

CPSC 340: Machine Learning and Data Mining

Ensemble Methods

Fall 2021

Admin

- Course webpage:
 - <https://github.com/UBC-CS/cpsc340-2021w1>
- Assignment 2 is out
 - Due Friday of next week. It's long – start early
 - Keep an eye on Piazza and/or commits on the site for updates/fixes
- Looking again for a volunteer to keep an eye on Piazza Live Q&A

Last Time: K-Nearest Neighbours (KNN)

- K-nearest neighbours algorithm for classifying \tilde{x}_i :

- Find ‘k’ values of x_i that are most similar to \tilde{x}_i .
 - Use mode of corresponding y_i .

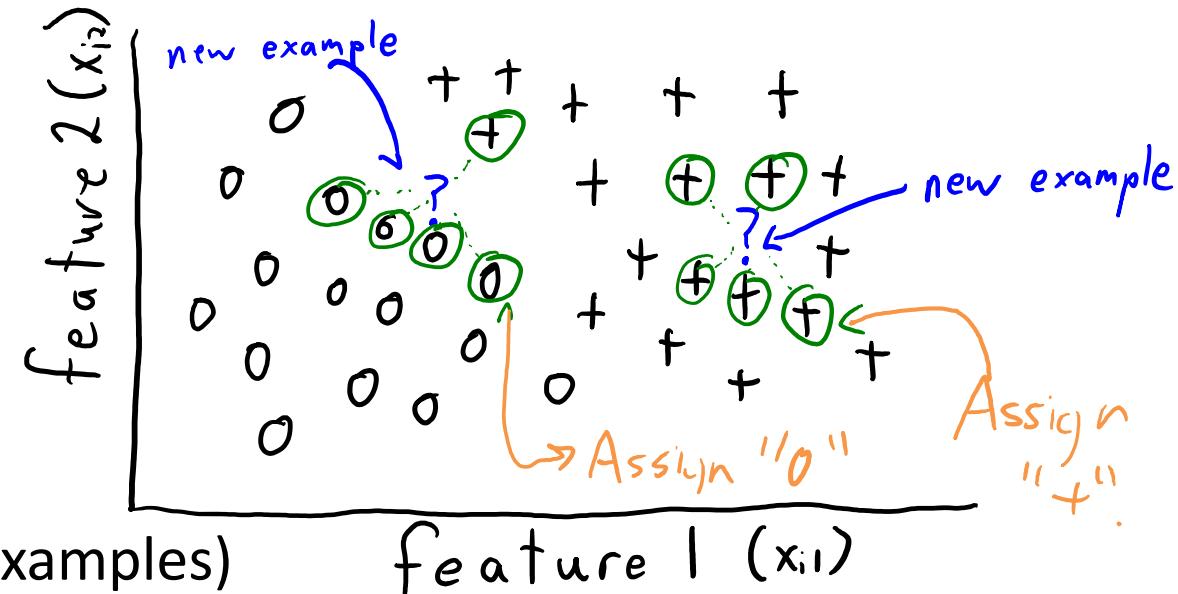
- Lazy learning:

- To “train” you just store X and y.

- Non-parametric:

- Size of model grows with ‘n’ (number of examples)
 - Nearly-optimal test error with infinite data.

- But high prediction cost and may need large ‘n’ if ‘d’ is large.



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”):

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

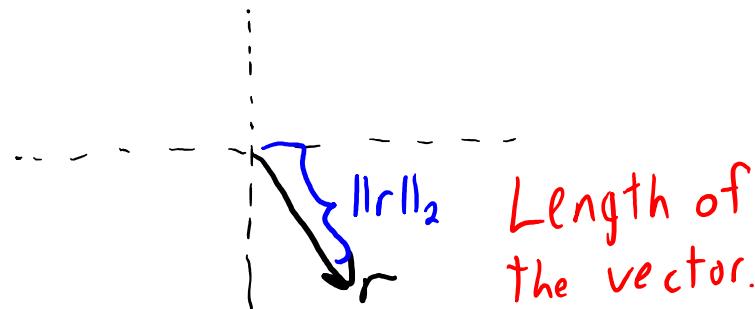
- Here, the “norm” of the difference is the standard Euclidean distance.

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:

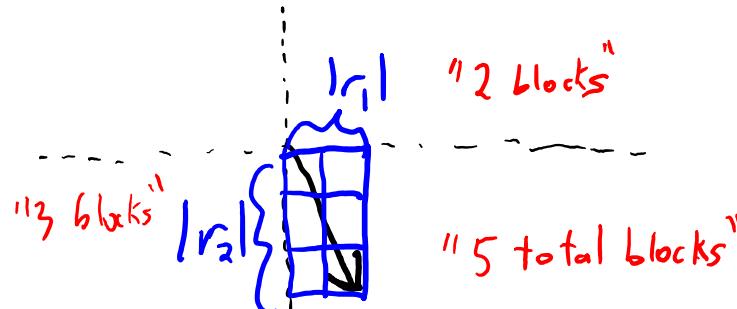
L₂ or "Euclidean" norm.

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



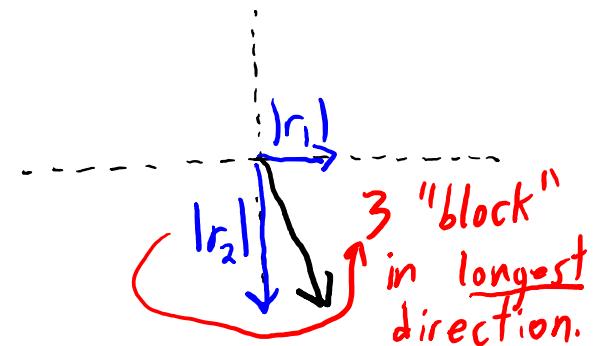
L₁ or "Manhattan" norm:

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



L _{∞} or "max" norm:

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



- Definitions of these norms in d-dimensions.

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

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

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

Norm and Norm^p Notation (MEMORIZE)

- Notation:

- We often leave out the “2” for the L2-norm: We use $\|r\|$ for $\|r\|_2$
- We use superscripts for raising norms to powers: We use $\|r\|^2$ for $(\|r\|)^2$
- You should understand why all of the following quantities are equal:

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

$$= \langle r, r \rangle$$

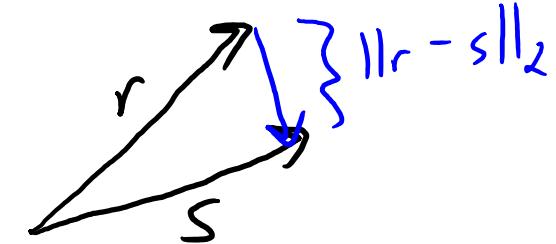
(we'll use
these later)

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."

- Place different “weights” on large differences:
 - L_1 : differences are equally notable.
 - L_2 : bigger differences are more important (because of squaring).
 - L_∞ : only biggest difference is important.

bonus!

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):

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

\uparrow "weight" of feature j

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

- See bonus slide for what functions define a “norm”.

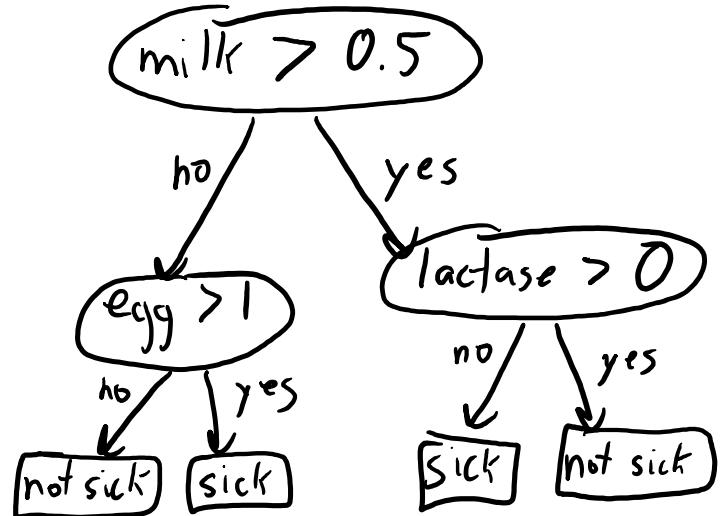
- But we can consider other distance/similarity functions:

- Jaccard similarity (if x_i are sets).

- Edit distance (if x_i are strings).

- Metric learning (*learn* the best distance function).

Decision Trees vs. Naïve Bayes vs. KNN



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

$(\text{milk} = 0.6, \text{egg} = 2, \text{lactase} = 0, ?)$ is close to
 $(\text{milk} = 0.7, \text{egg} = 2, \text{lactase} = 0, \text{sick})$ so predict sick.

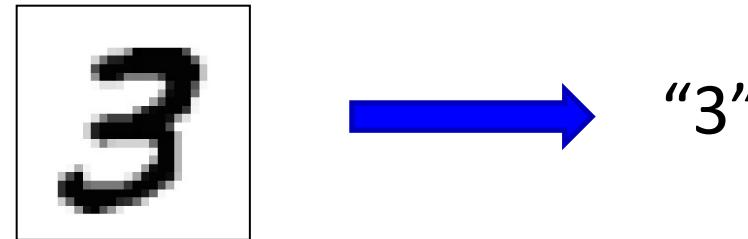
Application: Optical Character Recognition

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

(, 8) (, 6) (, 3) (, 7) (, 5) (, 8) (, 0) (, 9)
(, 1) (, 0) (, 3) (, 6) (, 2) (, 8) (, 3) (, 3)
(, 6) (, 4) (, 7) (, 2) (, 0) (, 6) (, 2) (, 8)
(, 9) (, 8) (, 9) (, 2) (, 2) (, 1) (, 1) (, 4)
(, 4) (, 7) (, 6) (, 4) (, 3) (, 1) (, 5) (, 3)
(, 9) (, 3) (, 9) (, 0) (, 5) (, 9) (, 6) (, 5)
(, 7) (, 4) (, 1) (, 3) (, 4) (, 4) (, 4) (, 8)
(, 0) (, 4) (, 3) (, 6) (, 8) (, 7) (, 6) (, 0)
(, 9) (, 2) (, 5) (, 7) (, 2) (, 1) (, 1) (, 6)
(, 8) (, 9) (, 4) (, 6) (, 5) (, 2) (, 1) (, 0)

Application: Optical Character Recognition

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



- Turning this into a supervised learning problem (with 28 by 28 images):

$$X = \begin{bmatrix} (1,1) & (2,1) & (3,1) & \dots & (28,1) & (1,2) & (2,2) & \dots & (14,14) & \dots & (28,28) \end{bmatrix}$$
$$Y = \begin{bmatrix} 3 \\ 6 \\ 0 \\ 9 \end{bmatrix}$$

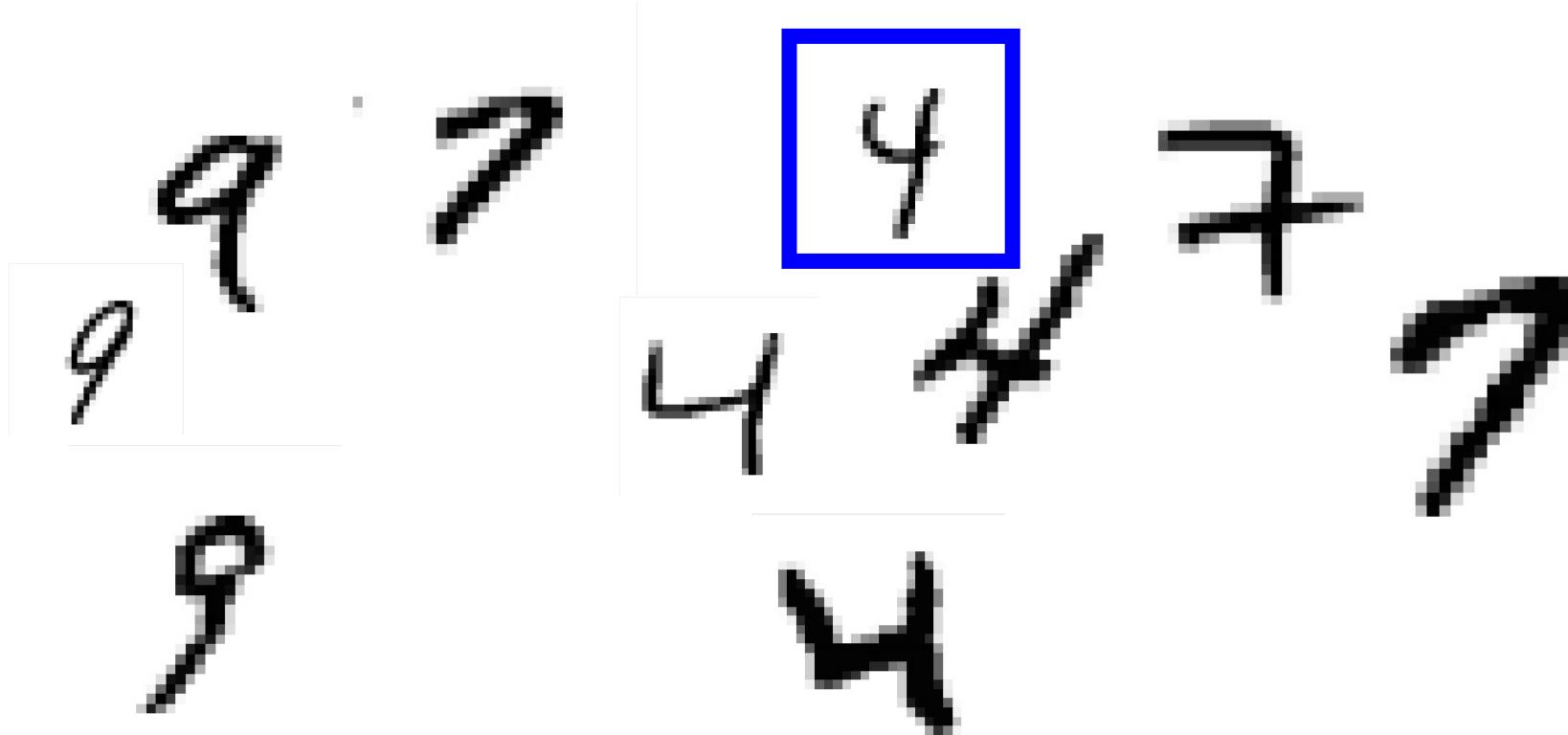
The table X represents a 28x28 image of the digit '3' as a vector of 784 features. The vector is organized into 28 columns, each representing a row of 28 pixels. The first few columns are labeled $(1,1)$, $(2,1)$, $(3,1)$, ..., $(28,1)$, and the next few columns are labeled $(1,2)$, $(2,2)$, ..., $(14,14)$, ..., $(28,28)$. The value 1 is located at index $(14,14)$, indicating the center pixel of the digit '3'.

Each feature is grayscale intensity of one of the 784 pixels

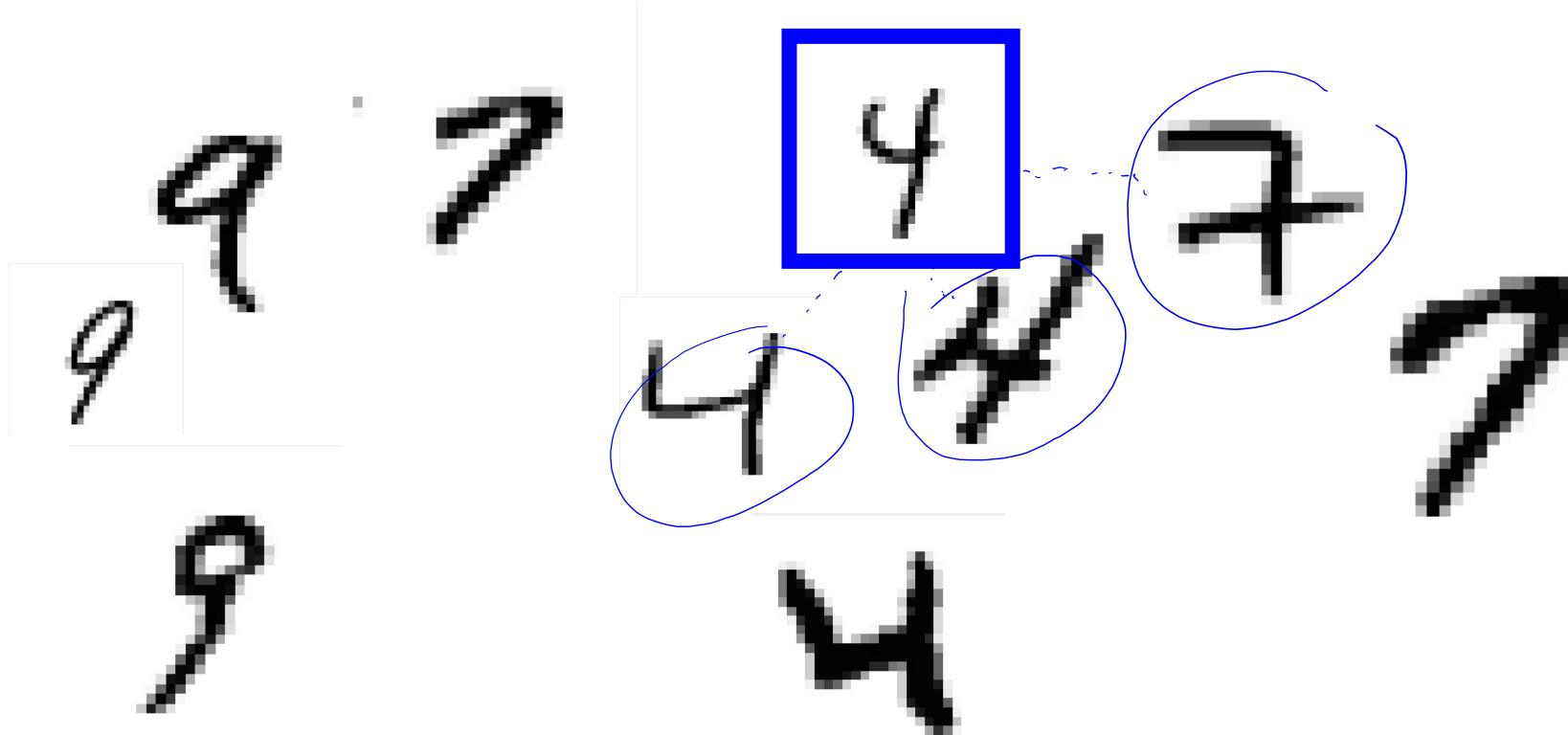
KNN for Optical Character Recognition



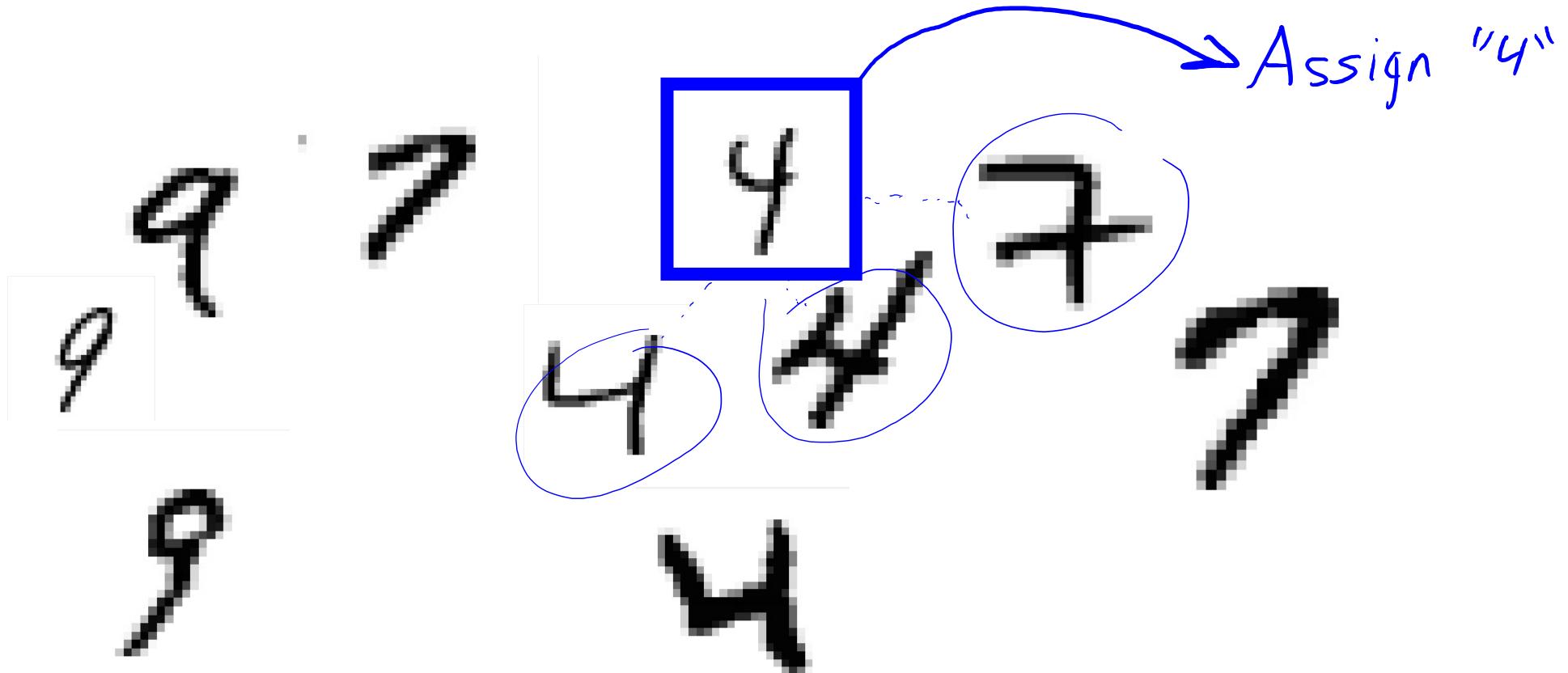
KNN for Optical Character Recognition



KNN for Optical Character Recognition



KNN for Optical Character Recognition



Human vs. Machine Perception

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

What we see:



What the computer “sees”:

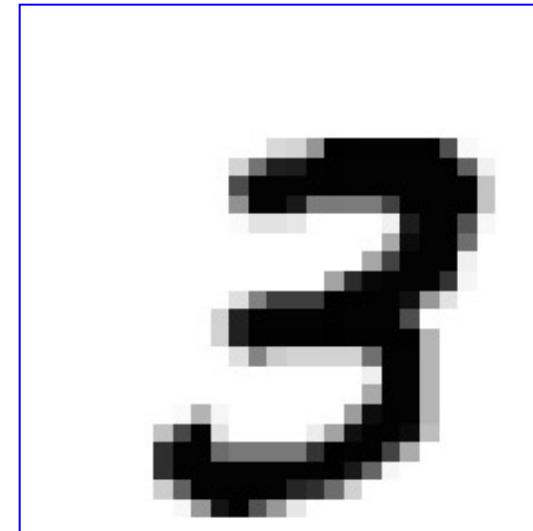
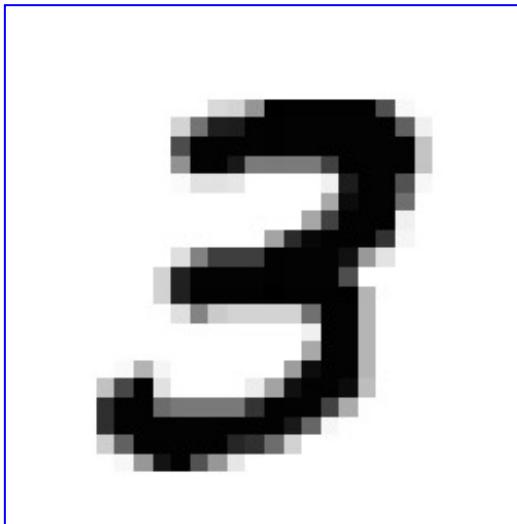


Actually, it's worse:



What the Computer Sees

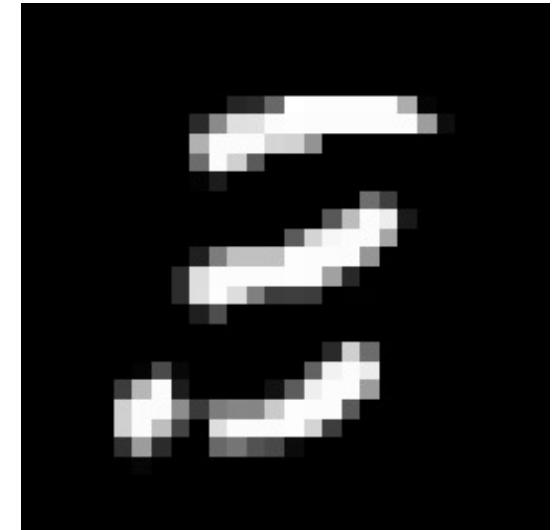
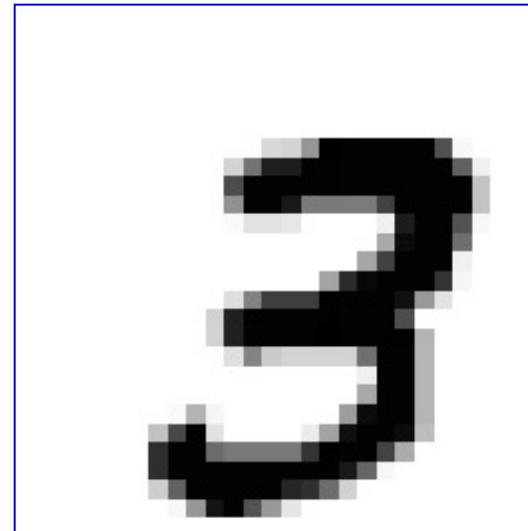
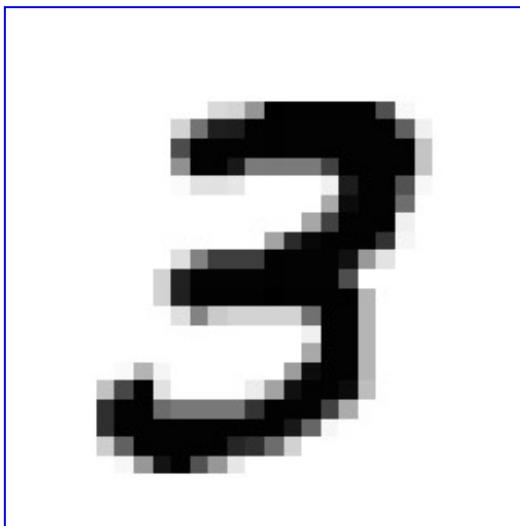
- Are these two images “similar”?



What the Computer Sees

- Are these two images “similar”?

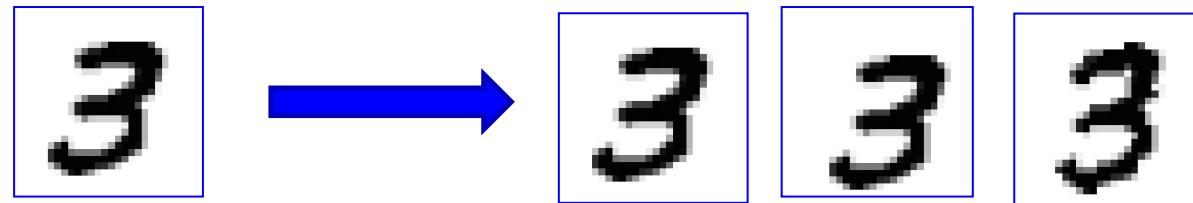
Difference:



- KNN does not know that labels should be translation invariant.

Encouraging Invariance

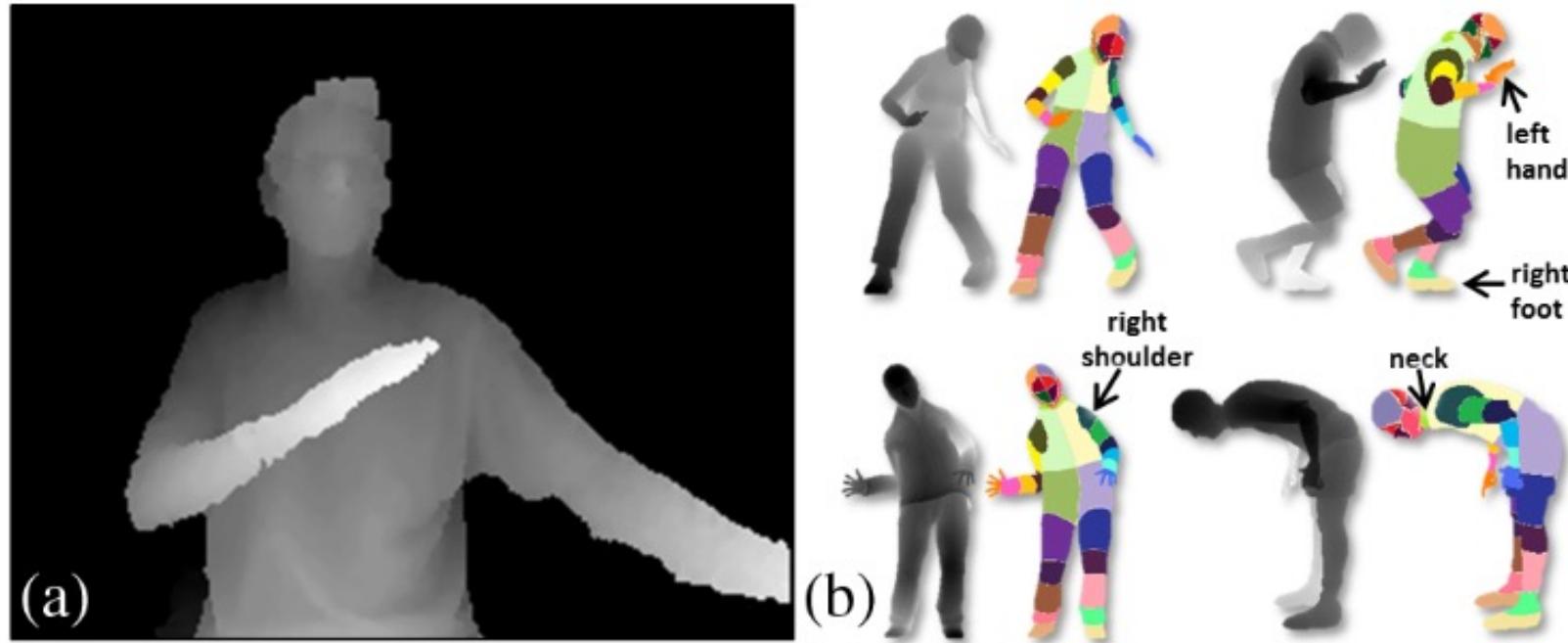
- May want classifier to be invariant to certain feature transforms.
 - Images: translations, small rotations, changes in size, mild warping,...
- The **hard/slow way** is to modify your distance function:
 - Find neighbours that require the “smallest” transformation of image.
- The **easy/fast way** is to just **add transformed data** during training:
 - Add translated/rotate/resized/warped versions of training images.



- “Data augmentation”: crucial part of many successful vision systems.
- Also really important for sound (translate, change volume, and so on).

Application: Body-Part Recognition

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

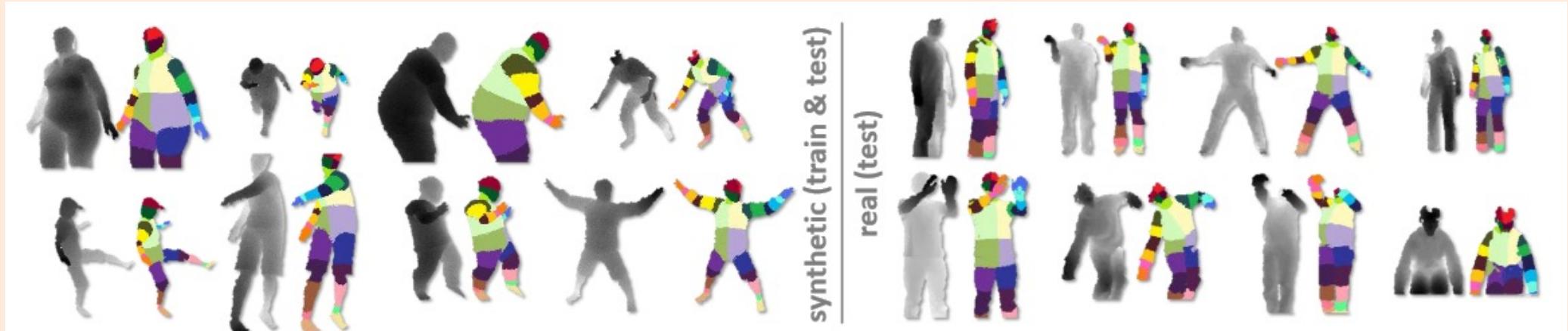


- How could we write a program to do this?

bonus!

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. Treat **classifying body part of a pixel as a supervised learning** problem.
5. Run **classifier in parallel on all pixels** using graphical processing unit (GPU).

Supervised Learning Step

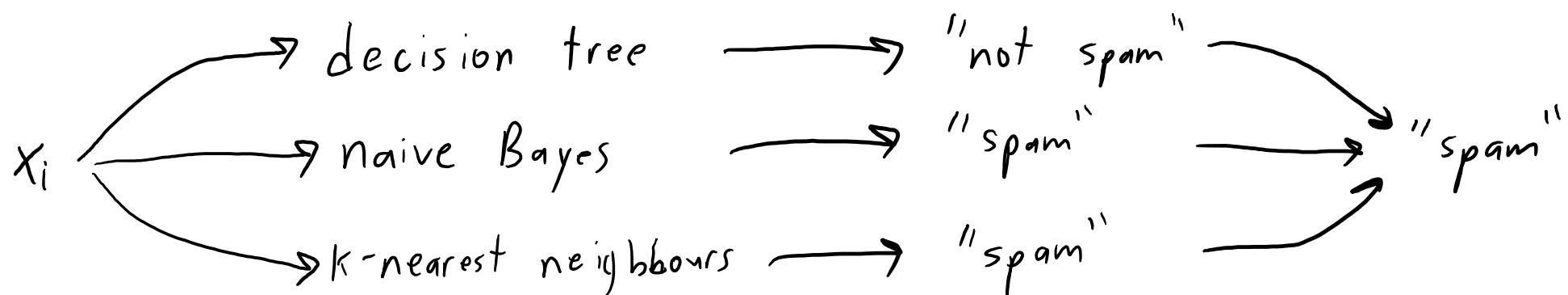
- ALL steps are important, but we'll focus on the learning step.
- Do we have any classifiers that are accurate and run in real time?
 - Decision trees and naïve Bayes are fast, but often not very accurate.
 - KNN is often accurate, but not very fast.
- Deployed system uses an ensemble method called random forests.

Ensemble Methods

- Ensemble methods are classifiers that combine other classifiers.
- They have the best names:
 - Averaging.
 - Blending.
 - Boosting.
 - Bootstrapping.
 - Bagging.
 - Cascading.
 - Random Forests.
 - Stacking.
 - Voting.
- Ensemble methods 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.



bonus!

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”**:
 - $P(\text{all 3 right}) = 0.8^3 = 0.512.$
 - $P(2 \text{ rights}, 1 \text{ wrong}) = 3 * 0.8^2(1-0.8) = 0.384.$
 - $P(1 \text{ right}, 2 \text{ wrongs}) = 3 * (1-0.8)^20.8 = 0.096.$
 - $P(\text{all 3 wrong}) = (1-0.8)^3 = 0.008.$
 - 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?

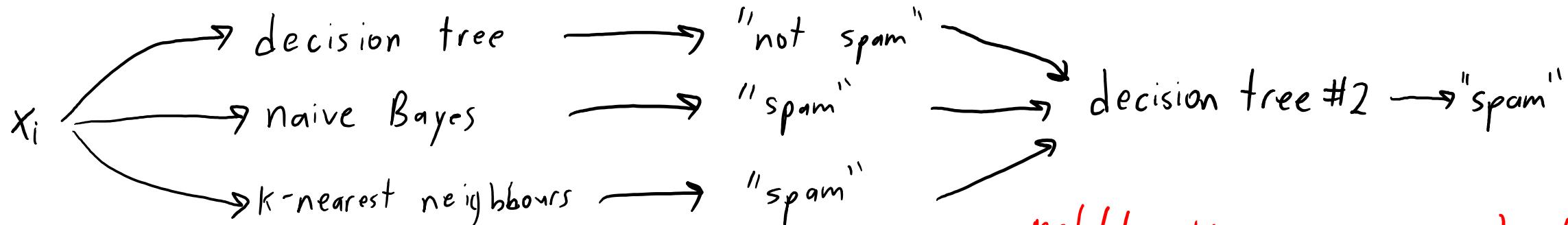
- Why can voting lead to better results?
- Consider classifiers that overfit (like deep decision trees):
 - If they all overfit in exactly the same way, voting does nothing.
- But if they make **independent errors**:
 - Probability that “vote” is wrong can be lower than for each classifier.
 - Less attention 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).

Digression: Stacking

- Another variation on voting is **stacking**
 - Fit another classifier that uses the predictions as features.



- Can tune second classifier using validation data.
 - Sometimes called "**blending**".
- Stacking often performs **better than individual models**.
 - Typically used by Kaggle winners.
 - E.g., Netflix \$1M user-rating competition winner was stacked classifier.

$$X = \begin{bmatrix} \text{model 1} & \text{model 2} & \text{model 3} \\ \text{not spam} & \text{spam} & \text{spam} \\ \text{spam} & \text{spam} & \text{spam} \\ \text{not spam} & \text{not spam} & \text{spam} \\ \vdots & \vdots & \vdots \end{bmatrix}$$
$$y = \begin{bmatrix} \text{true label} \\ \text{spam} \\ \text{spam} \\ \text{not spam} \\ \vdots \end{bmatrix}$$

Random Forests

- Random forests take vote from a set of deep decision trees.
 - Tend to be one of the best “out of the box” classifiers.
 - Often close to the best performance of any method on the first run.
 - And predictions are very fast.
- Do deep decision trees make independent errors?
 - No: with the same training data you’ll get the same decision tree.
- Two key ingredients in random forests:
 - Bootstrapping.
 - Random trees.

Bootstrap Sampling

- Start with a standard deck of 52 cards:

- Sample a random card:

(put it back and re-shuffle)



- Sample a random card:

(put it back and re-shuffle)



- Sample a random card:

(put it back and re-shuffle)



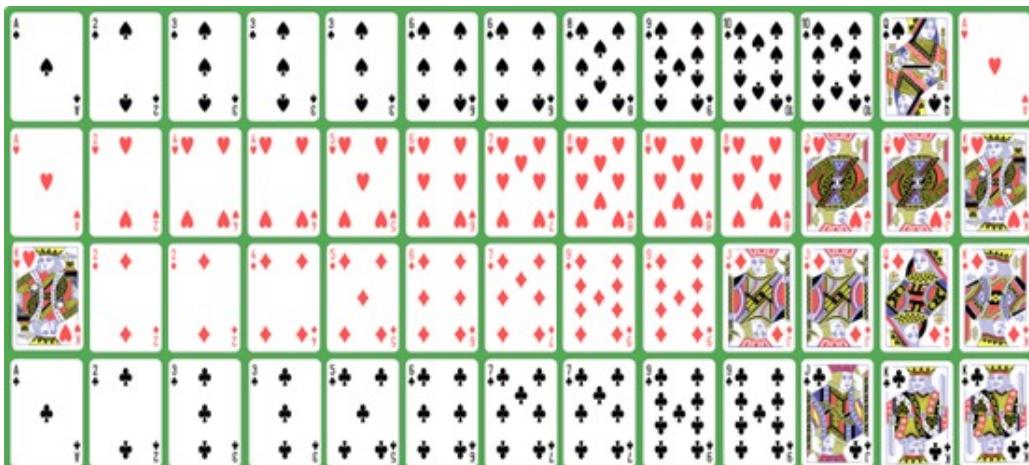
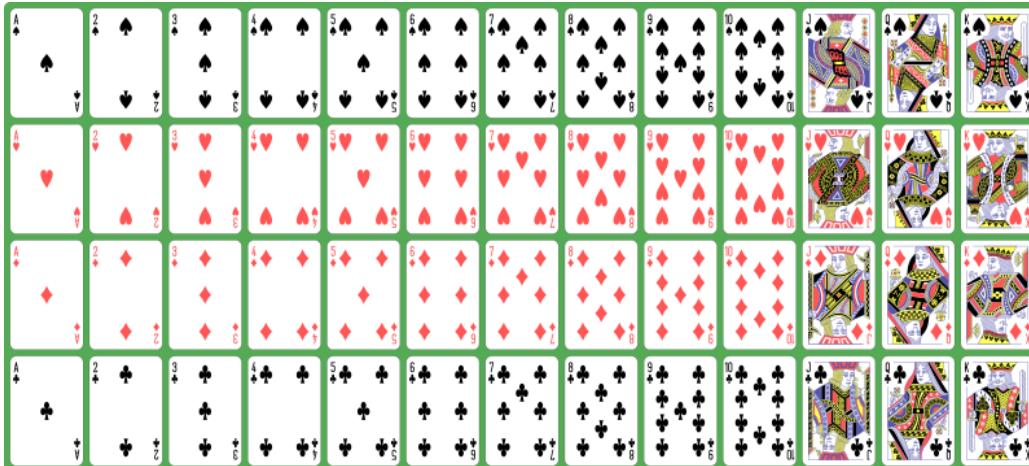
— ...

- Sample a random card:

(which may be a repeat)

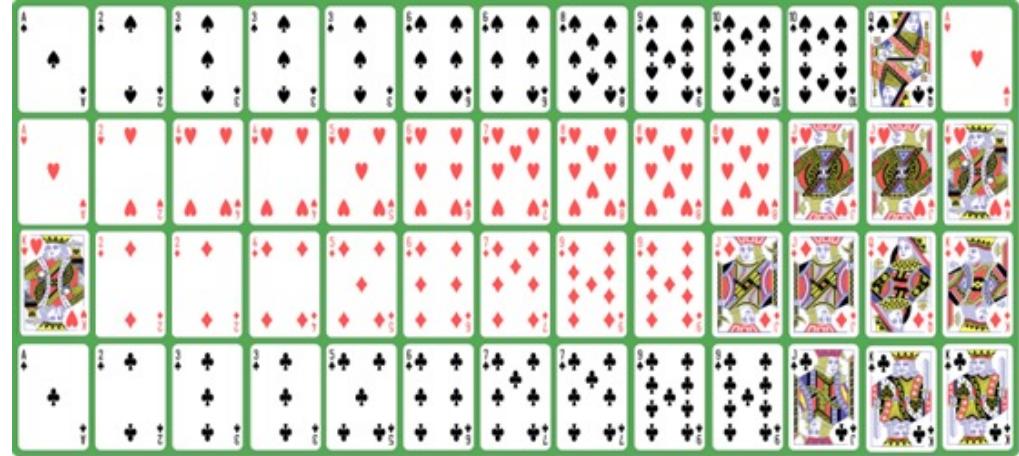


- Makes a new deck of the 52 samples:



Bootstrap Sampling

- New 52-card deck is called a “bootstrap sample”:



- Some cards will be missing, and some cards will be duplicated.
 - So calculations on the bootstrap sample will give different results than original data.
- However, the bootstrap sample roughly maintains trends:
 - Roughly 25% of the cards will be diamonds.
 - Roughly 3/13 of the cards will be “face” cards.
 - There will be roughly four “10” cards.
- Common use: compute a statistic based on several bootstrap samples.
 - Gives you an idea of how the statistic varies as you vary the data.

Random Forest Ingredient 1: Bootstrap

- **Bootstrap sample** of a list of ‘n’ examples:

- A new set of size ‘n’ chosen independently with replacement.

```
for i in 1:n  
    j = rand(1:n) # pick a random number from {1,2,...,n}  
    X_bootstrap[i,:] = X[j,:] # use the random sample
```

- Gives new dataset of ‘n’ examples, with some duplicated and some missing.
 - For large ‘n’, approximately 63% of original examples are included.

- **Bagging**: using bootstrap samples for ensemble learning.

- Generate several **bootstrap samples of the examples** (x_i, y_i).
 - Fit a **classifier** to each bootstrap sample.
 - At test time, take **vote based on the predictions**.

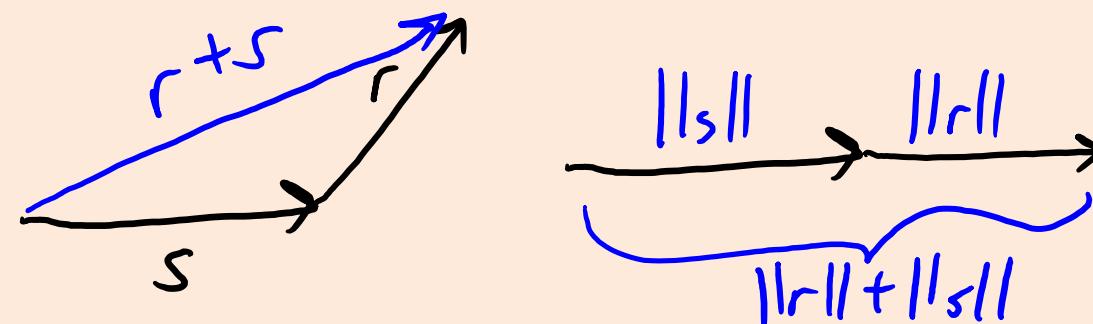
Summary

- Encouraging invariance with **data augmentation**:
 - 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.
- **Bagging**:
 - Ensemble method where we apply same classifier to “bootstrap samples”.
- Next time:
 - Unsupervised learning.

bonus!

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”: $\| r + s \| \leq \| r \| + \| s \|$.



bonus!

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:

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

bonus!

L_p-norms

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

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

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

Why does Bootstrapping select approximately 63%?

bonus!

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

$p(\text{selected at least once in } n \text{ trials})$

$$= 1 - p(\text{not selected in any of } n \text{ trials})$$

$$= 1 - (p(\text{not selected in one trial}))^n$$

$$= 1 - (1 - 1/n)^n$$

$$\approx 1 - 1/e$$

$$\approx 0.63$$

(trials are independent)

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

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

bonus!

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:

$$c \sigma^2 + \frac{(1-c)}{k} \sigma^2$$

- Where ‘c’ is the correlation.
- 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

bonus!

- 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.