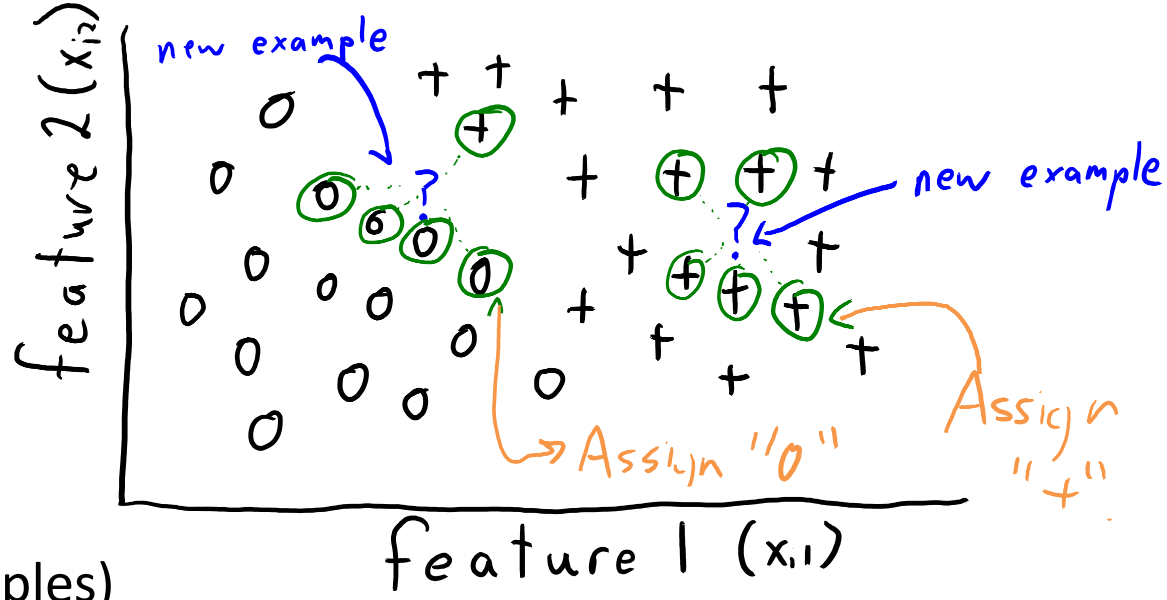
# CPSC 340:
# Machine Learning and Data Mining

Ensemble Methods

Fall 2022

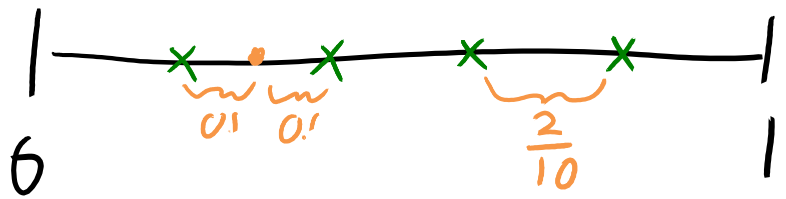
# Last Time: K-Nearest Neighbours (KNN)

- K-nearest neighbours algorithm for classifying $\tilde{x}_i$: (with hyper-parameter 'k').
  - 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.
- High prediction and storage cost.
- Non-parametric:
  - Size of model grows with 'n' (number of examples)
- Universal consistency:
  - Optimal test error with infinite data for appropriately-growing 'k'.



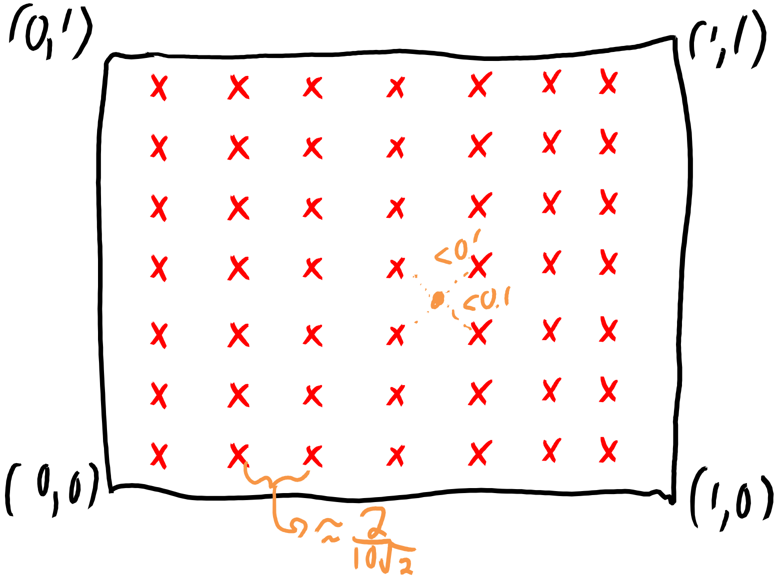
# Curse of Dimensionality

- "Curse of dimensionality": volume grows exponentially with dimension.
  - Consider the interval from 0 to 1 (d=1).
    - If want every location on to have a "neighbor" with distance $\epsilon$, need at least O(1/$\epsilon$) points.
      - With 4 well-placed points you can guarantee that you have a "neighbor" within 0.1.



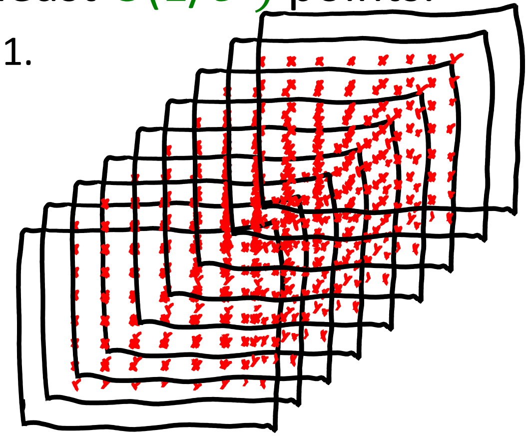
# Curse of Dimensionality

- "Curse of dimensionality": volume grows exponentially with dimension.
  - Consider the interval from 0 to 1 (d=1).
    - If want every location on to have a "neighbor" with distance $\epsilon$, need at least $O(1/\epsilon)$ points.
      - With 4 well-placed points you can guarantee that you have a "neighbor" within 0.1.
  - Consider the unit square (d=2).
    - If want every location on to have a "neighbor" with distance $\epsilon$, need at least $O(1/\epsilon^2)$ points.
      - With 49 well-placed points you can guarantee that you have a "neighbor" within 0.1.



# Curse of Dimensionality

- "Curse of dimensionality": volume grows exponentially with dimension.
  - Consider the interval from 0 to 1 (d=1).
    - If want every location on to have a "neighbor" with distance $\epsilon$, need at least $O(1/\epsilon)$ points.
      - With 4 well-placed points you can guarantee that you have a "neighbor" within 0.1.
  - Consider the unit square (d=2).
    - If want every location on to have a "neighbor" with distance $\epsilon$, need at least $O(1/\epsilon^2)$ points.
      - With 49 well-placed points you can guarantee that you have a "neighbor" within 0.1.
  - Consider the unit cube (d=3).
    - If want every location on to have a "neighbor" with distance $\epsilon$, need at least $O(1/\epsilon^3)$ points.
      - Wtih 512 well-placed points you can guarantee that you have a "neighbor" within 0.1.

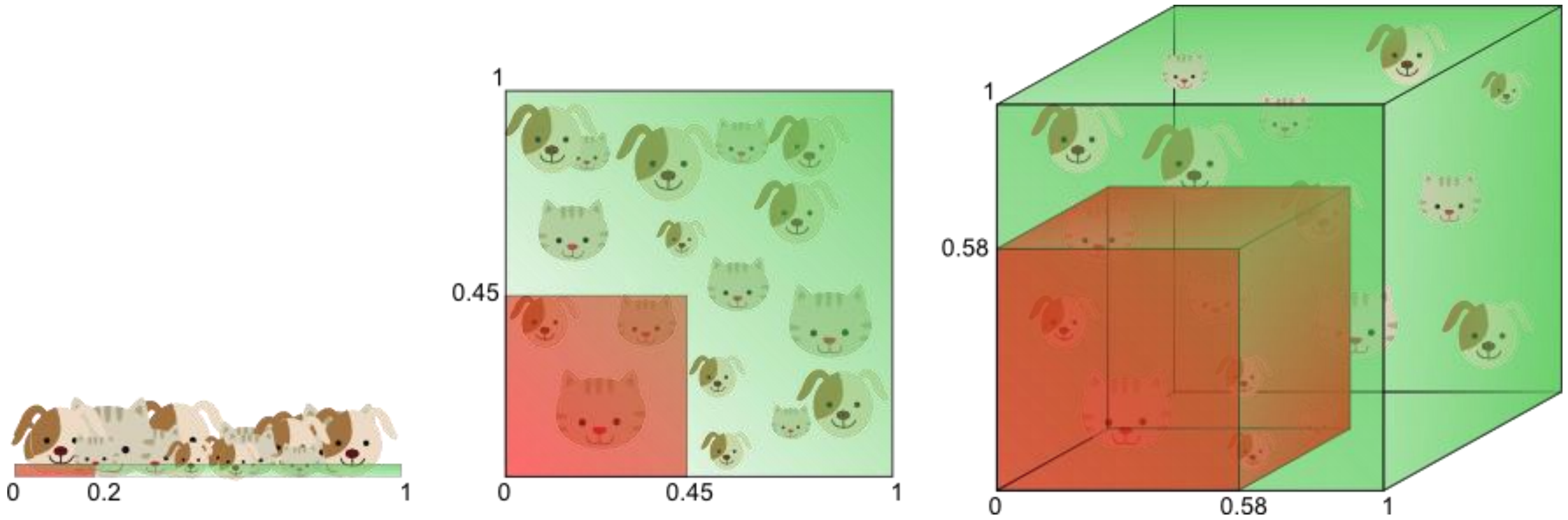


# Curse of Dimensionality

- "Curse of dimensionality": volume grows exponentially with dimension.
  - Consider the interval from 0 to 1 (d=1).
    - If want every location on to have a "neighbor" with distance $\epsilon$, need at least $O(1/\epsilon)$ points.
      - With 4 well-placed points you can guarantee that you have a "neighbor" within 0.1.
  - Consider the unit square (d=2).
    - If want every location on to have a "neighbor" with distance $\epsilon$, need at least $O(1/\epsilon^2)$ points.
      - With 49 well-placed points you can guarantee that you have a "neighbor" within 0.1.
  - Consider the unit cube (d=3).
    - If want every location on to have a "neighbor" with distance $\epsilon$, need at least $O(1/\epsilon^3)$ points.
      - With 512 well-placed points you can guarantee that you have a "neighbor" within 0.1.
  - Consider the unit hyper-cube (d=4).
    - If want every location on to have a "neighbor" with distance $\epsilon$, need at least $O(1/\epsilon^4)$ points.
      - With 6561 well-placed points you can guarantee that you have a "neighbor" within 0.1.

# Curse of Dimensionality

- Need exponentially more points to "fill" a high-dimensional space.
  - Need at least $O(1/\epsilon^d)$ points to guarantee "close" points exist everywhere.
  - In worst case, "nearest" neighbours in high-dimensions may be really far.




https://www.visiondummy.com/2014/04/curse-dimensionality-affect-classification/

# Curse of Dimensionality

- Need exponentially more points to “fill” a high-dimensional space.
  - Need at least O($1/\epsilon^d$) points to guarantee “close” points exist everywhere.
  - In worst case, “nearest” neighbours in high-dimensions may be really far.
- KNN is also problematic if features have very different scales.
  - Comparing a feature measured in grams vs one measured in kilograms.
    - Measurement in grams can have much more influence (values 1000 times larger).
- Nevertheless, KNN is really easy to use and often hard to beat!
  - Classes are often far apart, so neighbours do not need to be “close”.

https://www.visiondummy.com/2014/04/curse-dimensionality-affect-classification/

# Defining "Distance" with "Norms"

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

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

train example — test example — "$L_2$-norm"

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

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

    - The L2-norm is simply the length of the vector.

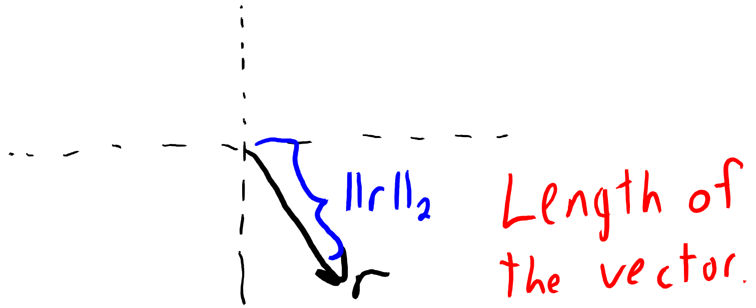
# L2-norm, L1-norm, and L∞-Norms.

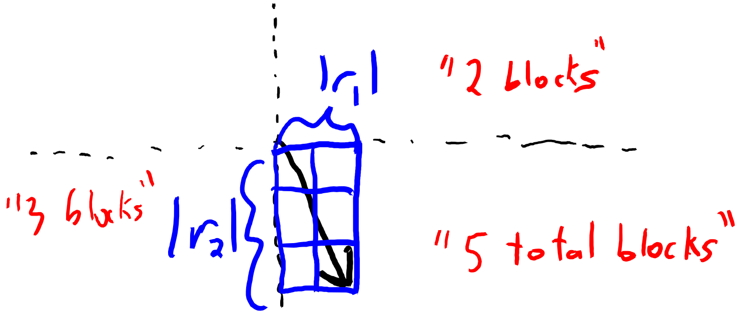
- The three most common norms: L2-norm, L1-norm, and L∞-norm.
  - Definitions of these norms with two-dimensions:

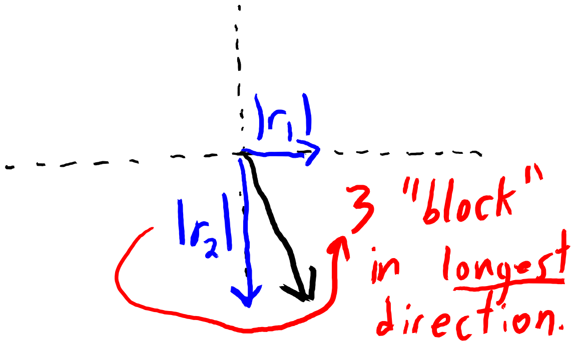
$L_2$ or "Euclidean" norm.
$$\|r\|_2 = \sqrt{r_1^2 + r_2^2}$$

$L_1$ or "Manhattan" norm:
$$\|r\|_1 = |r_1| + |r_2|$$

$L_\infty$ or "max" norm:
$$\|r\|_\infty = \max\{|r_1|, |r_2|\}$$




  - Definitions of these norms in d-dimensions.

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

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

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

[Infinite Series Video]

# 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

- Can define a "distance" between vectors by taking norm of difference:

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

$$= \|r - s\| \quad \text{"Euclidean distance"}$$


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

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

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

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

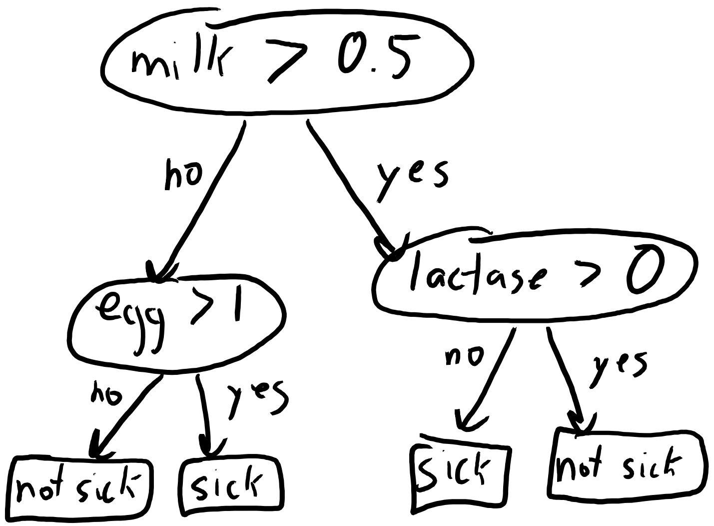
- Place different "weights" on large differences:
  - $L_1$: differences are equally notable.
  - $L_2$: bigger differences are more important (because of squaring).
  - $L_\infty$: only biggest difference is important.

# KNN Distance Functions

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

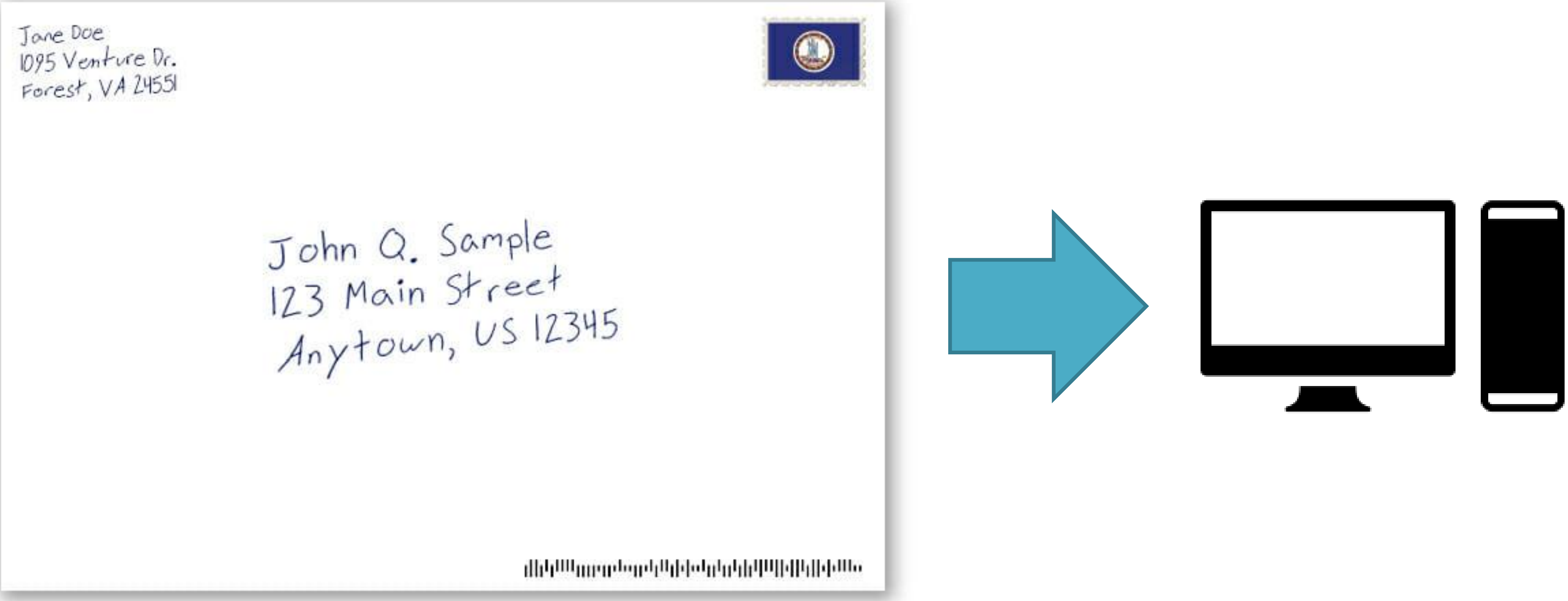



$$p(\text{sick} \mid \text{milk}, \text{egg}, \text{lactase})$$
$$\approx p(\text{milk} \mid \text{sick})\,p(\text{egg} \mid \text{sick})\,p(\text{lactase} \mid \text{sick})\,p(\text{sick})$$

(milk = 0.6, egg = 2, lactase = 0, ?) is close to
(milk = 0.7, egg = 2, lactase = 0, sick) so predict sick.

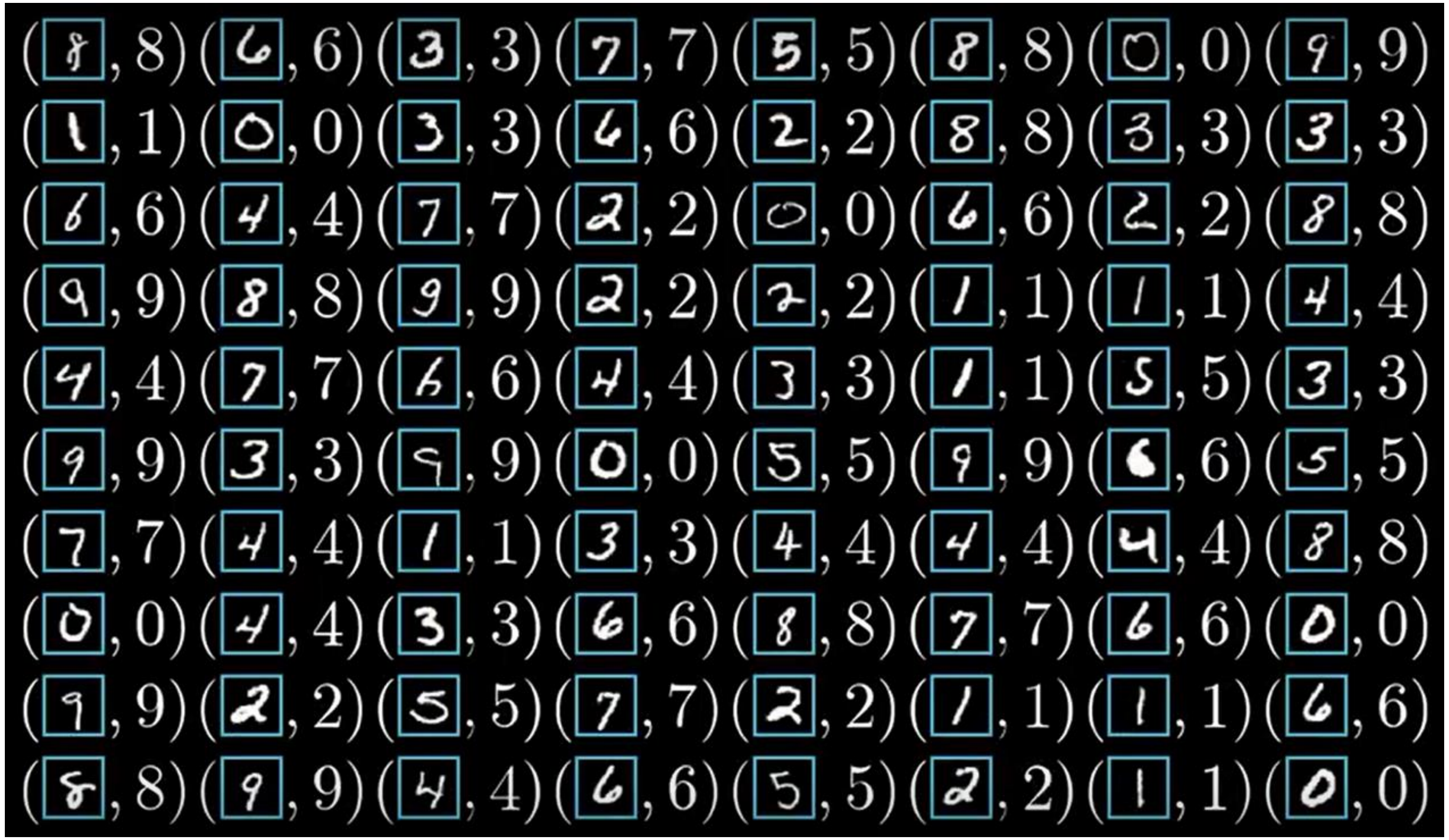
# Application: Optical Character Recognition

- How can we convert handwritten zip/postal codes to strings?



# Application: Optical Character Recognition

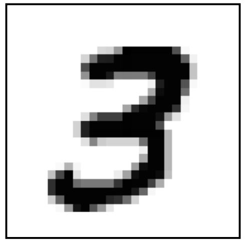
- To scan documents, we want to turn images into characters:
  - "Optical character recognition" (OCR).




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

# Application: Optical Character Recognition

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



→ "3"

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

$X =$

| (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 |

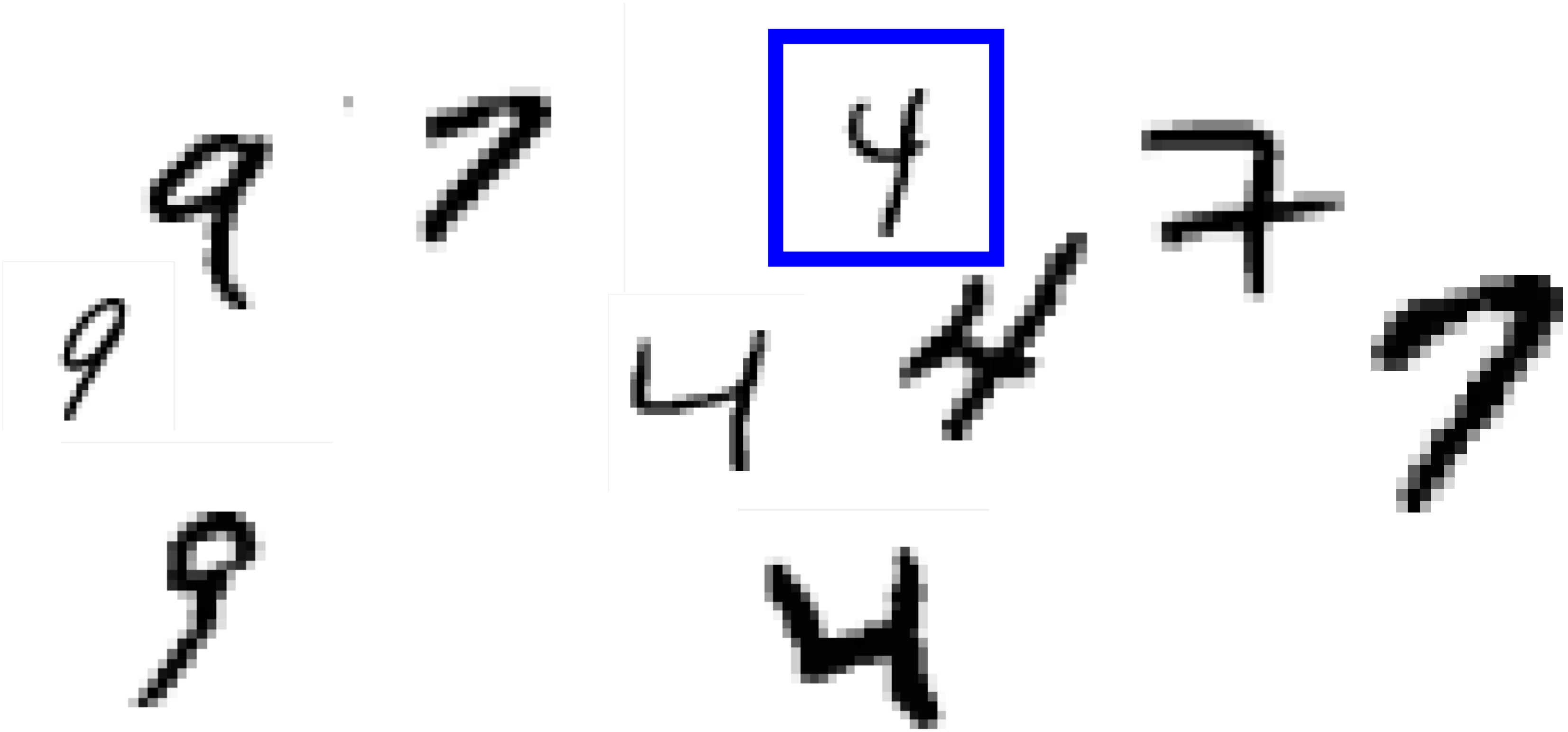
Each feature is grayscale intensity of one of the 784 pixels

⇒ $y =$

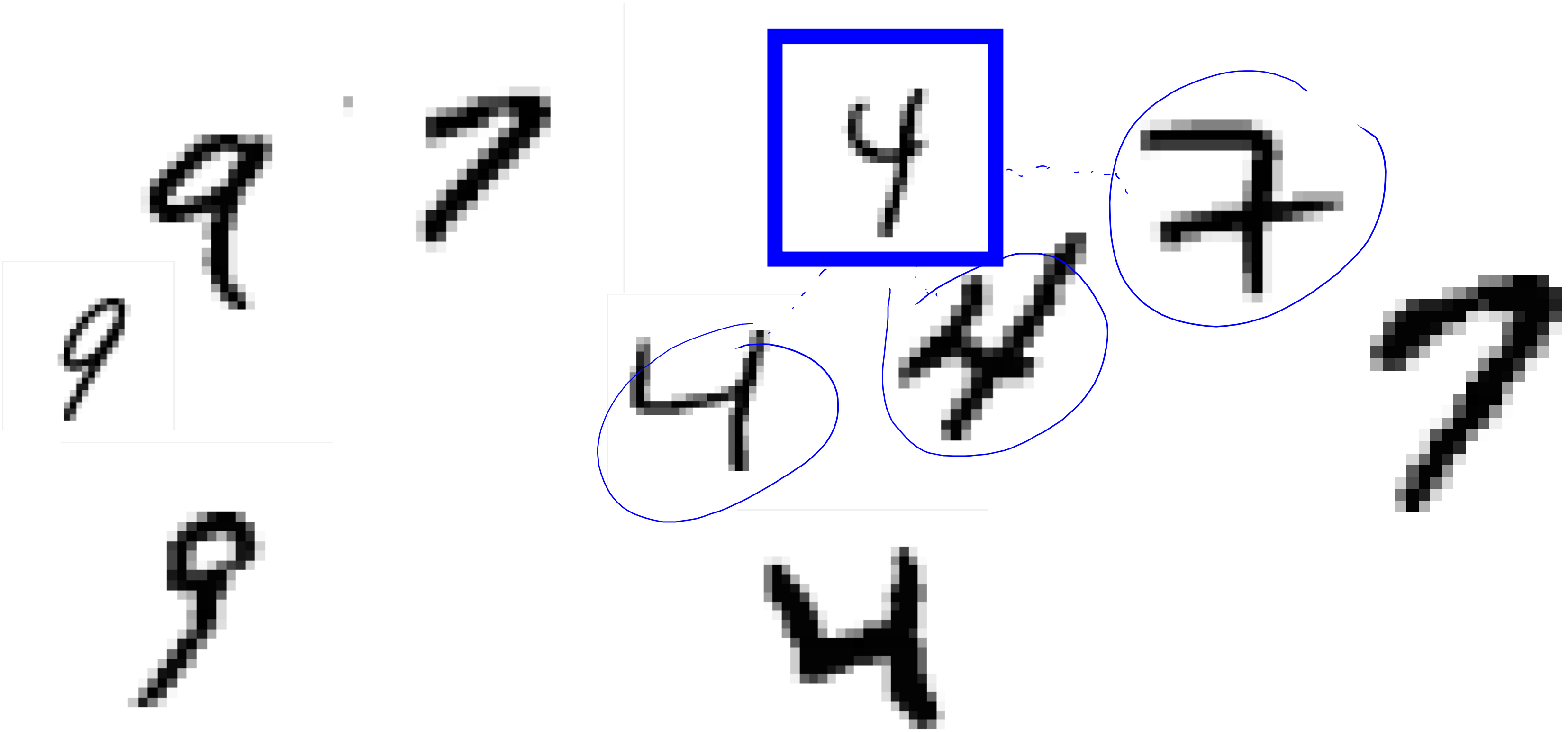
| char |
|---|
| 3 |
| 6 |
| 0 |
| 9 |

# KNN for Optical Character Recognition

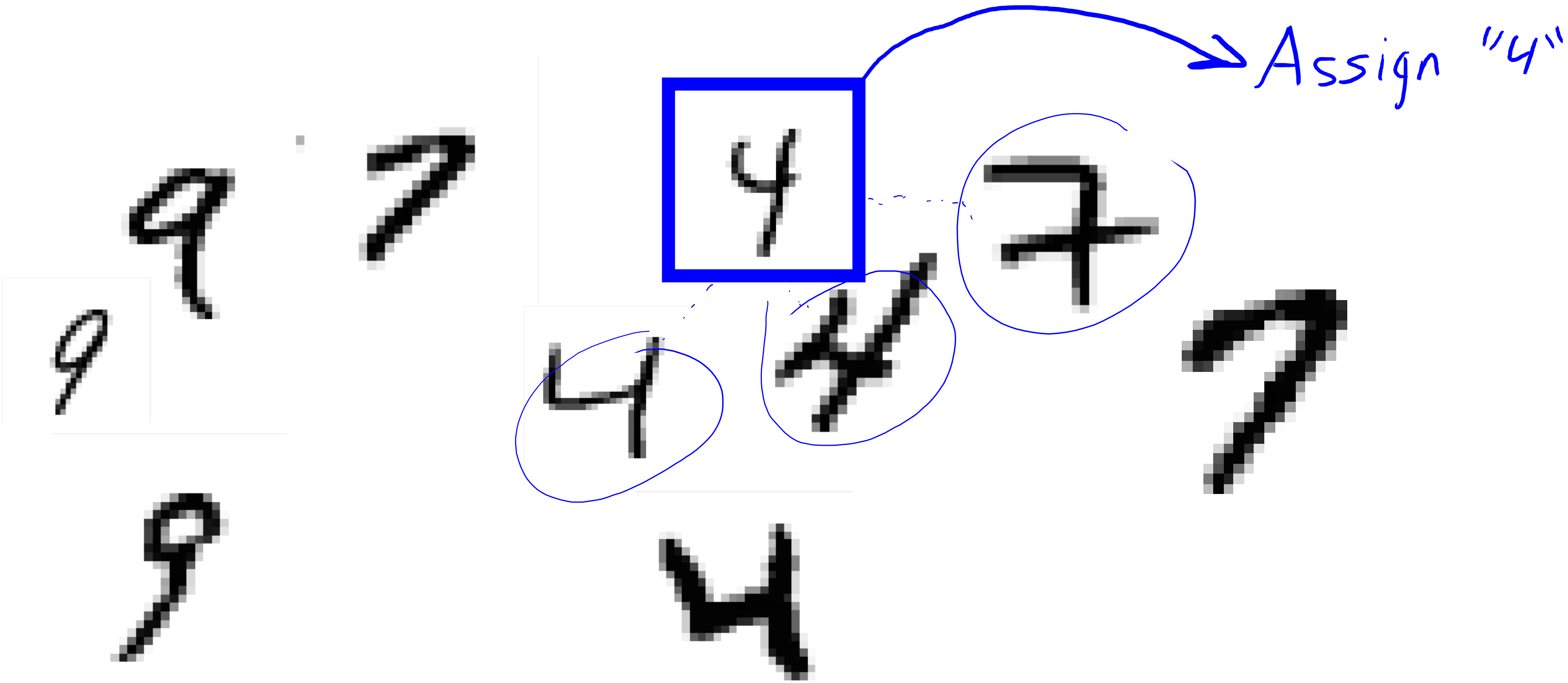
# KNN for Optical Character Recognition



# KNN for Optical Character Recognition



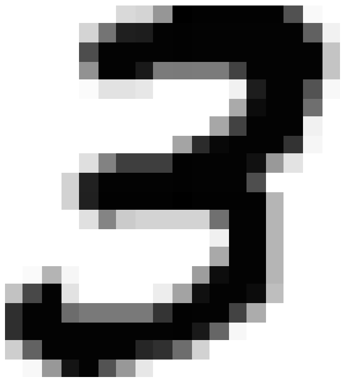
# KNN for Optical Character Recognition



# Human vs. Machine Perception

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

What we see:

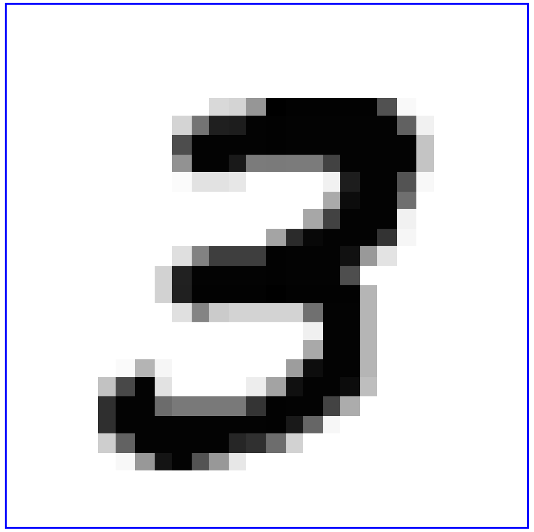


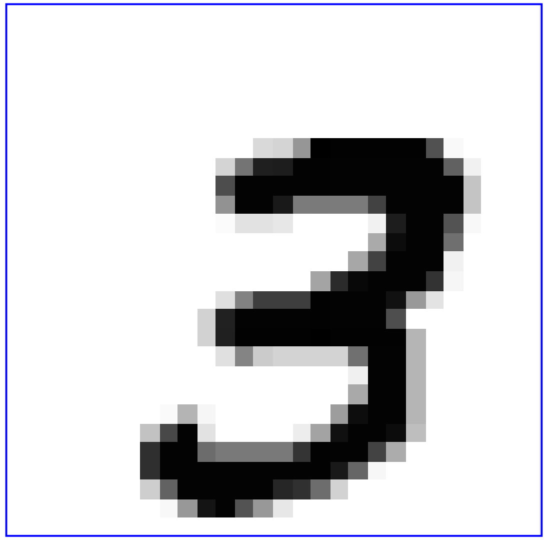
What the computer “sees”:

Actually, it’s worse:

# What the Computer Sees

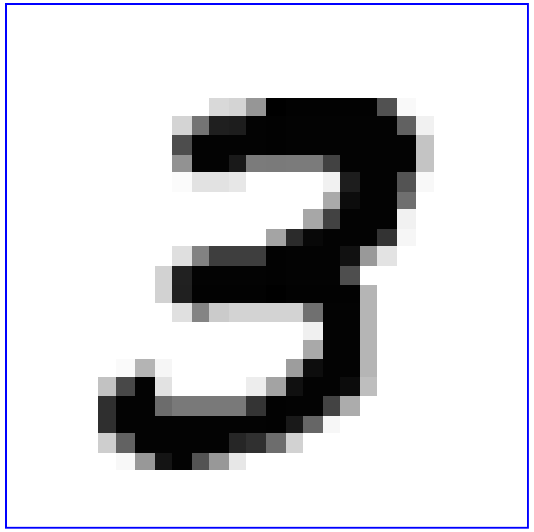
- Are these two images “similar”?

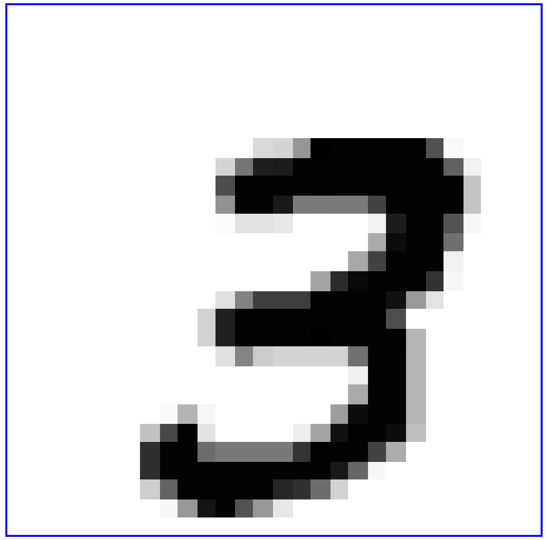




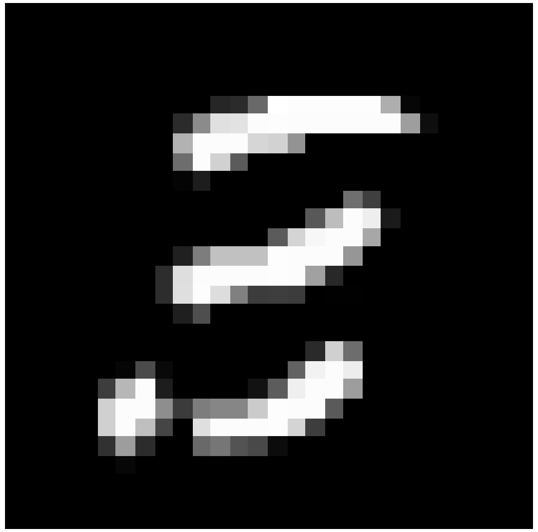
# What the Computer Sees

- Are these two images “similar”?





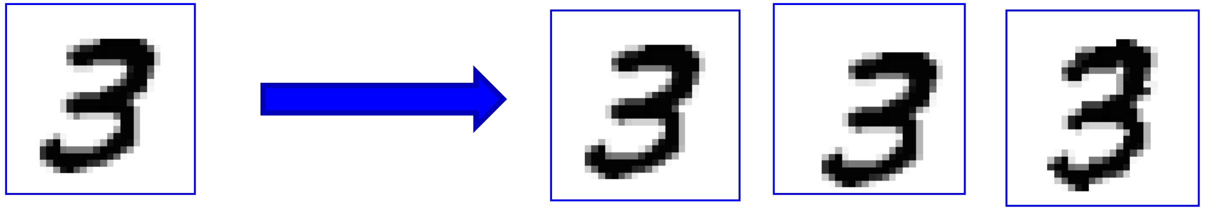
Difference:



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

# Encouraging Invariance with Data Augmentation

- May want classifier to be invariant to certain feature transforms.
  - Images: translations, small rotations, changes in size, mild warping,…
    - Recognize same signal in different-looking images.
- 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 use data augmentation.
  - Just add transformed versions of your training examples to the training set.
    - Make translated/rotate/resized/warped versions of training images, and add them to train set.

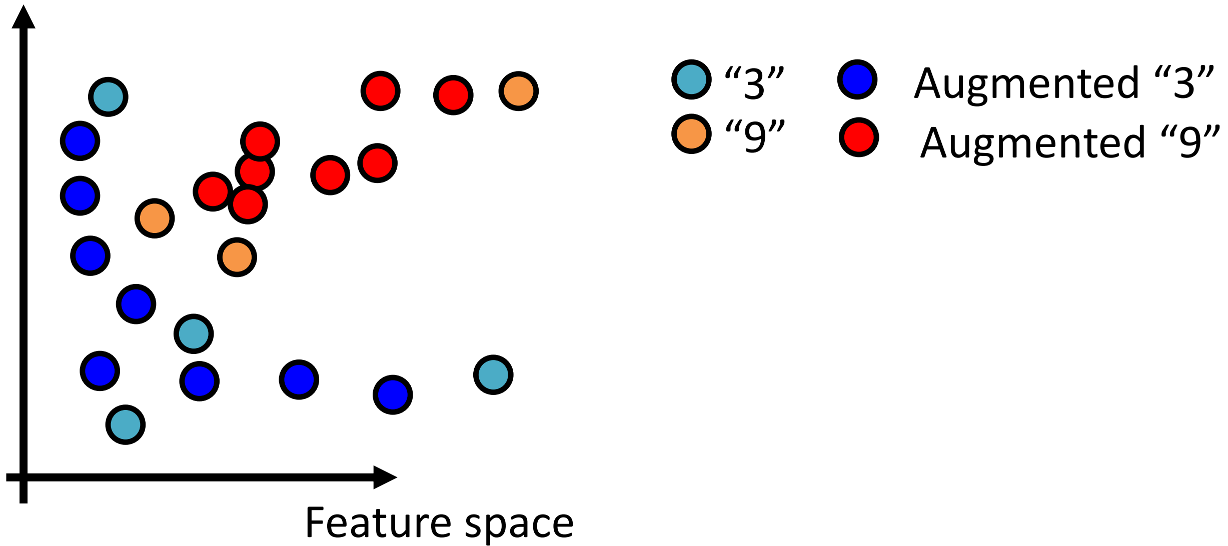


  - Crucial part of many successful vision systems.
  - Also really important for sound (translate, change volume, and so on).

# Encouraging Invariance with Data Augmentation

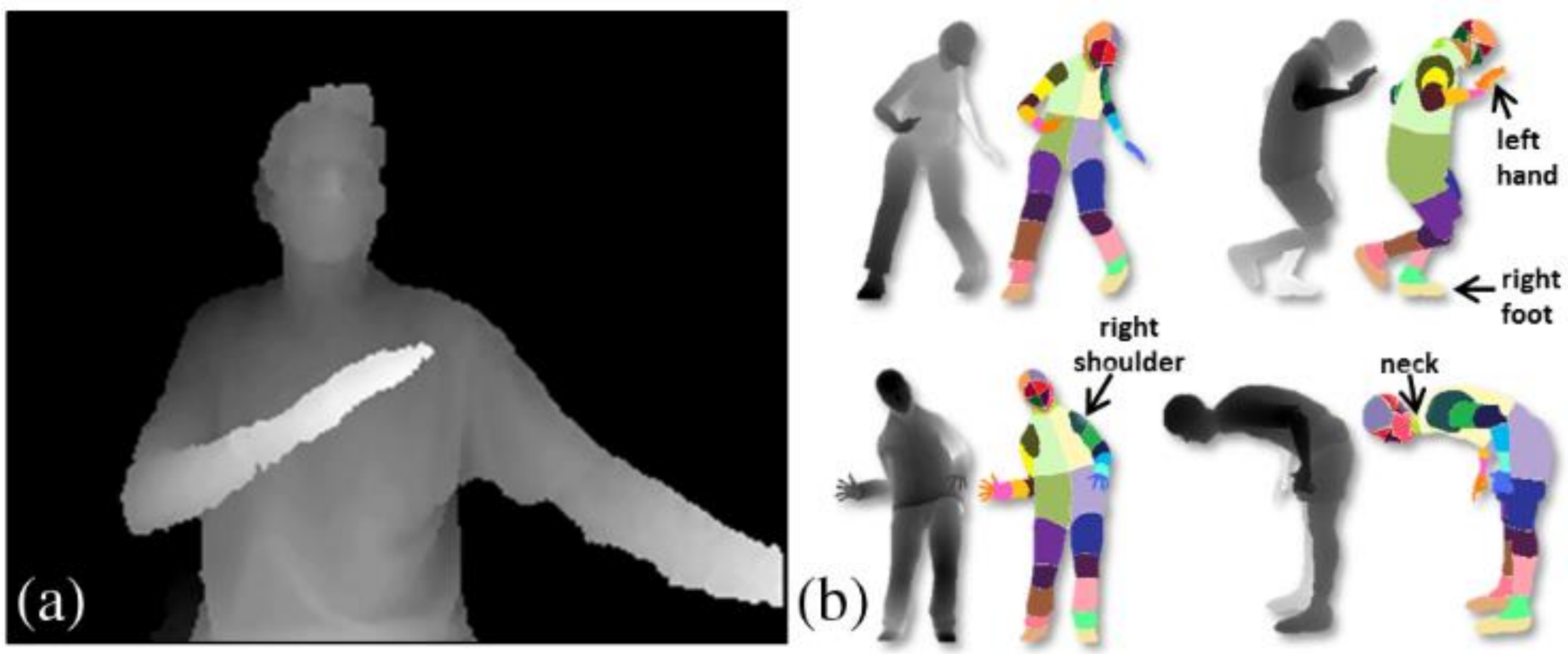
$$X = \underbrace{\begin{bmatrix} \; \end{bmatrix}}_{d}\ (n \text{ rows}) \qquad y = \underbrace{\begin{bmatrix} \; \end{bmatrix}}_{1}\ (n \text{ rows}) \;\rightarrow\; X_{aug} = \underbrace{\begin{bmatrix} X \\ \cdots \\ X_+ \end{bmatrix}}_{d}\ (n,\ n_+ \text{ rows}) \qquad y_{aug} = \underbrace{\begin{bmatrix} y \\ \cdots \\ y_+ \end{bmatrix}}_{1}\ (n,\ n_+ \text{ rows})$$

- Augmentation helps “fill the space”:



# 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?

# 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).

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

# Supervised Learning Step

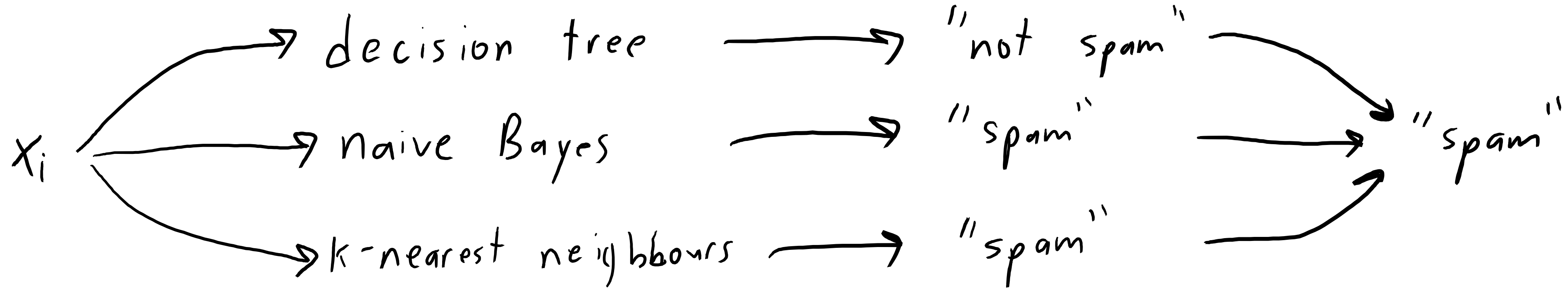
- ALL steps are important, but we'll focus on the learning step.
- Do we have any classifiers that are accurate and run in real time?
  - Decision trees and naïve Bayes are fast, but often not very accurate.
  - KNN is often accurate, but not very fast.
- Deployed system uses an ensemble method called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

# Ensemble Methods

- Ensemble methods 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 methods often have higher accuracy than input classifiers.

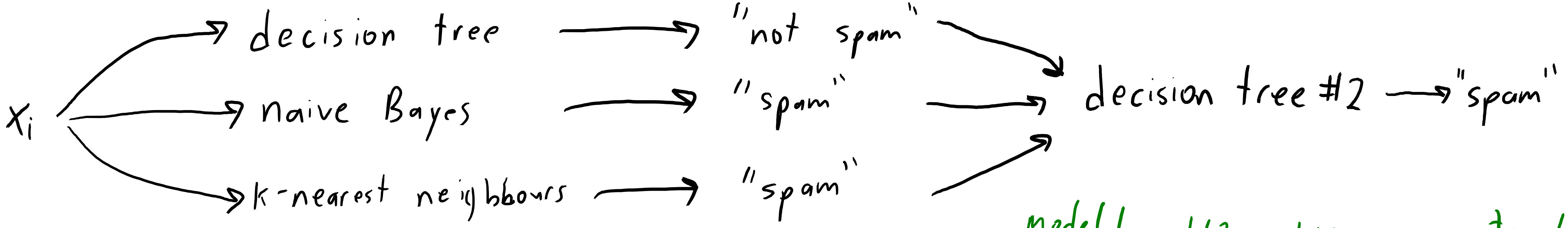
# Ensemble Method Example: Voting

- Ensemble methods use predictions of a set of models.
  - For example, we could have:
    - 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.

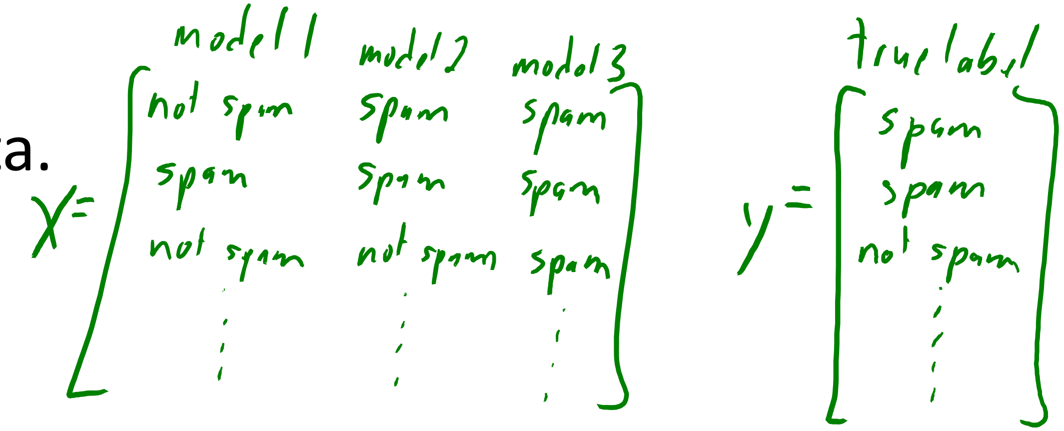


# Digression: Stacking

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




- Can tune second classifier with 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.



# 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).
    - You can derive the precise probability with binomial probabilities.

- 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.
    - But accuracy does not have to the same across classifiers ("weak" classifiers can help "strong" ones).

# 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 will not be completely independent.
    - For a variety of reasons (incorrect labels, classifiers use same training set, and so on).

# 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.

# Random Forests

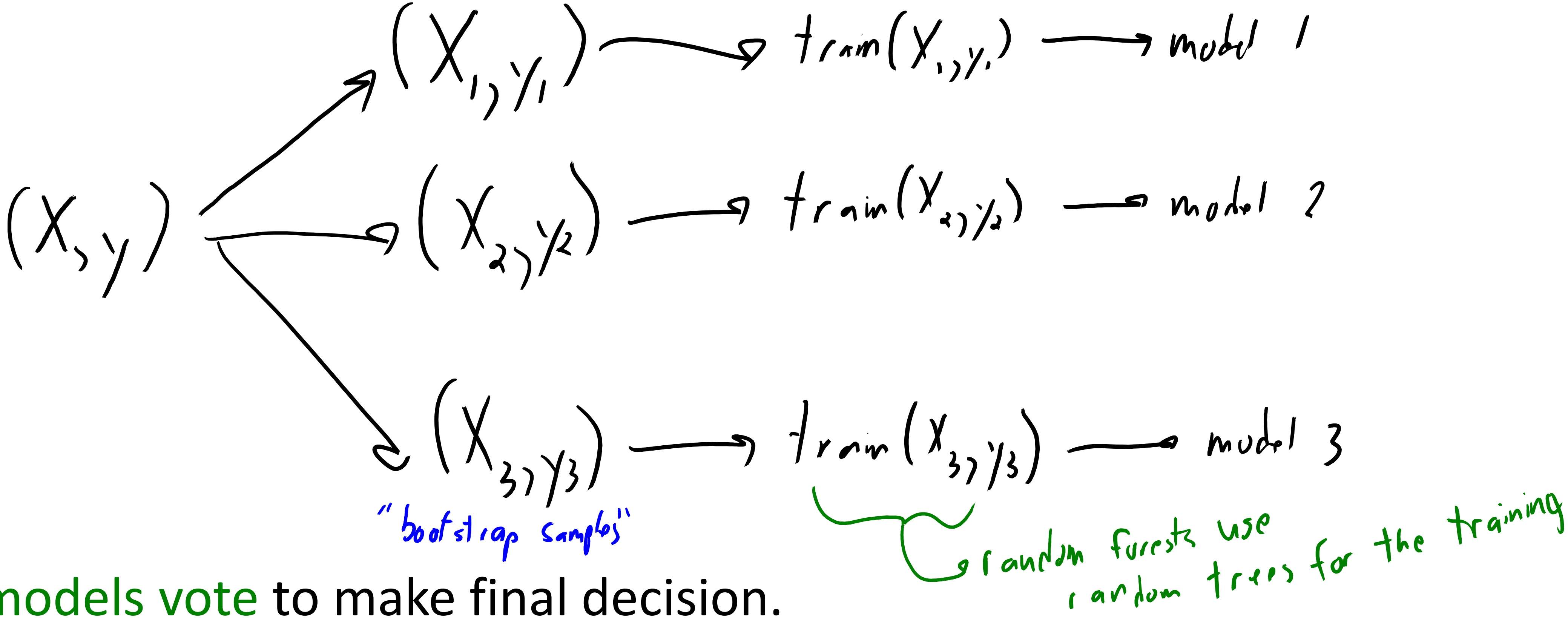
- Random forests take vote from a set of deep decision trees.
  - Tend to be one of the best “out of the box” classifiers.
    - Often close to the best performance of any method on the first run.
  - And predictions are very fast.

- 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: a way to generate different “versions” of your dataset.
  - Random trees: a way to grow decision trees incorporating randomness.

# Overview of Random Forests

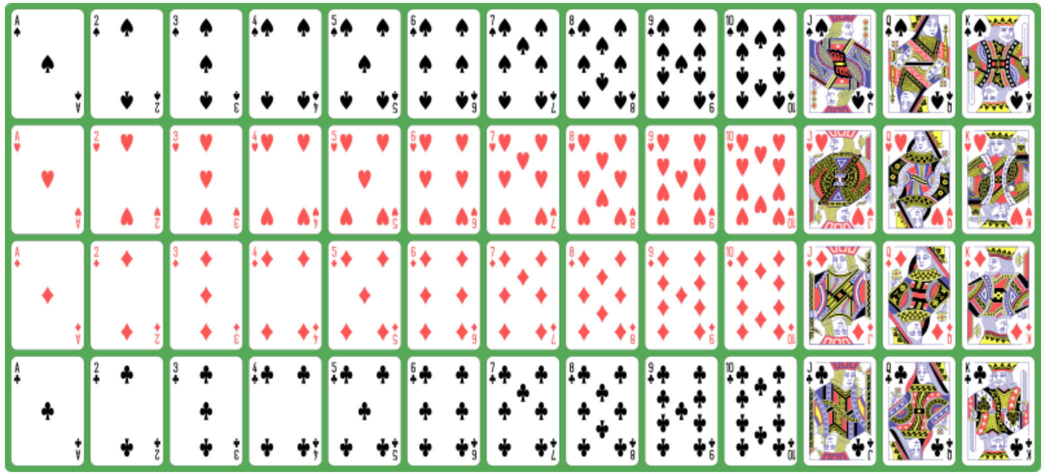
- Random forests train on different "bootstrap samples" of your dataset:



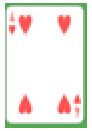

- And models vote to make final decision.
  - The hope is that the "boostrap samples" make errors more independent.

# Bootstrap Sampling

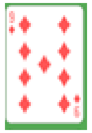
- Start with a standard deck of 52 cards:



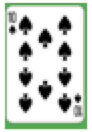
  1. Sample a random card:
     (put it back and re-shuffle)



  2. Sample a random card:
     (put it back and re-shuffle)

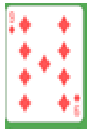


  3. Sample a random card:
     (put it back and re-shuffle)

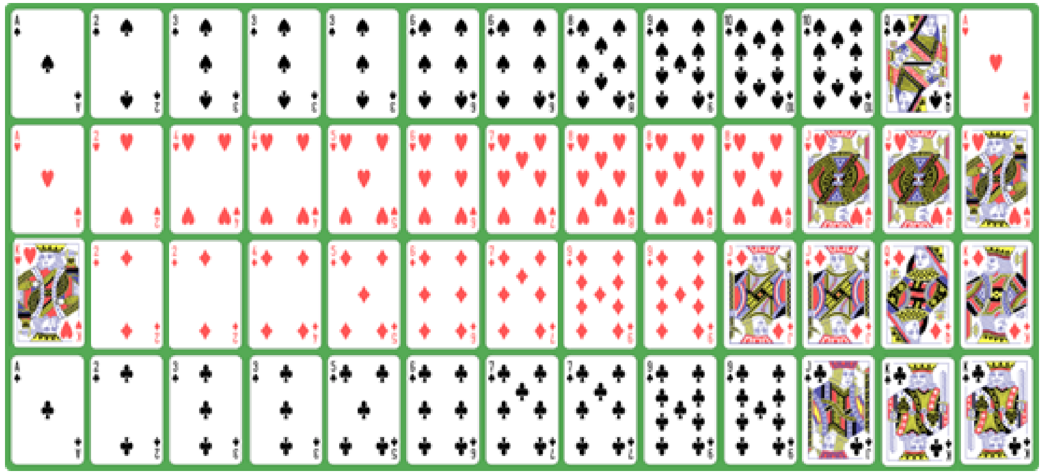


  – …

  52. Sample a random card:
      (which may be a repeat)

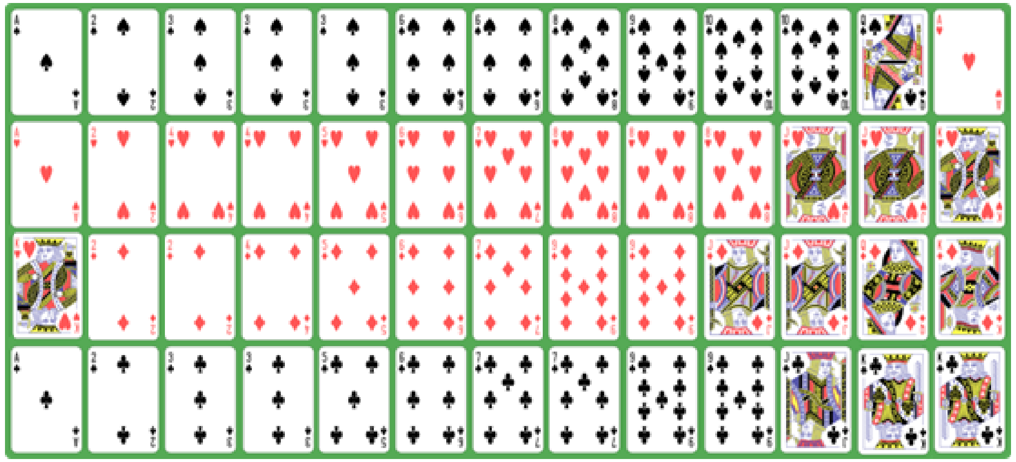


- Makes a new deck of the 52 samples:



# 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.
  - Bootstrap sampling is a general technique that is used in many settings:
    - Sample 'n' examples with replacement from your set of size 'n'.
    - Repeat this several times, and compute some statistic on each bootstrap sample.
      - Gives you an idea of how the statistic varies as you vary the data.

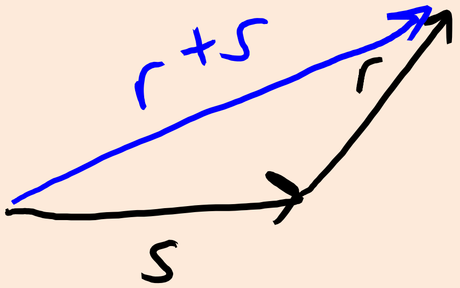
# Summary

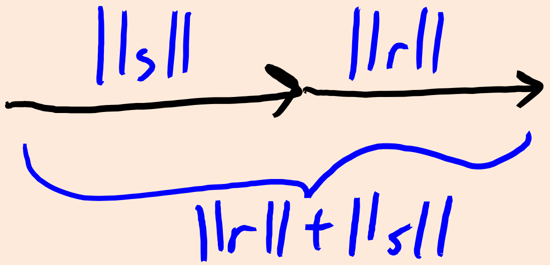
- Curse of dimensionality:
  - Number of points to “fill” a space grows exponentially with dimension.
- Data augmentation:
  - Add transformed data to be invariant to transformations that preserve label.
- Ensemble methods take multiplier classifiers as inputs.
- Voting ensemble method:
  - Improves predictions of multiple classifiers if errors are independent.
- Bootstrap sampling:
  - Generating a new dataset, by sampling ‘n’ examples with replacement.

- Next time:
  - We start unsupervised learning.

# 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:

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

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 $L_\infty$-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 $L_\infty$-norm is the limit as 'p' goes to $\infty$.

- 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:

$$p(\text{selected at least once in 'n' trials})$$
$$= 1 - p(\text{not selected in any of 'n' trials})$$
$$= 1 - (p(\text{not selected in one trial}))^n \quad \text{(trials are } \underline{\text{independent}}\text{)}$$
$$= 1 - (1 - 1/n)^n \quad \left(\text{prob} = \frac{n-1}{n} \text{ for choosing any of the } n-1 \text{ } \underline{\text{other}} \text{ samples}\right)$$
$$\approx 1 - 1/e \quad \left((1-1/n)^n \rightarrow e^{-1} \text{ as } n \rightarrow \infty\right)$$
$$\approx 0.63$$

# Why Averaging Works

- Consider ‘k’ independent classifiers, whose errors have a variance of $\sigma^2$.
- If the errors are IID, the variance of the vote is $\sigma^2/k$.
  - 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

- Here is an 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.