

# CPSC 340: Machine Learning and Data Mining

Ensemble Methods

Fall 2017

# Admin

- Welcome to the course!
- Assignment 0:
  - 2 late days to hand it in tonight, 0 after that.
- **Assignment 1** is due next Friday.

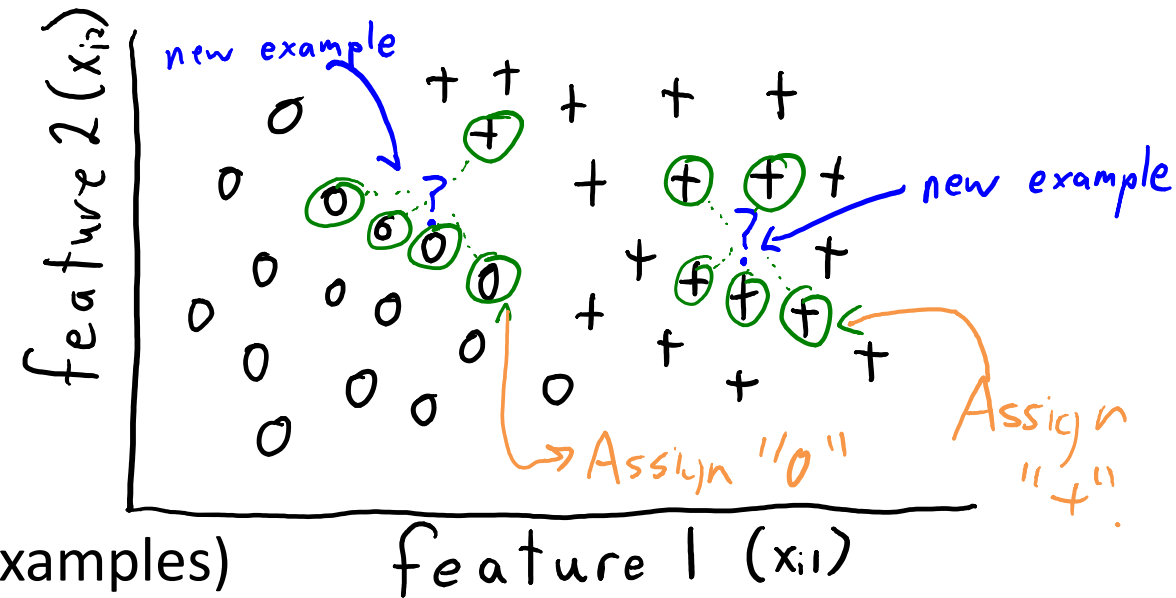
# Last Time: K-Nearest Neighbours (KNN)

- 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  $y_i$ .

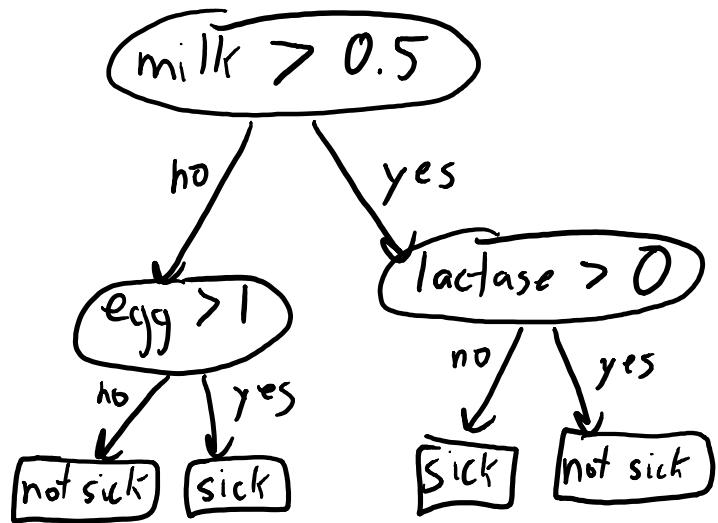
- Lazy learning:
  - To "train" you just store X and y.

- 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.



# Decision Trees vs. Naïve Bayes vs. KNN

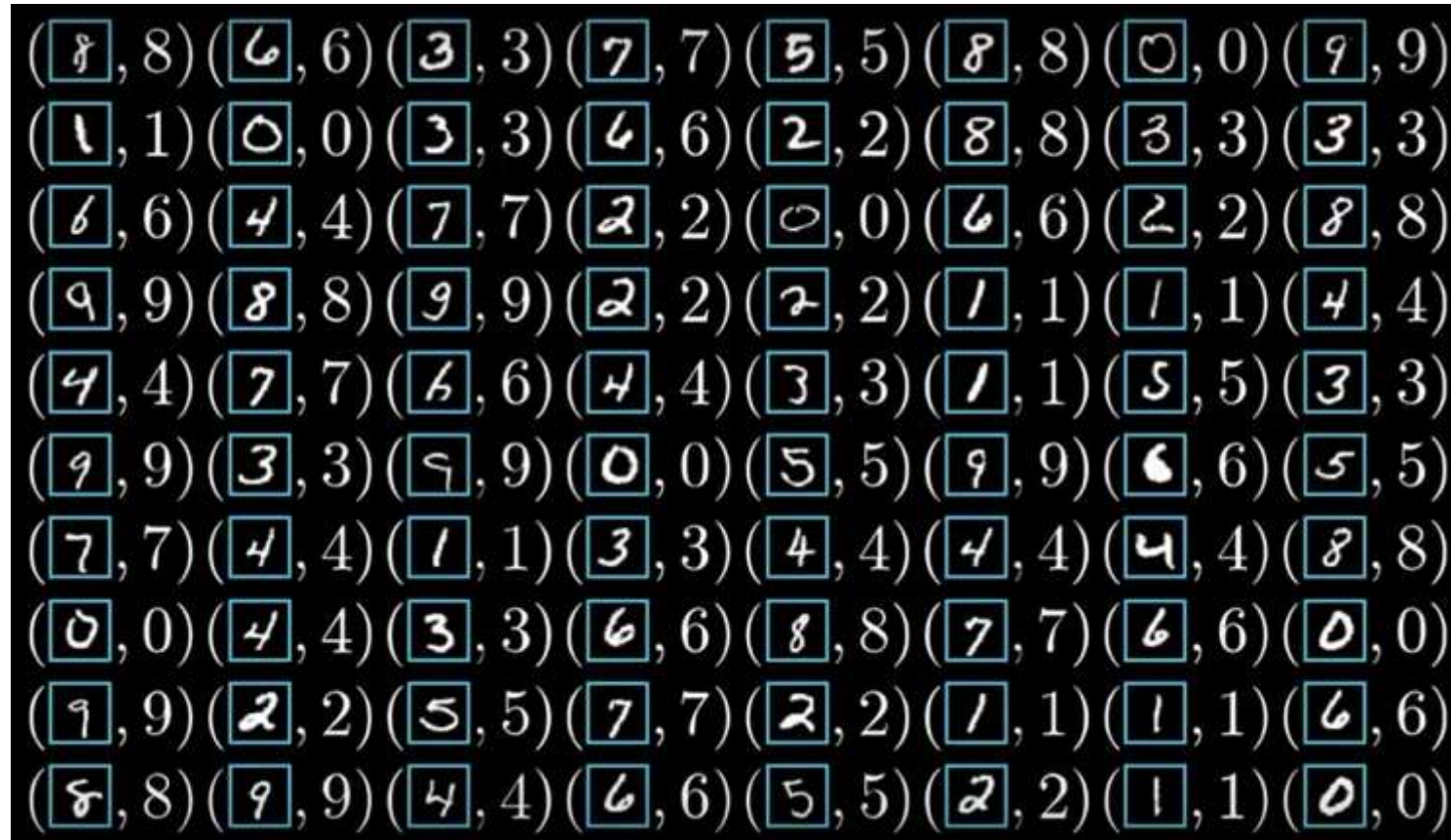


$$p(\text{sick} \mid \text{milk}, \text{egg}, \text{lactase}) \\ \approx p(\text{milk} \mid \text{sick}) p(\text{egg} \mid \text{sick}) p(\text{lactase} \mid \text{sick}) p(\text{sick})$$

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

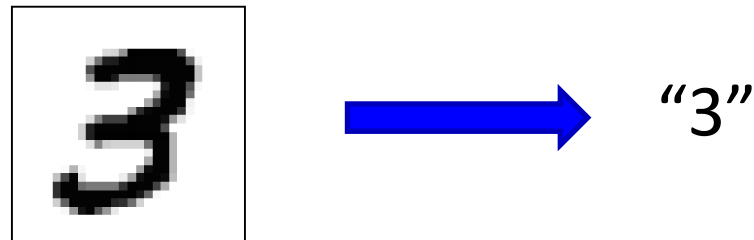
# Application: Optical Character Recognition

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

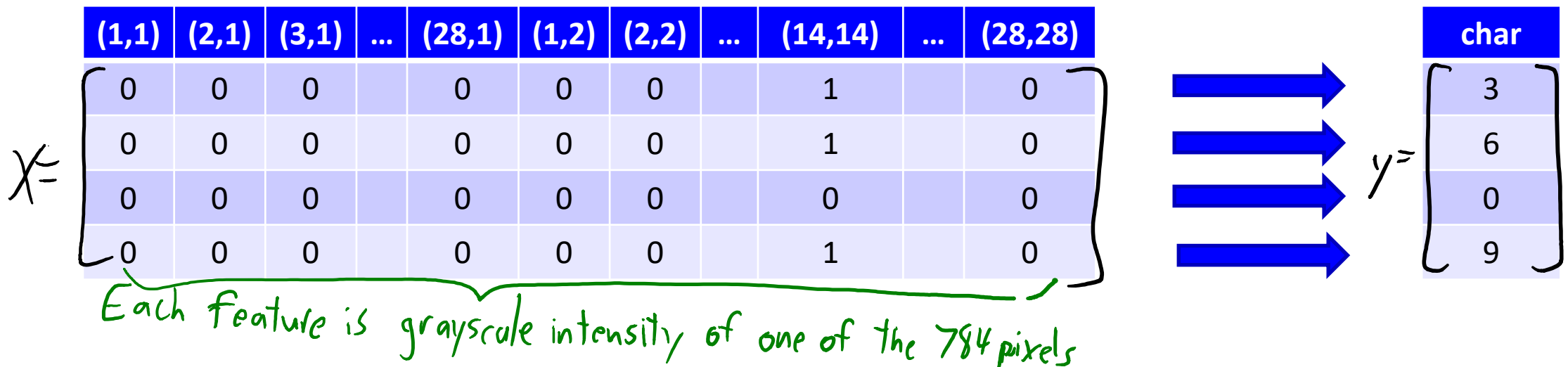


# Application: Optical Character Recognition

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



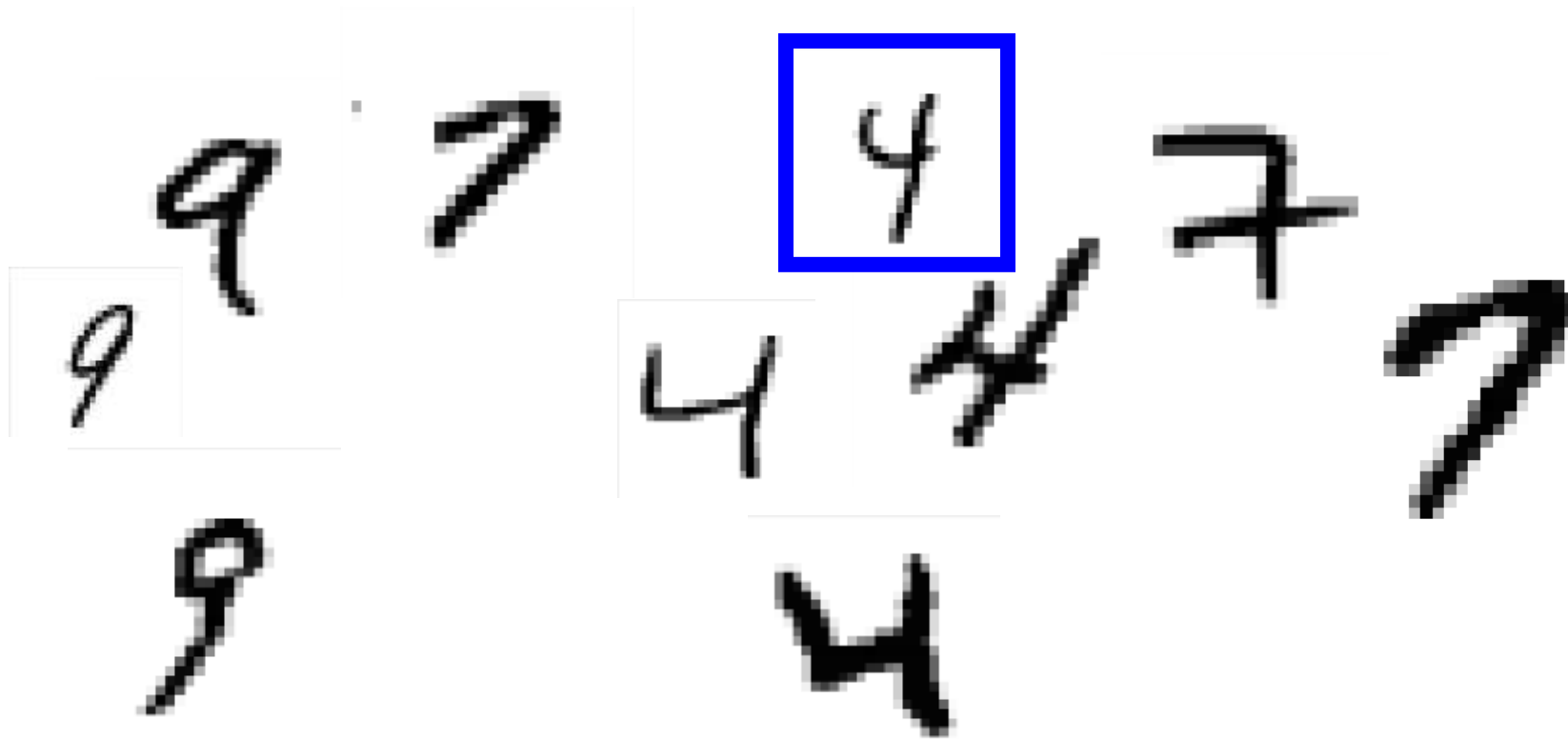
- Turning this into a **supervised learning** problem (with 28 by 28 images):



# KNN for Optical Character Recognition

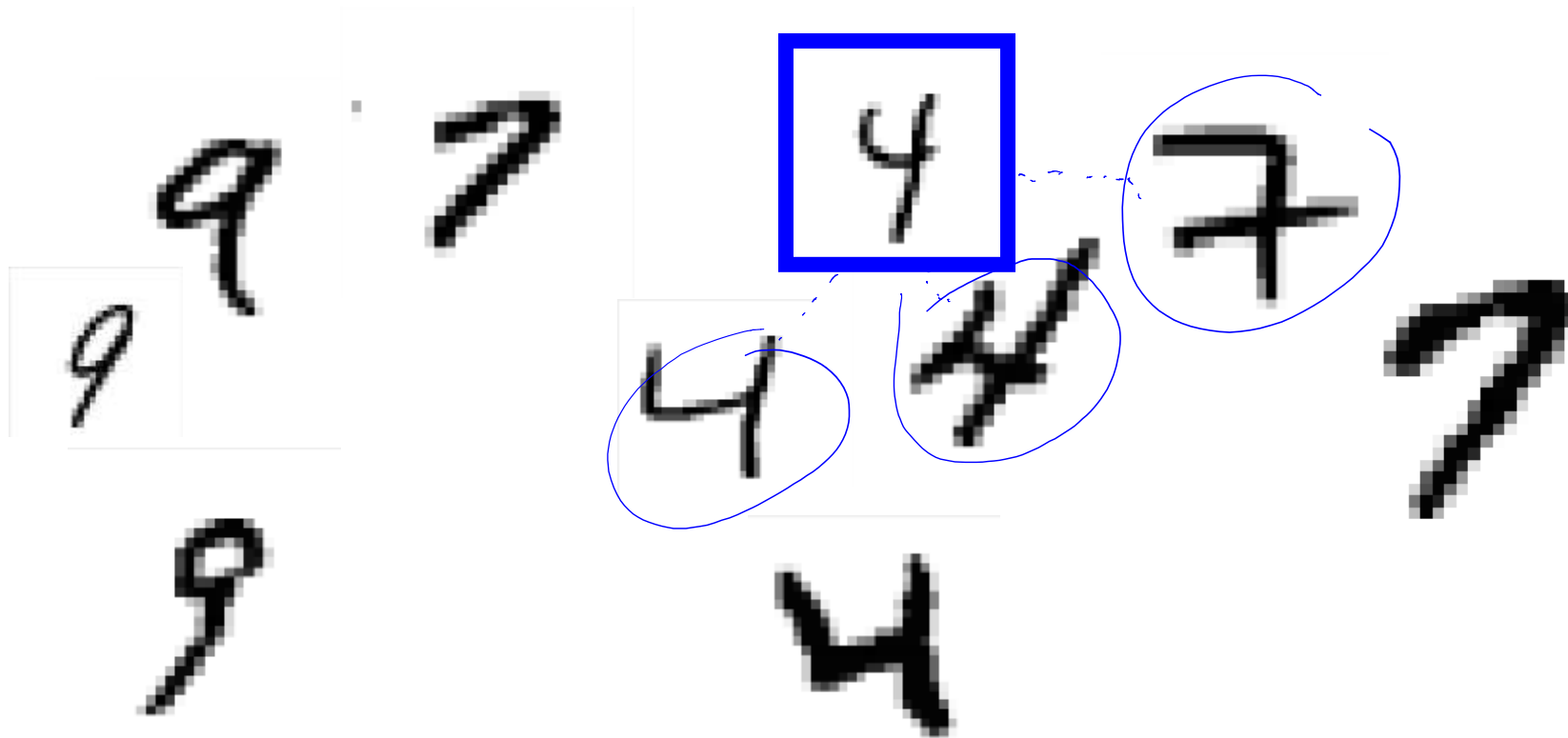


# KNN for Optical Character Recognition

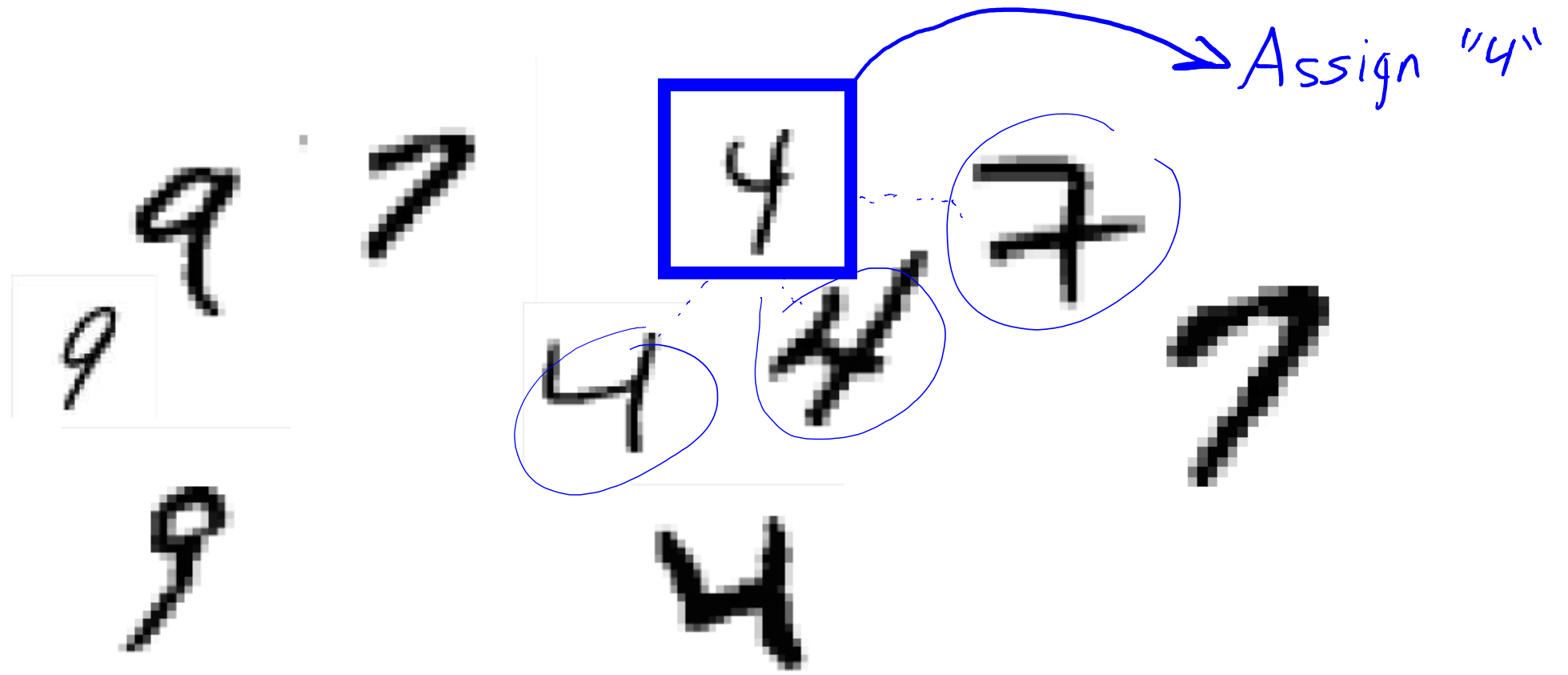




# KNN for Optical Character Recognition



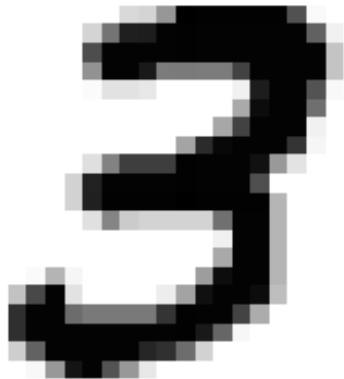
# KNN for Optical Character Recognition



# Human vs. Machine Perception

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

What we see:



What the computer “sees”:

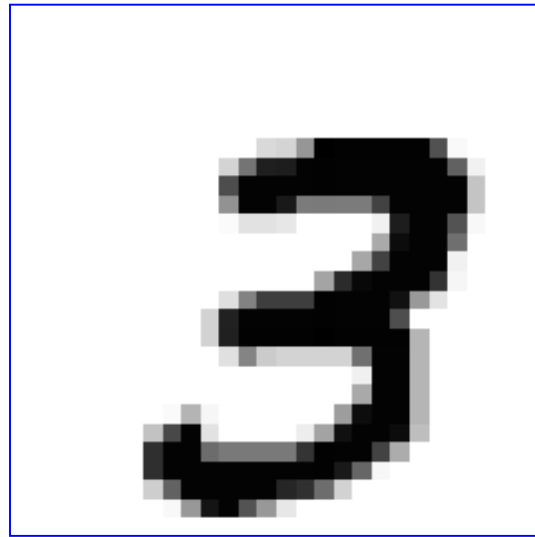
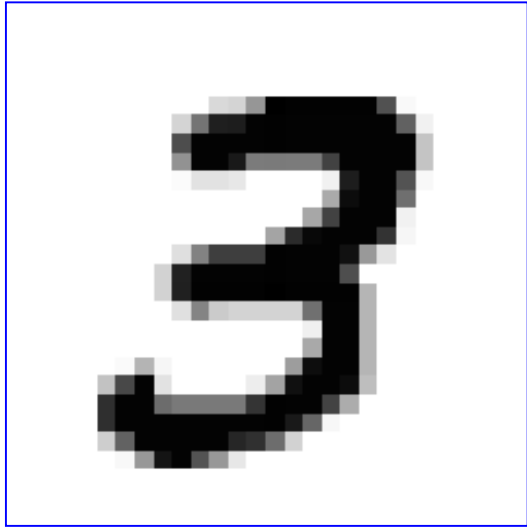


Actually, it's worse:



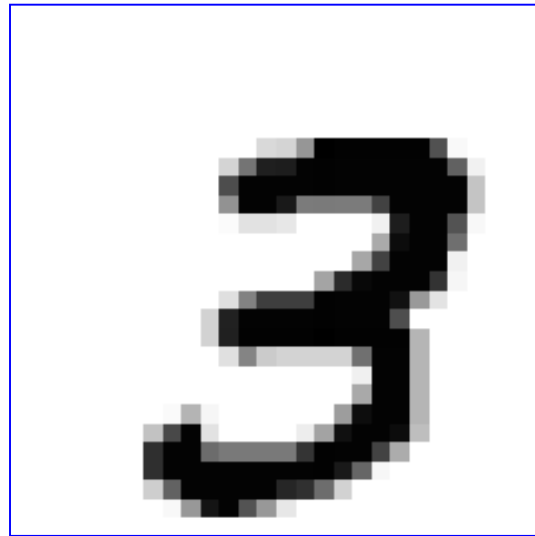
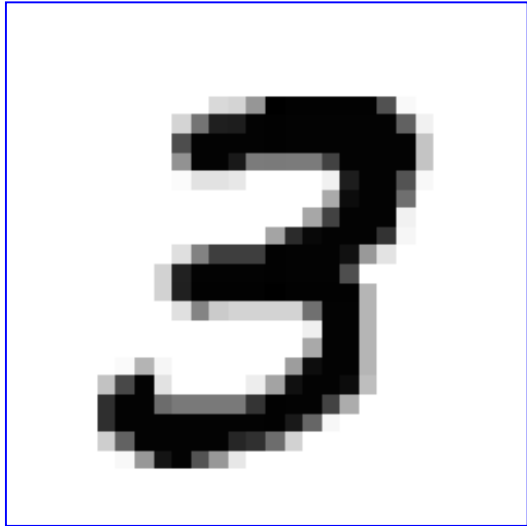
# What the Computer Sees

- Are these two images “similar”?

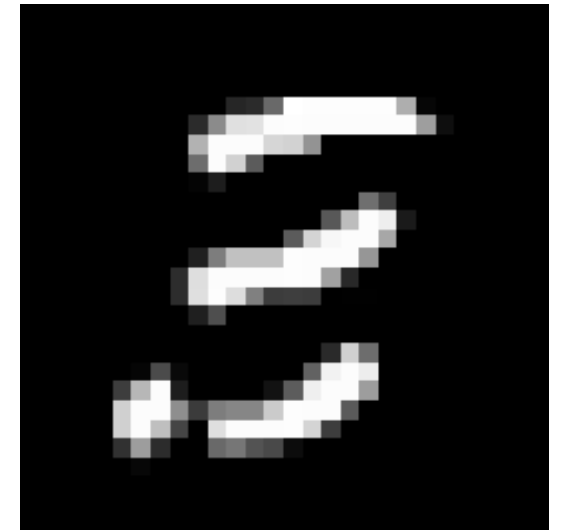


# What the Computer Sees

- Are these two images “similar”?



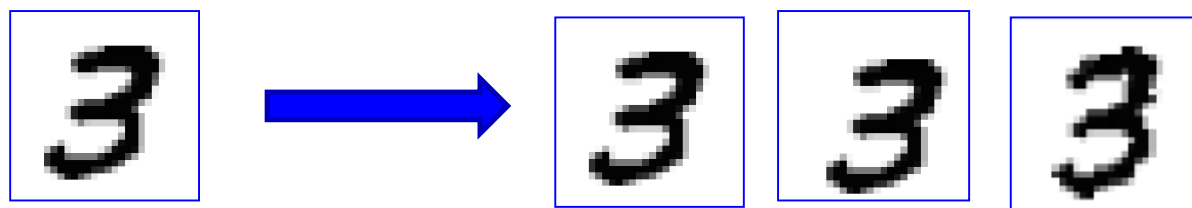
Difference:



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

# Encouraging Invariance

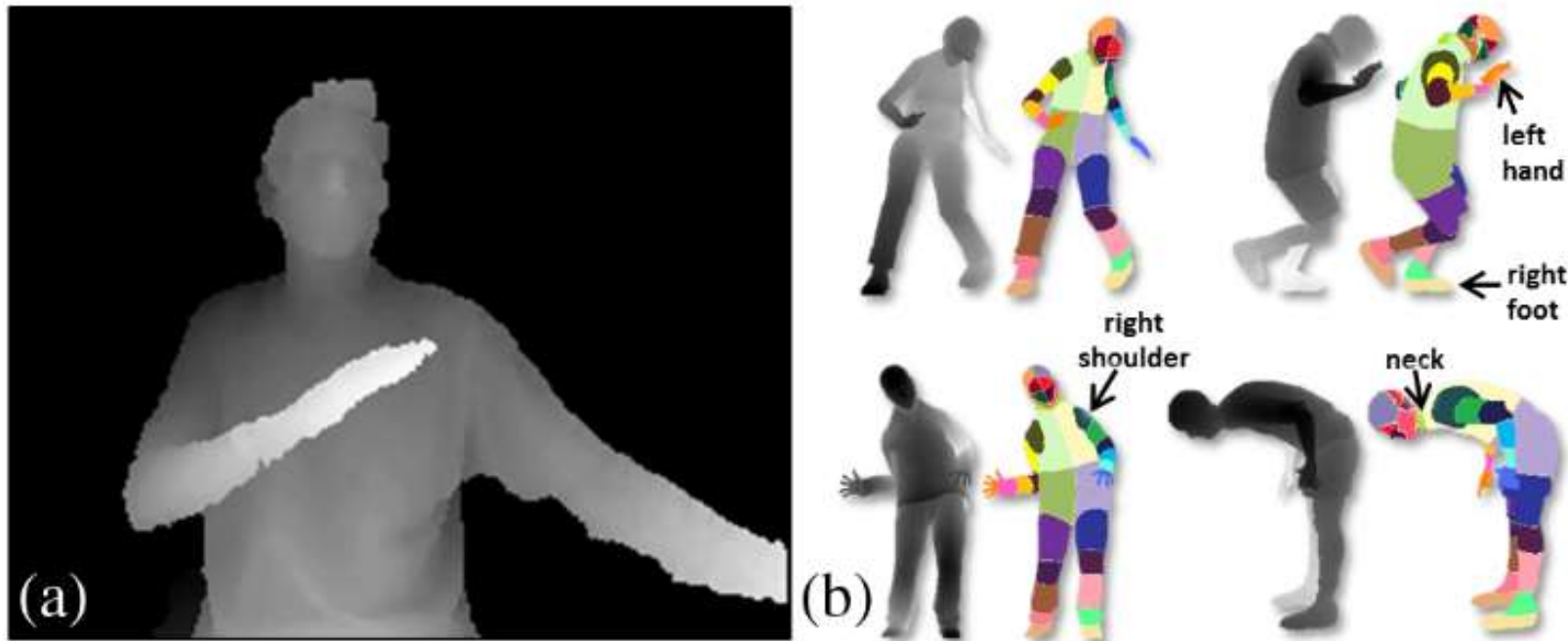
- May want classifier to be invariant to certain feature transforms.
  - Images: translations, small rotations, changes in size, mild warping,...
- The **hard/slow way** is to modify your distance function:
  - Find neighbours that require the ‘smallest’ transformation of image.
- The **easy/fast way**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



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

# Application: Body-Part Recognition

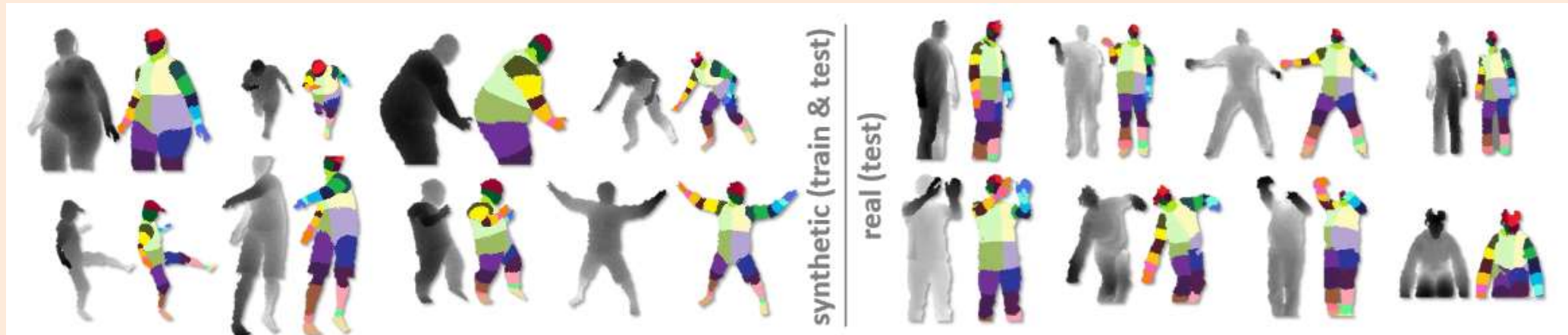
- Microsoft Kinect:
  - Real-time recognition of 31 body parts from laser depth data.



- How could we write a program to do this?

# 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

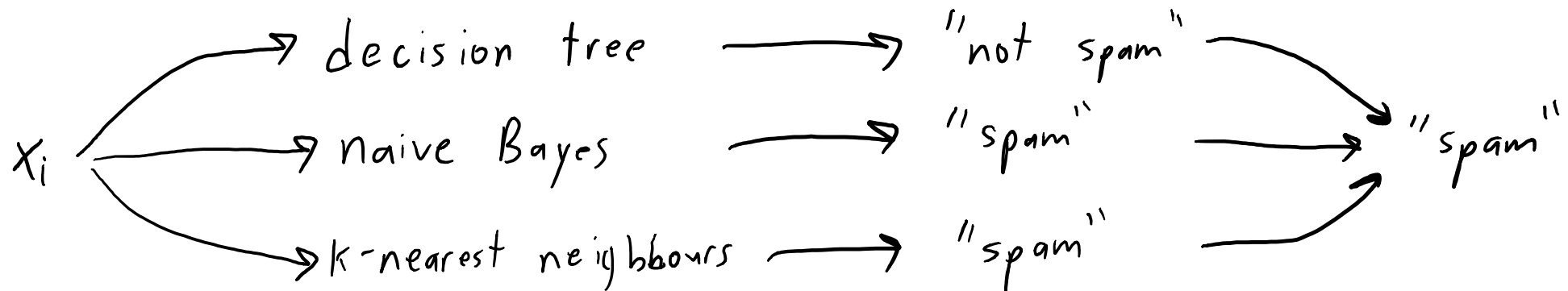
- Ensemble methods are **classifiers that have classifiers as input**.
  - Also called “meta-learning”.
- They have the best names:
  - Averaging.
  - Boosting.
  - Bootstrapping.
  - Bagging.
  - Cascading.
  - Random Forests.
  - Stacking.
- **Ensemble methods often have higher accuracy** than input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

# Ensemble Methods

- Remember the fundamental trade-off:
  1.  $E_{\text{train}}$ : How small you can make the training error.
  - vs.
  2.  $E_{\text{approx}}$ : How well training error approximates the test error.
- Goal of ensemble methods is that meta-classifier:
  - Does much better on one of these than individual classifiers.
  - Doesn't do too much worse on the other.
- This suggests two types of ensemble methods:
  1. **Boosting**: improves training error of classifiers with high  $E_{\text{train}}$ .
  2. **Averaging**: improves approximation error of classifiers with high  $E_{\text{approx}}$ .

# Averaging

- Input to **averaging** is the predictions of a set of models:
  - Decision trees make one prediction.
  - Naïve Bayes makes another prediction.
  - KNN makes another prediction.
- Simple **model averaging**:
  - Take the **mode of the predictions** (or average if probabilistic).



# Why can Averaging Work?

- Consider 3 binary classifiers, each **independently correct** with probability 0.80:
  - $P(\text{all 3 right}) = 0.8^3 = 0.512$ .
  - $P(\text{2 rights, 1 wrong}) = 3 * 0.8^2(1-0.8) = 0.384$ .
  - $P(\text{1 right, 2 wrongs}) = 3 * (1-0.8)^2 * 0.8 = 0.096$ .
  - $P(\text{all 3 wrong}) = (1-0.8)^3 = 0.008$ .
- So **ensemble is right with probability 0.896** (which is  $0.512+0.384$ ).
- Notes:
  - For averaging to work, **classifiers need to be at least somewhat independent**.
  - You also want the probability of being right to be  $> 0.5$ , otherwise it will do much worse.
  - Probabilities also shouldn't be too different (otherwise, it might be better to take most accurate).

# Averaging

- 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).

# Stacking

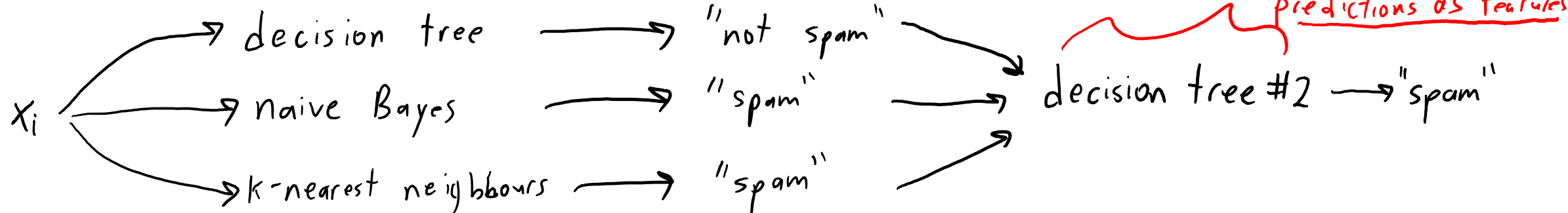
- Input to averaging is the predictions of a set of models:

- Decision trees make one prediction.
- Naïve Bayes makes another prediction.
- KNN makes another prediction.

	model 1	model 2	model 3	true label
$X =$	not spam	spam	spam	spam
	spam	spam	spam	spam
	not spam	not spam	spam	not spam
	⋮	⋮	⋮	⋮

- **Stacking:**

- Fit **another classifier** that uses the predictions as features.



# Averaging and Stacking

- Averaging/stacking often **performs better than individual models**:
  - Averaging/stacking typically used by Kaggle winners.
  - E.g., Netflix \$1M user-rating competition winner was stacked classifier.
- Why does this work?
- Consider classifiers that tend to overfit (like deep decision trees):
  - If they all overfit in exactly the same way, averaging does nothing.
- But if they make **independent errors**:
  - Probability of **error of average can be lower** than individual classifiers.
  - Less attention to specific overfitting of each classifier.



# Random Forests

- Random forests **average 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:

1. Sample a random card:  
(put it back in the deck)



2. Sample a random card:  
(put it back in the deck)



3. Sample a random card:  
(put it back in the deck)

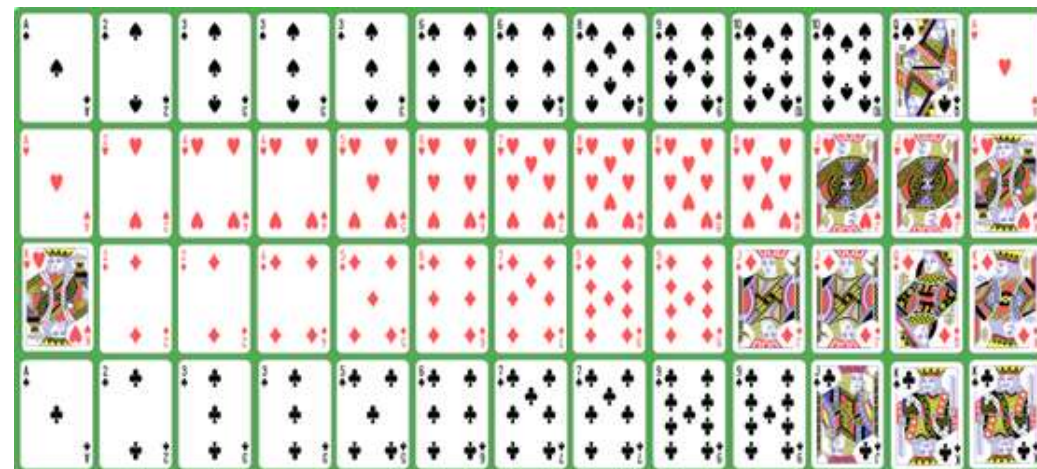
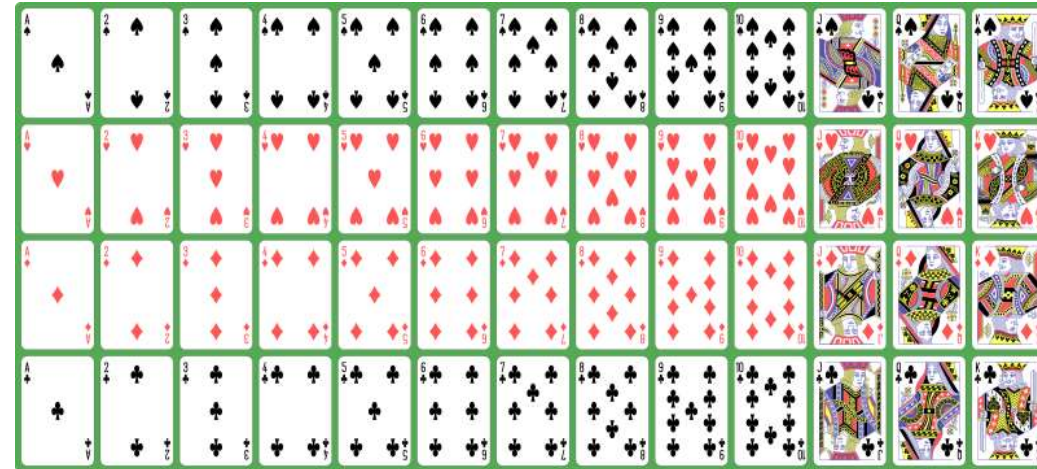


— ...

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



- We now have a new deck of 52 cards:





# Random Forest Ingredient 1: Bootstrap

- **Bootstrap sample** of a list of 'n' objects:

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

for  $i$  in  $1:n$   
 $j = \text{rand}(1:n)$  # pick a random number from  $\{1, 2, \dots, n\}$   
 $X_{\text{bootstrap}}[i, :] = X[j, :]$  # use the random sample

- Gives new dataset of 'n' objects, with some duplicated and some missing.

- Approximately 63% of original objects will be included for large 'n'.

- Very common in statistics to estimate sensitivity of statistic to data.

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

- Generate several **bootstrap samples of the objects**  $(x_i, y_i)$ .

- Fit a **classifier to each bootstrap** sample.

- At test time, **average the predictions**.

} Decision trees will make  
different splits.

# Random Forest Ingredient 2: Random Trees

- For each split in a random tree model:
  - Randomly sample a small number of possible features.
  - Only consider these random features when searching for the optimal rule.

Random tree 1:

- sample (milk, oranges)  $\text{milk} > 0.5$

Random tree 2:

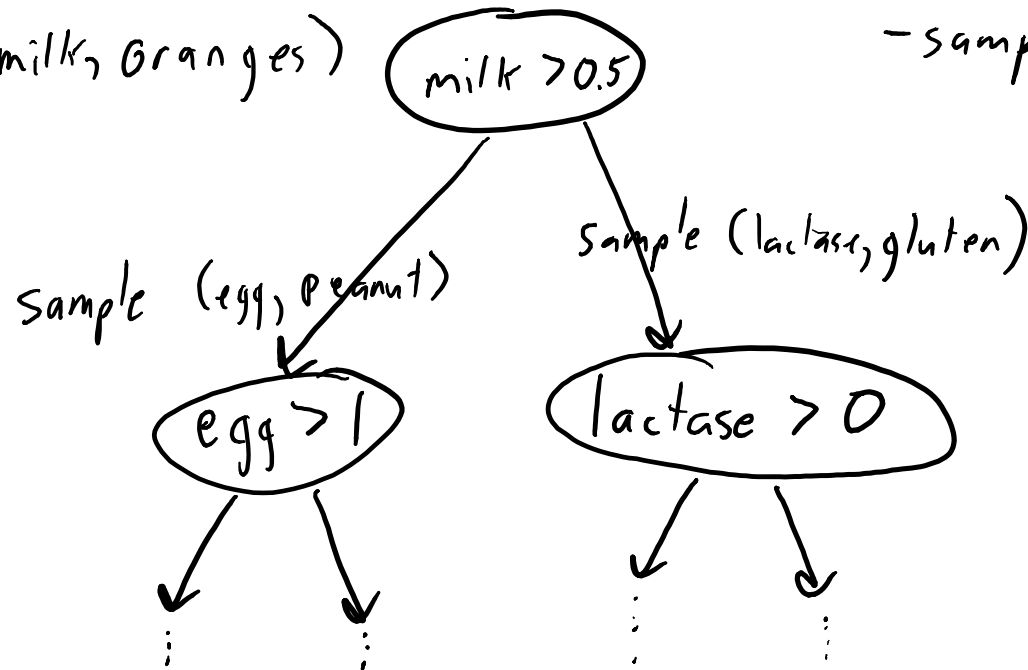
- sample (egg, lactase)  $\text{egg} > 0$

# Random Forest Ingredient 2: Random Trees

- For **each split** in a **random tree** model:
  - **Randomly sample** a small number of possible features.
  - **Only consider these random features** when searching for the optimal rule.

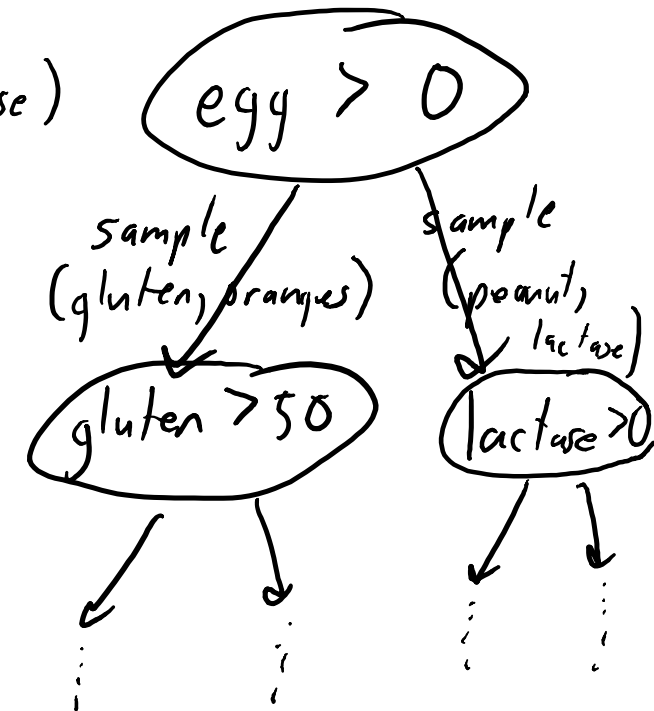
Random tree 1:

- sample (milk, oranges)



Random tree 2:

- sample (egg, lactase)



# Random Forest Ingredient 2: Random Trees

- For **each split** in a **random tree** model:
  - **Randomly sample** a small number of possible features.
  - **Only consider these random features** when searching for the optimal rule.
- Splits will tend to use **different features in different trees**.
  - They will still overfit, but hopefully make *\*independent\** errors.
- So the average tends to have a **much lower test error**.
- Empirically, random forests are one of the “best” classifiers.
- Fernandez-Delgado et al. [2014]:
  - Compared 179 classifiers on 121 datasets.
  - Random forests are most likely to be the best classifier.

# Summary

- Encouraging invariance:
  - Add transformed data to be insensitive to the transformation.
- Ensemble methods take classifiers as inputs.
  - Try to reduce either  $E_{\text{train}}$  or  $E_{\text{approx}}$  without increasing the other much.
- Averaging:
  - Improves predictions of multiple classifiers if errors are independent.
- Random forests:
  - Averaging of deep randomized decision trees.
  - One of the best “out of the box” classifiers.
- Next time:
  - We start unsupervised learning.



# Text Example 1: Language Identification

- Consider data that doesn't look like this:

$$X = \begin{bmatrix} 0.5377 & 0.3188 & 3.5784 \\ 1.8339 & -1.3077 & 2.7694 \\ -2.2588 & -0.4336 & -1.3499 \\ 0.8622 & 0.3426 & 3.0349 \end{bmatrix}, \quad y = \begin{bmatrix} +1 \\ -1 \\ -1 \\ +1 \end{bmatrix},$$

- But instead looks like this:

$$X = \begin{bmatrix} \text{Do you want to go for a drink sometime?} \\ \text{J'achète du pain tous les jours.} \\ \text{Fais ce que tu veux.} \\ \text{There are inner products between sentences?} \end{bmatrix}, \quad y = \begin{bmatrix} +1 \\ -1 \\ -1 \\ +1 \end{bmatrix}.$$

- How should we represent sentences using features?

# A (Bad) Universal Representation

- Treat character in position 'j' of the sentence as a categorical feature.
  - "fais ce que tu veux" =>  $x_i = [\text{f a i s " c e " q u e " t u " v e u x .}]$
- "Pad" end of the sentence up to maximum #characters:
  - "fais ce que tu veux" =>  $x_i = [\text{f a i s " c e " q u e " t u " v e u x . \gamma \gamma \gamma \gamma \gamma \gamma \gamma \gamma \dots}]$
- Advantage:
  - No information is lost, KNN can eventually solve the problem.
- Disadvantage: **throws out everything we know about language.**
  - Needs to learn that "veux" starting from any position indicates "French".
    - Doesn't even use that sentences are made of words (this must be learned).
  - High overfitting risk, you will need a lot of examples for this easy task.

# Bag of Words Representation

- Bag of words represents sentences/documents by **word counts**:

The **International Conference on Machine Learning (ICML)** is the leading international academic conference in machine learning



ICML	International	Conference	Machine	Learning	Leading	Academic
1	2	2	2	2	1	1

- Bag of words **loses a ton of information/meaning**:
  - But it **easily solves language identification** problem

# Universal Representation vs. Bag of Words

- Why is **bag of words** better than “**string of characters**” here?
  - It needs less data because it **captures invariances** for the task:
    - Most features give strong indication of one language or the other.
    - It doesn't matter *where* the French words appear.
  - It overfits less because it **throws away irrelevant information**.
    - Exact sequence of words isn't particularly relevant here.

# Text Example 2: Word Sense Disambiguation

- Consider the following two sentences:
  - “The cat ran after the **mouse**.”
  - “Move the **mouse** cursor to the File menu.”
- **Word sense disambiguation** (WSD): classify “meaning” of a word:
  - A surprisingly difficult task.
- You can do ok with bag of words, but it will have problems:
  - “Her **mouse** clicked on one **cat** video after another.”
  - “We saw the **mouse** run out from behind the **computer**.”
  - “The **mouse** was gray.” (ambiguous without more context)

# Bigrams and Trigrams

- A **bigram** is an ordered set of two words:
  - Like “computer mouse” or “mouse ran”.
- A **trigram** is an ordered set of three words:
  - Like “cat and mouse” or “clicked mouse on”.
- These give more context/meaning than bag of words:
  - Includes **neighbouring words** as well as **order of words**.
  - Trigrams are widely-used for various language tasks.
- General case is called **n-gram**.
  - Unfortunately, **coupon collecting** becomes a problem with larger ‘n’.

# Why does Bootstrapping select approximately 63%?

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

$$\begin{aligned} & p(\text{selected at least once in 'n' trials}) \\ &= 1 - p(\text{not selected in any of 'n' trials}) \\ &= 1 - (p(\text{not selected in one trial}))^n \\ &= 1 - (1 - 1/n)^n \\ &\approx 1 - 1/e \\ &\approx 0.63 \end{aligned}$$

(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 Slide: Why Random Forests Work

- Consider ‘k’ independent classifiers, whose errors have a variance of  $\sigma^2$ .
- If the errors are IID, the variance of the average is  $\sigma^2/k$ .
  - So the more classifiers you average, 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”](#).



# Boosting: Key Ideas

- Input to boosting is classifier that:
  - Is simple enough that it doesn't overfit much.
  - Can obtain  $>50\%$  weighted training accuracy.
- Example: decision stumps or low-depth decision trees.

# Boosting: Key Ideas

- Basic steps:
  1. Fit a classifier on the training data.
  2. Give a higher weight to examples that the classifier got wrong.
  3. Fit a classifier on the weighted training data.
  4. Go back to 2.
- Final prediction: weighted vote of individual classifier predictions.
- Boosted decision trees are very fast/accurate classifiers.
  - “AdaBoost”: classic boosting method.
  - “XGBoost”: recent method that has been winning Kaggle competitions.

# 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.

# Bonus Slide: Bayesian Model Averaging

- Recall the key observation regarding ensemble methods:
  - If **models overfit in “different” ways, averaging gives better performance.**
- But should all models get equal weight?
  - E.g., decision trees of different depths, when lower depths have low training error.
  - E.g., a random forest where one tree does very well (on validation error) and others do horribly.
  - In science, research may be fraudulent or not based on evidence.
- In these cases, naïve **averaging may do worse.**

# Bonus Slide: Bayesian Model Averaging

- Suppose we have a set of 'm' probabilistic binary classifiers  $w_j$ .
- If each one gets equal weight, then we predict using:

$$p(y_i | x_i) = \frac{1}{m} p(y_i | w_1, x_i) + \frac{1}{m} p(y_i | w_2, x_i) + \dots + \left(\frac{1}{m}\right) p(y_i | w_m, x_i)$$

- **Bayesian model averaging** treats model ' $w_j$ ' as a random variable:  $w_j \perp x_i$   
↑  
Assume

$$p(y_i | x_i) = \sum_{j=1}^m p(y_i, w_j | x_i) = \sum_{j=1}^m p(y_i | w_j, x_i) p(w_j | x_i) = \sum_{j=1}^m p(y_i | w_j, x_i) p(w_j)$$

- So we should weight by probability that  $w_j$  is the correct model:
  - Equal weights assume all models are equally probable.

# Bonus Slide: Bayesian Model Averaging

- Can get better weights by conditioning on training set:

$$p(w_j | X, y) \propto p(y | w_j, X) p(w_j | X) = p(y | w_j, X) p(w_j)$$

- The 'likelihood'  $p(y | w_j, X)$  makes sense:
  - We should give more weight to models that predict 'y' well.
  - Note that hidden denominator penalizes complex models.
- The 'prior'  $p(w_j)$  is our 'belief' that  $w_j$  is the correct model.
- This is how rules of probability say we should weigh models.
  - The 'correct' way to predict given what we know.
  - But it makes some people unhappy because it is subjective.

Again, assuming  $w_j | X$