CPSC 340: Machine Learning and Data Mining

Ensemble Methods

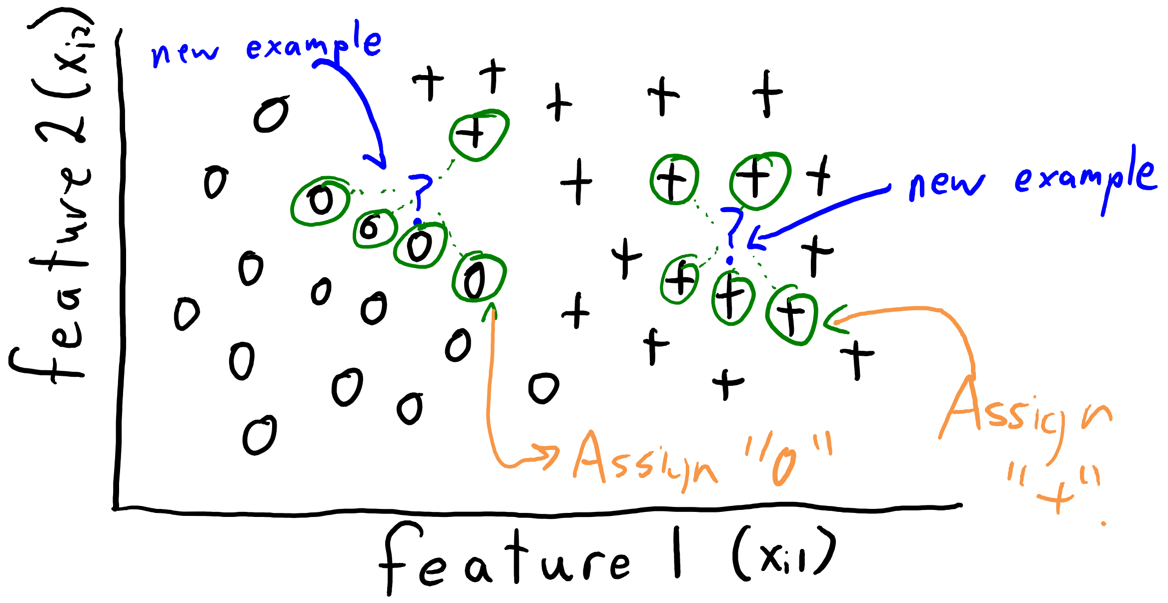
Fall 2016

Admin

- Welcome to the course!
  - You should know if you are officially in/out.
- Assignment 1 is due Friday.
  - Setup your CS undergrad account ASAP to use Handin:
    - https://www.cs.ubc.ca/getacct
  - Instructions for handin posted to Piazza.
  - 1 late day to hand it in before Monday's class.
  - 2 late days to hand it in before Wednesday's class.
  - 3 late days to hand it in before Friday of next week's class.
  - 0 after that.

K-Nearest Neighbours (KNN)

- K-nearest neighbours algorithm for classifying ‘x‘:
  - Find ‘k’ values of x_i that are most similar to x.
  - Use mode of corresponding y_i.



- Non-parametric:
  - Size of model grows with ‘n’.
- Consistency:
  - Nearly-optimal test error with infinite data.
- But how many examples are needed?

Curse of Dimensionality

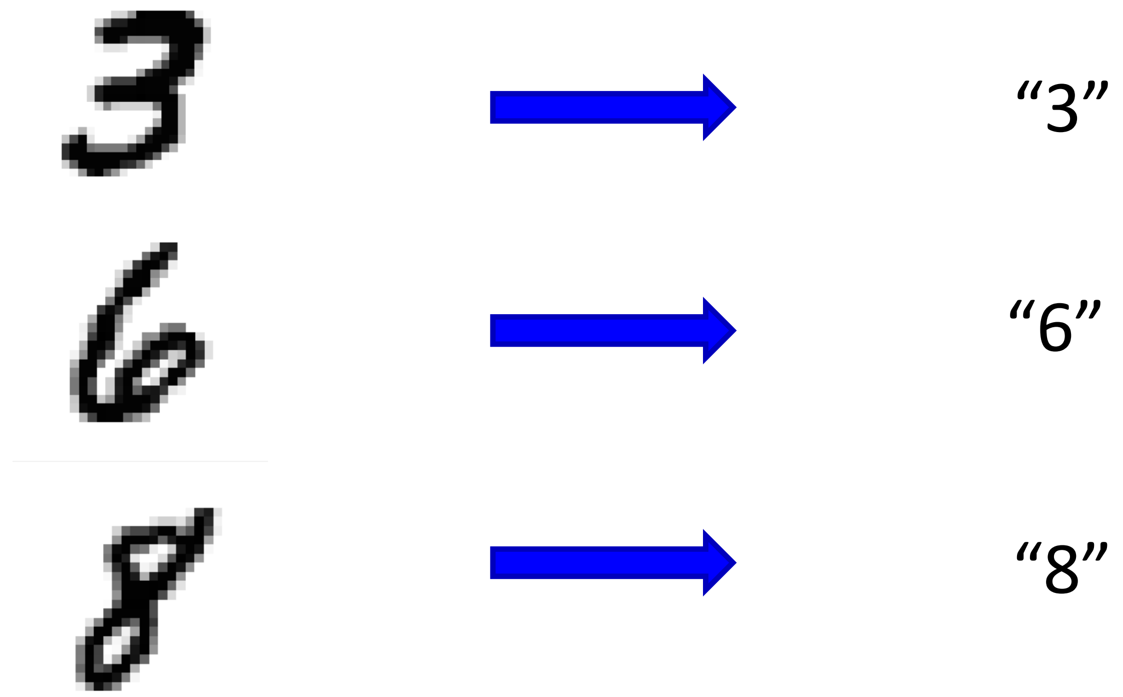
- "Curse of dimensionality": problems with high-dimensional spaces.
  - Volume of space grows exponentially with dimension.
  - Need exponentially more points to 'fill' a high-dimensional volume.
  - Distances become less meaningful:
    - All vectors may have similar distances.
  - Emergence of "hubs":
    - some datapoints are neighbours to many more points than average.

- KNN is also problematic if features have different scales.

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

Application: Optical Character Recognition

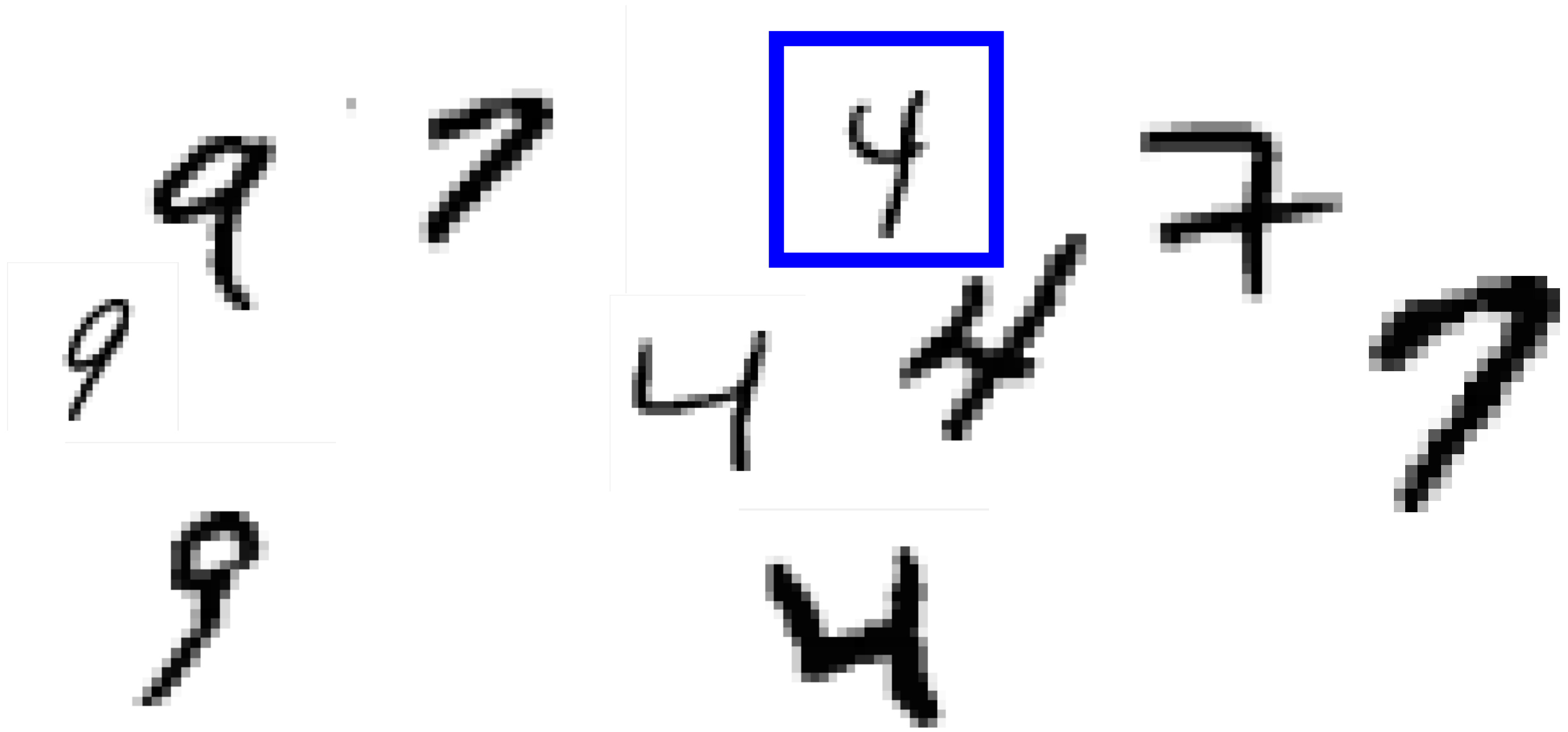
- We have collection of letter/digit images, and corresponding labels:



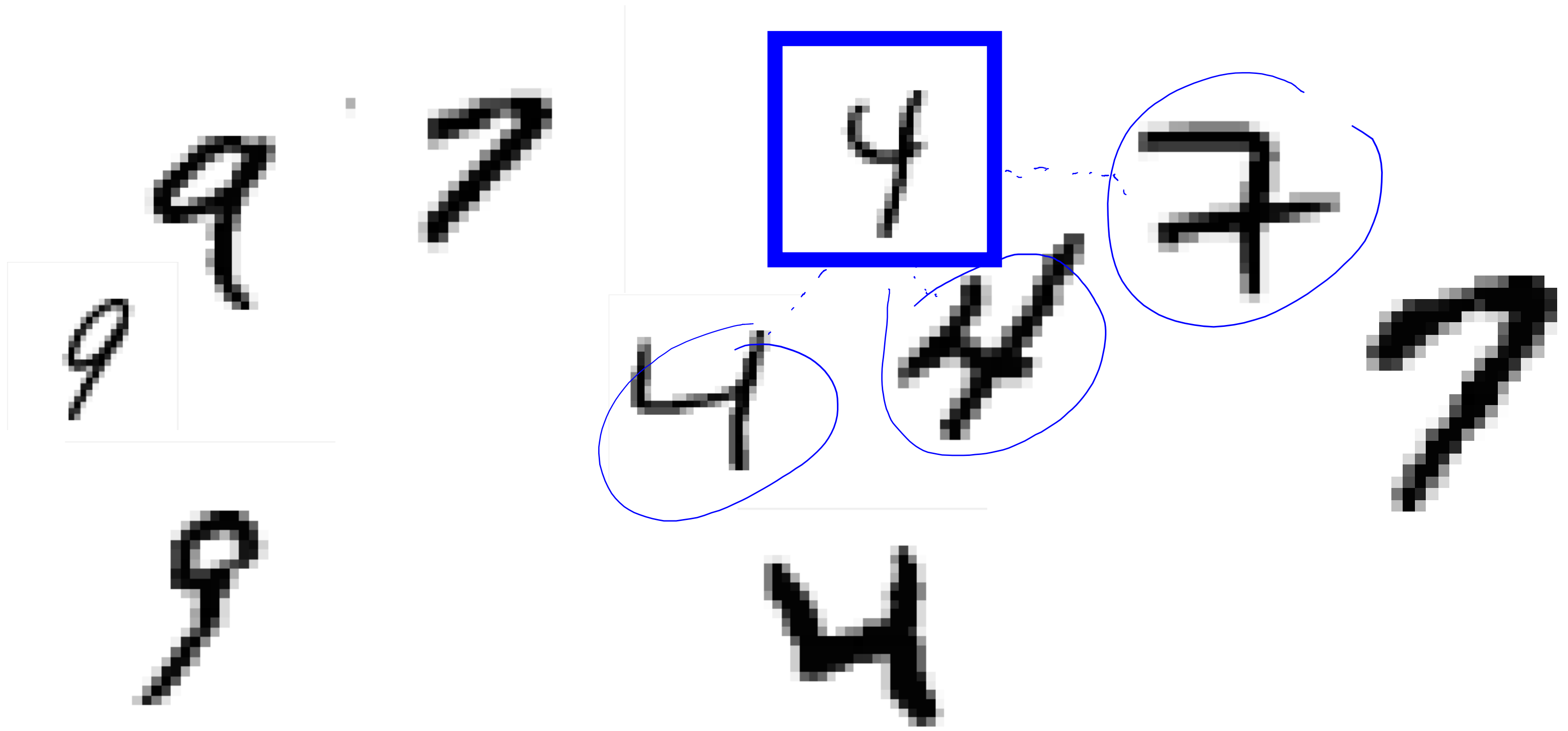
- Use supervised learning to automatically recognize letters/digits:
  - y_i could be the letter/digit, x_i could be the values of the pixels.

KNN for Optical Character Recognition

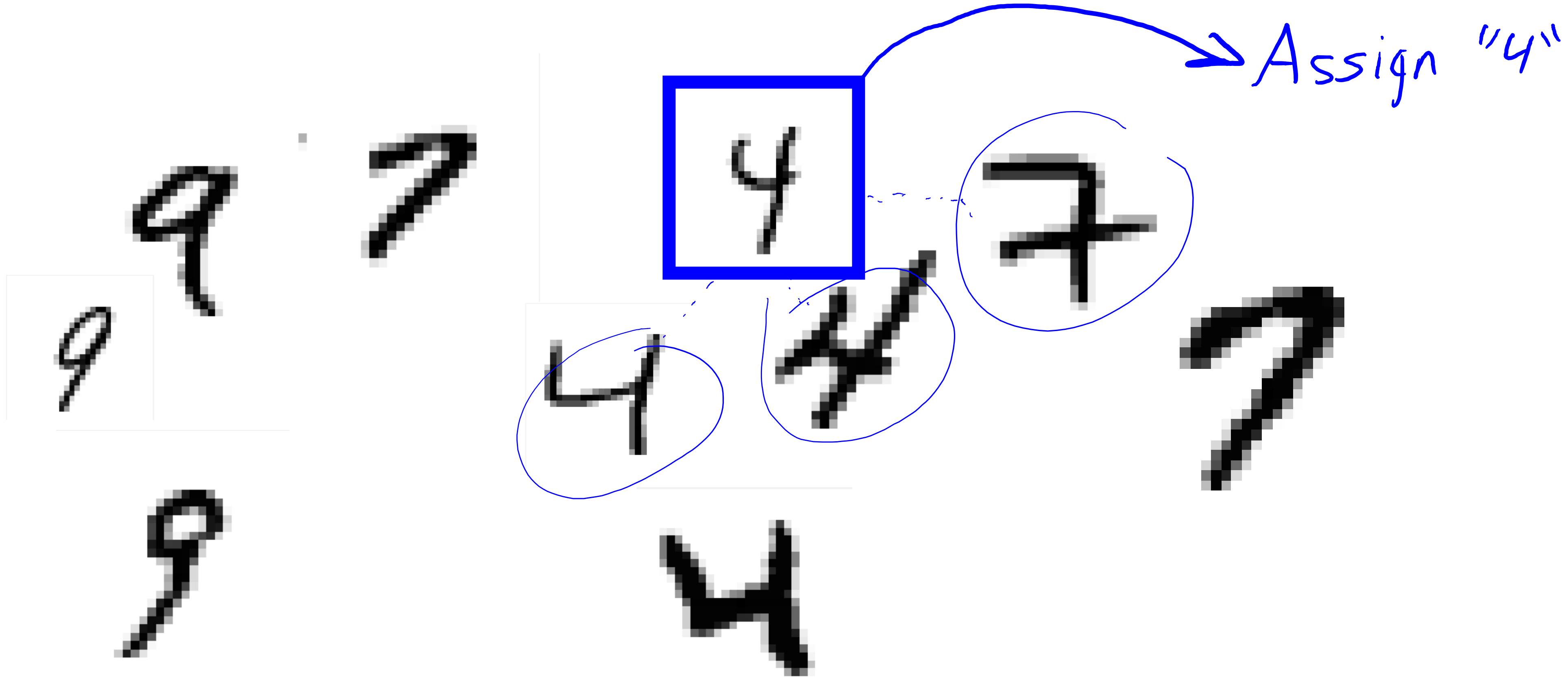
KNN for Optical Character Recognition



KNN for Optical Character Recognition



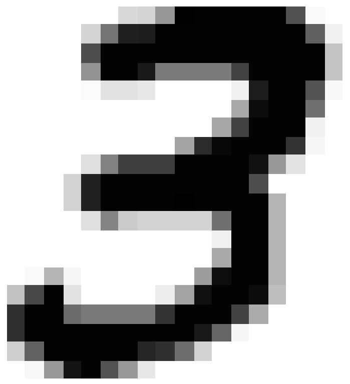
KNN for Optical Character Recognition



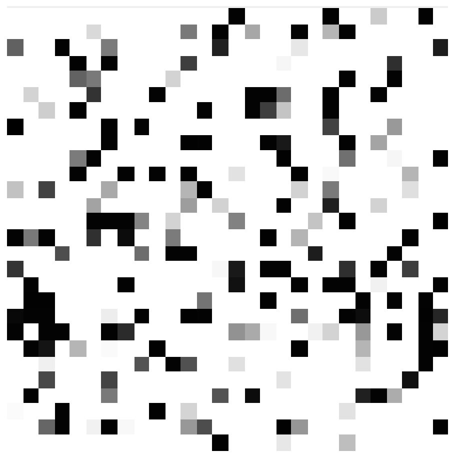
Human vs. Machine Perception

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

What we see:



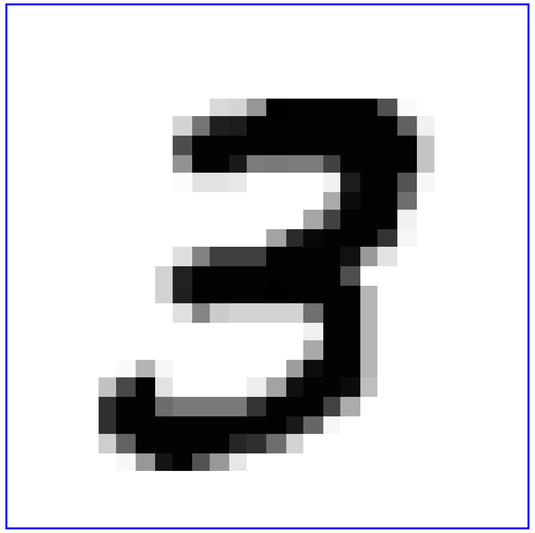
What the computer "sees":

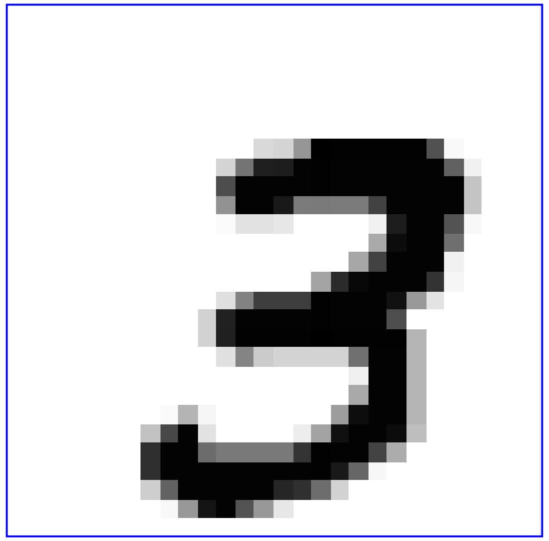


Actually, it's worse:

What the Computer Sees

- Are these two images ‘similar’?

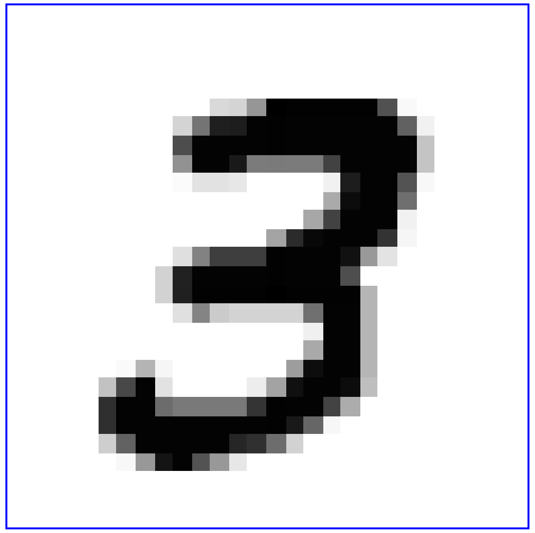
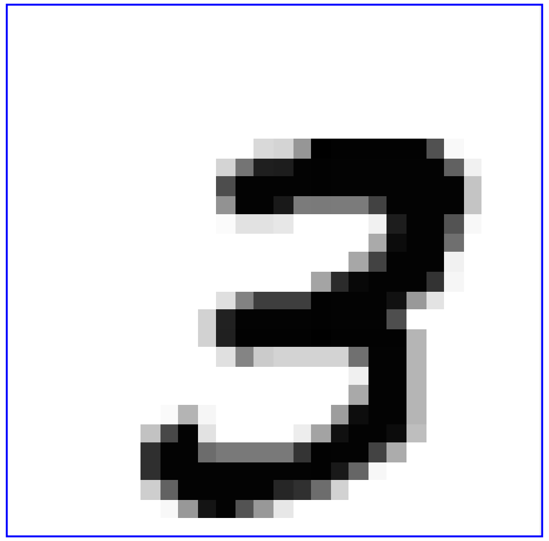
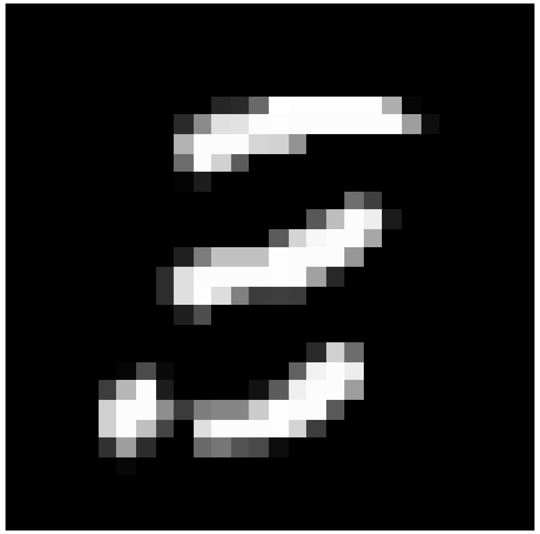




What the Computer Sees

- Are these two images 'similar'?

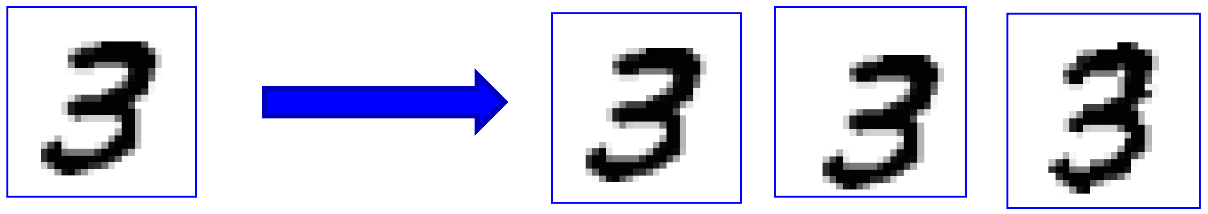
Difference:

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

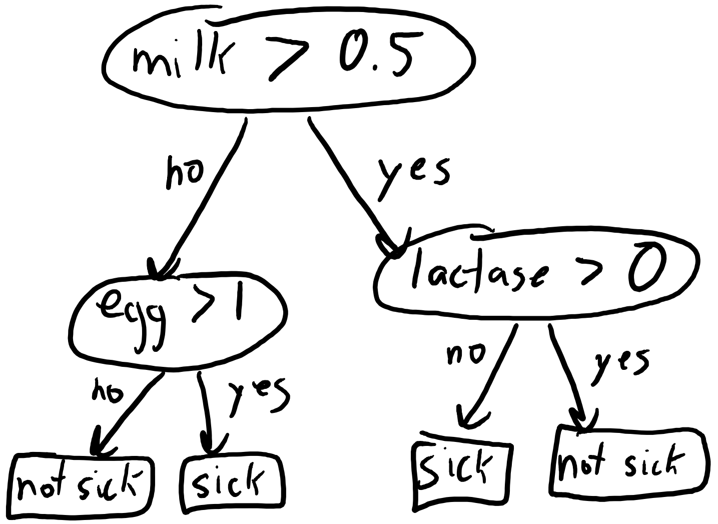
Encouraging Invariance

- May want classifier to be invariant to certain feature transforms.
  - Digits: 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.

Decision Trees vs. Naïve Bayes vs. KNN

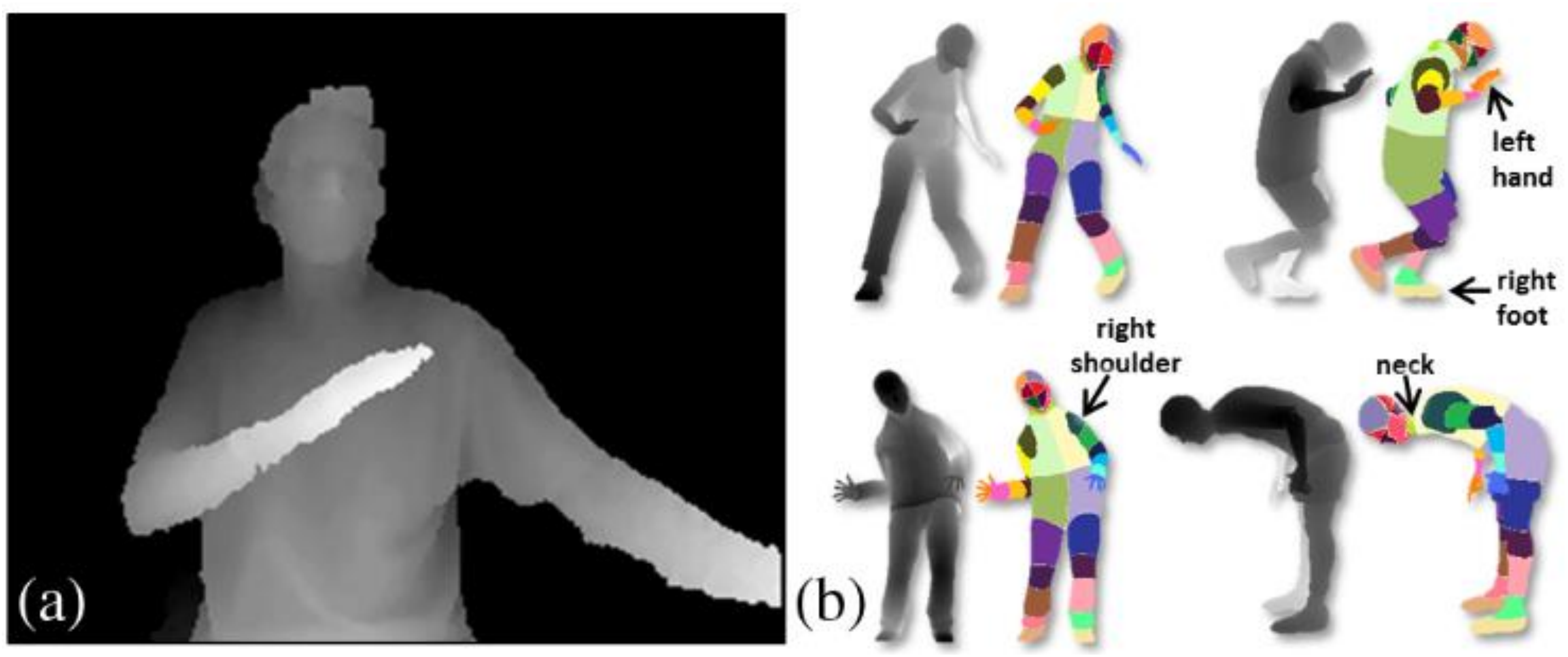


$$p(\text{sick} \mid \text{milk}, \text{egg}, \text{lactase})$$
$$\approx p(\text{milk} \mid \text{sick})\, p(\text{egg} \mid \text{sick})\, p(\text{lactase} \mid \text{sick})\, p(\text{sick})$$

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

Application: Body-Part Recognition

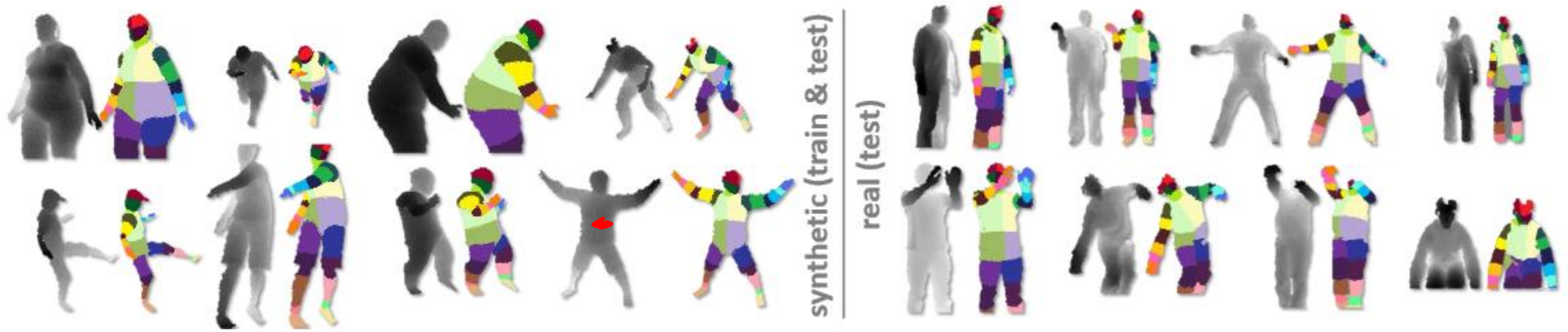
- Microsoft Kinect:
  - Real-time recognition of 31 body parts from laser depth data.



- 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.
  - Boosting.
  - Bootstrapping.
  - Bagging.
  - Cascading.
  - Random Forests.
  - Stacking.
- Meta-classifier often have higher accuracy than input classifiers.

Ensemble Methods

- Remember the fundamental trade-off:
  1. How small you can make the training error.

     vs.

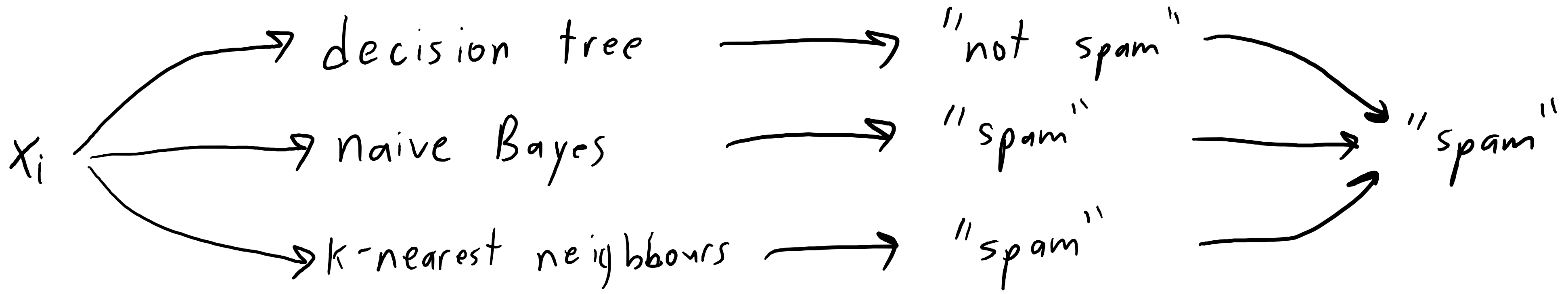
  2. How well training error approximates the test error.
- Goal of ensemble methods is that meta-classifier:
  - Does much better on one of these than individual classifiers.
  - Doesn't do too much worse on the other.
- This (roughly) gives two types:
  1. Boosting: take simple classifier that underfits, improve its training error.
  2. Averaging: take complex classifier that overfits, improve its test error.

Boosting

- 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.
- 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.

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).



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).
- Stacking:
  - Fit another classifier that uses the predictions as features.

$$\begin{tikzcd}
 & \text{decision tree} \arrow[r] & \text{"not spam"} \arrow[dr] & & \\
x_i \arrow[ur, bend left] \arrow[r] \arrow[dr, bend right] & \text{naive Bayes} \arrow[r] & \text{"spam"} \arrow[r] & \text{decision tree \#2} \arrow[r] & \text{"spam"} \\
 & \text{k-nearest neighbours} \arrow[r] & \text{"spam"} \arrow[ur] & &
\end{tikzcd}$$

Averaging

- Averaging often performs better than individual models:
  - Averaging typically used by Kaggle winners.
  - E.g., Netflix $1M user-rating competition winner was stacked classifier.
- Why does this work?
- Consider a set of classifiers that tend to overfit:
  - For example, 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 on 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?
  - If just fit a decision tree repeatedly to same data, all trees will be the same.

- Two key ingredients in random forests:
  - Bootstrap aggregation.
  - Random trees.

Random Forest Ingredient 1: Bagging

- Bootstrap sample of a list of 'n' objects:
  - A set of 'n' objects, chosen independently with replacement.

```
for i=1:n                          ⌈z⌉
    j = ceil(rand*n)   % Pick a random number from 1:n.
    X_bootstrap(i,:) = X(j,:)
```

  - Gives new dataset of 'n' objects, with some duplicated and some missing.
    - ~63% of original objects will be included.
  - Usually, it is used to estimate how sensitive a statistic is to the data.
- Bootstrap aggregation (bagging):
  - 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 <u>different</u> splits.

Random Forest Ingredient 2: Random Trees

- When fitting each decision stump to construct deep decision tree:
  - Do not consider all features when searching for the optimal rule.
  - Each split only considers a small number of randomly-chosen features.

Random tree 1:

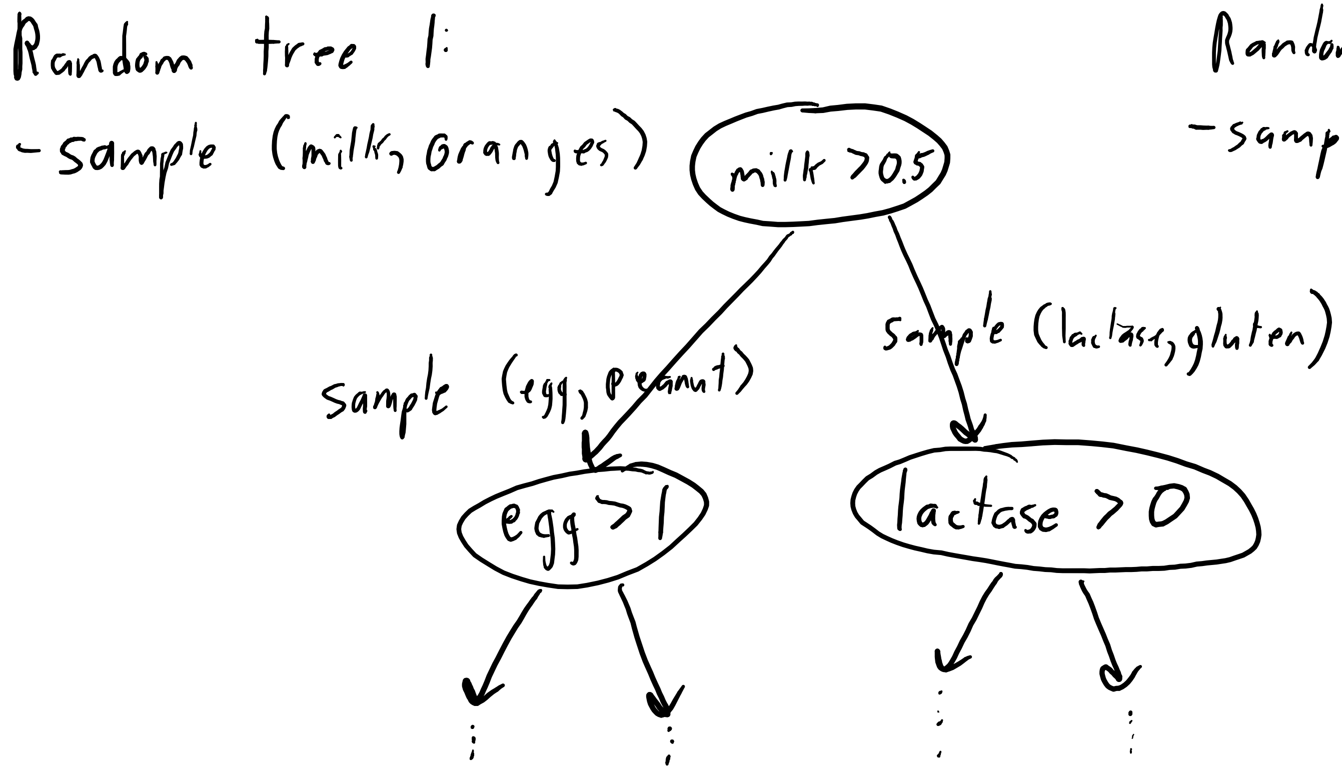
- sample (milk, oranges) $\boxed{\text{milk} > 0.5}$

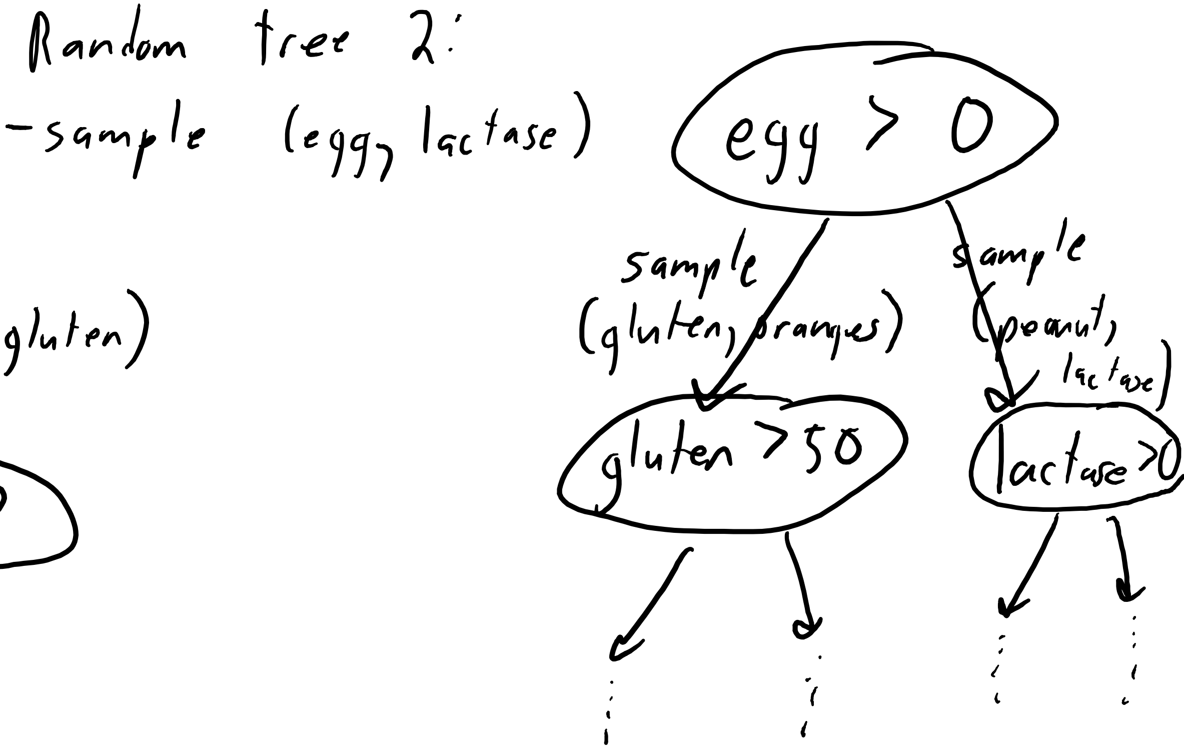
Random tree 2:

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

Random Forest Ingredient 2: Random Trees

- When fitting each decision stump to construct deep decision tree:
  - Do not consider all features when searching for the optimal rule.
  - Each split only considers a small number of randomly-chosen features.





Random Forest Ingredient 2: Random Trees

- When fitting each decision stump to construct deep decision tree:
  - Do not consider all features when searching for the optimal rule.
  - Each split only considers a small number of randomly-chosen features.
- These random trees will tend to be very different from each other.
  - They will still overfit, but in *different* ways.
- 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

1. Encouraging invariance:
   - Add transformed data to be insensitive to the transformation.
2. Ensemble methods take classifiers as inputs.
3. Boosting turns ‘weak’ classifiers into ‘strong’ classifiers.
4. Averaging predictions often improves performance.
5. Random forests:
   - Averaging of deep randomized decision trees.
   - One of the best “out of the box” classifiers.

- Next time:
  – We start unsupervised learning.

Bonus Slide: Why does Bootstrapping give 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

Bonus Slide: Why Random Forests Work

- Consider ‘k’ independent classifiers, whose errors have a variance of σ^2.
- If the errors are IID, the variance of the average is σ^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 decreasing correlation gets you closer to independent case.
- Randomization in random forests decreases correlation between trees.

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:

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

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

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

- 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

Again, assuming $w_j \perp X$

$\uparrow$

- Can get better weights by conditioning on training set:

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

- The 'likelihood' $p(y \mid w_j, X)$ 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 people uncomfortable because it is subjective.