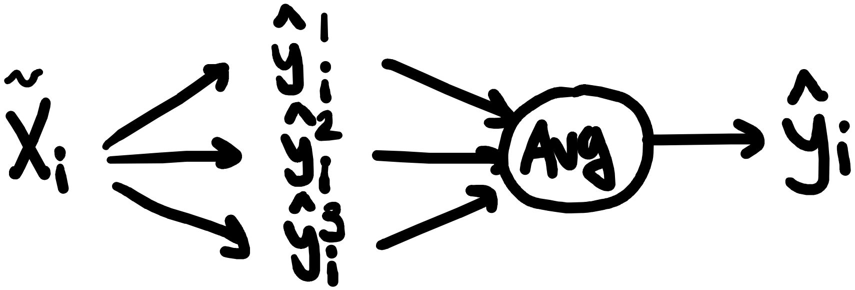
- But if they make _____________ errors:
  - Probability that “vote” is wrong can be lower than for each classifier.
  - Less attention to specific overfitting of each classifier.

Why can Voting Work?

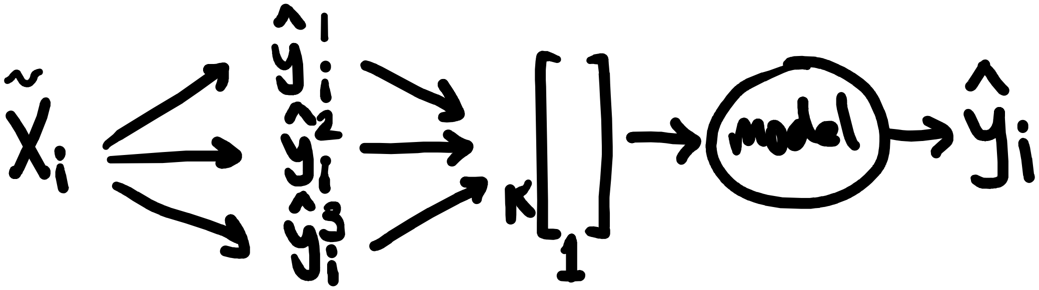
- Consider a set of classifiers that make these predictions:
  - Classifier 1: “spam”.
  - Classifier 2: “spam”.
  - Classifier 3: “spam”.
  - Classifier 4: “not spam”.
  - Classifier 5: “spam”.
  - Classifier 6: “not spam”.
  - Classifier 7: “spam”.
  - Classifier 8: “spam”.
  - Classifier 9: “spam”.
  - Classifier 10: “spam”.
- If these independently get 80% accuracy, mode will be close to 100%.
  - In practice errors won’t be completely independent (due to noise in labels).

Types of Ensemble Methods

- How predictions are populated:
  - Aggregation: take “average” of predictions
    - Voting is the special case we weight each prediction equally



  - Stacking: learn mapping of ensemble prediction -> final prediction



Types of Ensemble Methods

- How model is trained:
  - Bootstrapping: give each sub-model a uniquely shuffled dataset

$$\begin{tikzcd}
 & (X_{aug}, y_{aug})_1 \arrow[r] & \text{model}_1 \\
(X, y) \arrow[ur] \arrow[r] \arrow[dr] & (X_{aug}, y_{aug})_2 \arrow[r] & \text{model}_2 \\
 & (X_{aug}, y_{aug})_3 \arrow[r] & \text{model}_3
\end{tikzcd}$$

  - Boosting: chain sub-models and make later ones more “paranoid”
    - Will be covered later.

Summary

- Encouraging invariance:
  - Add transformed data to be insensitive to the transformation.
- Ensemble methods take multiplier classifiers as inputs.
- Voting ensemble method:
  - Improves predictions of multiple classifiers if errors are independent.

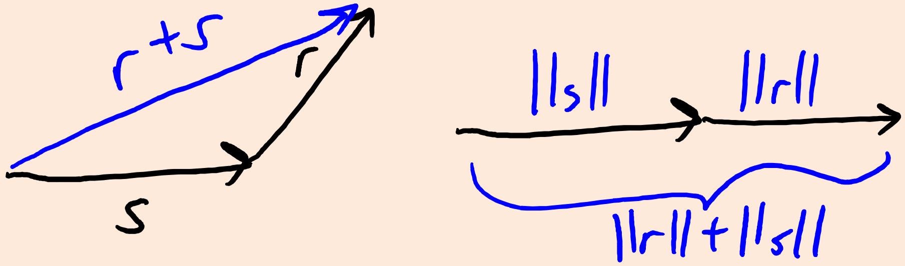
- Next time:
  - Random forests
  - We start unsupervised learning.

Review Questions

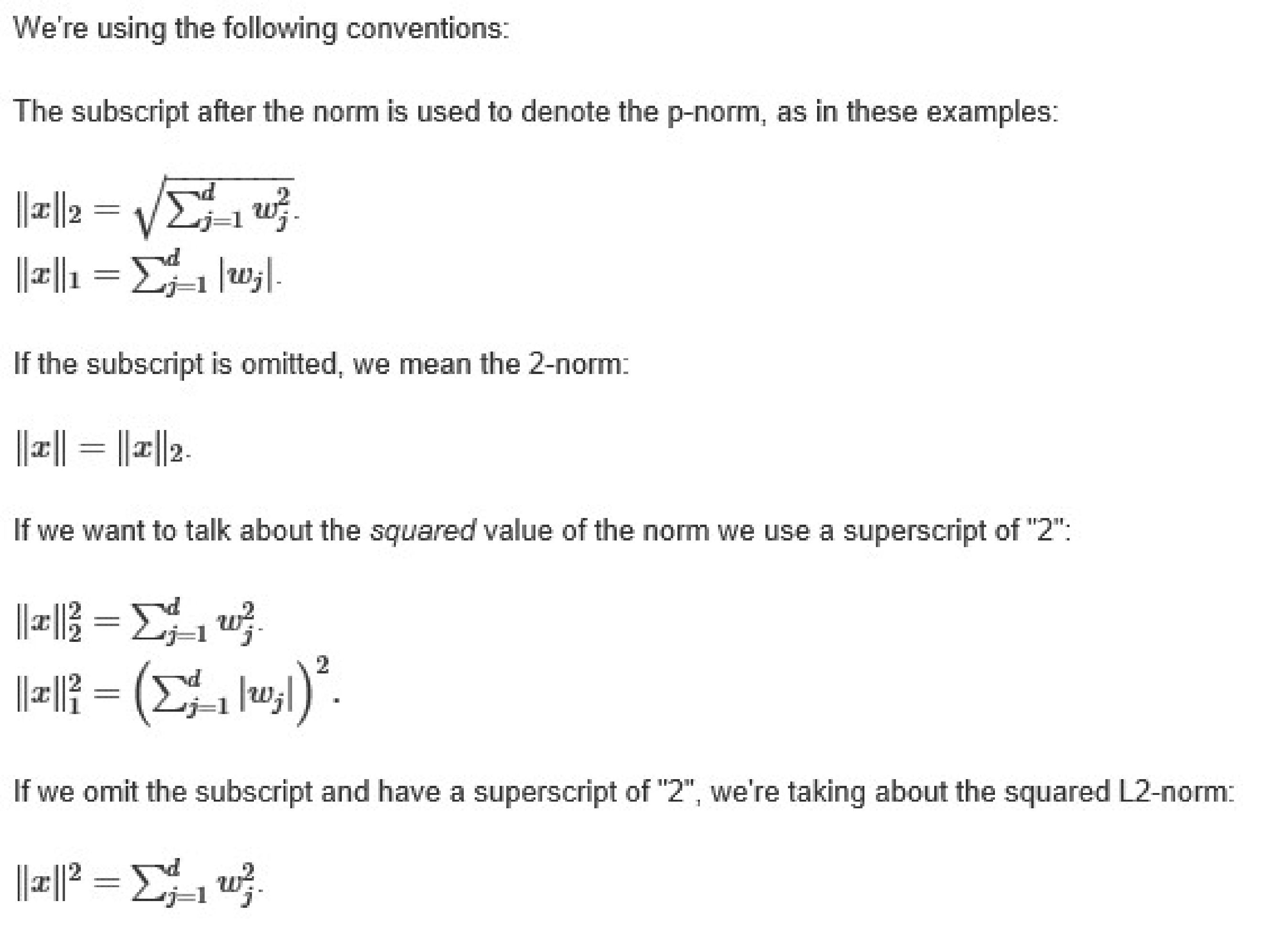
- Q1: KNN’s complexity increases with n. Does this mean KNN will overfit to the data for a large dataset?
- Q2: How does curse of dimensionality relate to the volume of a sphere in d-dimensions?
- Q3: What are the kinds of features you don’t want to encourage invariance for?
- Q4: Why is it important for models inside an ensemble to make independent errors?

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”: $\|\alpha r\| = |\alpha| \bullet \|r\|$.
     - 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”: $\|r + s\| \leq \|r\| + \|s\|$.



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:

$$\|x\|_2 = \sqrt{\textstyle\sum_{j=1}^{d} w_j^2}.$$
$$\|x\|_1 = \textstyle\sum_{j=1}^{d} |w_j|.$$

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":

$$\|x\|_2^2 = \textstyle\sum_{j=1}^{d} w_j^2.$$
$$\|x\|_1^2 = \left(\textstyle\sum_{j=1}^{d} |w_j|\right)^2.$$

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

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

Lp-norms

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

$$\|x\|_p = \left(|x_1|^p + |x_2|^p + \cdots + |x_n|^p\right)^{1/p}$$

- This gives a norm for any (real-valued) $p \geq 1$.
  - The L_∞-norm is the limit as ‘p’ goes to ∞.

- For $p < 1$, not a norm because triangle inequality not satisfied.

https://en.wikipedia.org/wiki/Lp_space

Why does Bootstrapping select approximately 63%?

- Probability of an arbitrary x_i being selected in a bootstrap sample:

p(selected at least once in 'n' trials)

= 1 − p(not selected in any of 'n' trials)

= 1 − (p(not selected in one trial))^n (trials are independent)

$= 1 - (1 - 1/n)^n$ (prob $= \frac{n-1}{n}$ for choosing any of the n-1 other samples)

$\approx 1 - 1/e$ ($(1 - 1/n)^n \to e^{-1}$ as $n \to \infty$)

≈ 0.63

Why Averaging Works

- Consider 'k' independent classifiers, whose errors have a variance of σ^2.
- If the errors are IID, the variance of the vote is σ^2/k.
  - So the more classifiers that vote, the more you decrease error variance. (And the more the training error approximates the test error.)
- Generalization to case where classifiers are not independent is:
  - Where 'c' is the correlation. $c\sigma^2 + \frac{(1-c)}{k}\sigma^2$
- So the less correlation you have the closer you get to independent case.
- Randomization in random forests decreases correlation between trees.
  - See also "Sensitivity of Independence Assumptions".

How these concepts often show up in practice

- Here is a recent e-mail related to many ideas we've recently covered:
  - "However, the performance did not improve while the model goes deeper and with augmentation. The best result I got on validation set was 80% with LeNet-5 and NO augmentation (LeNet-5 with augmentation I got 79.15%), and later 16 and 50 layer structures both got 70%~75% accuracy.

    In addition, there was a software that can use mathematical equations to extract numerical information for me, so I trained the same dataset with nearly 100 features on random forest with 500 trees. The accuracy was 90% on validation set.

    I really don't understand that how could deep learning perform worse as the number of hidden layers increases, in addition to that I have changed from VGG to ResNet, which are theoretically trained differently. Moreover, why deep learning algorithm cannot surpass machine learning algorithm?"
- Above there is data augmentation, validation error, effect of the fundamental trade-off, the no free lunch theorem, and the effectiveness of random forests.