Learning from Data

Russell and Norvig Chapter 18

Based on your internet history, you might be dumb enough to enjoy extreme sports.

Click here to buy a ticket to base jump from the international space station.

I think the internet is trying to kill me.

We call it "machine learning."
Learning

- Essential for agents working in unknown environments

- Learning is useful as a system construction method
  - Expose the agent to reality rather than trying to write it down

- Learning modifies the agent's decision mechanisms to improve performance
Learning from examples

- Supervised learning: Given labeled examples of each digit, learn a classification rule
Machine learning is ubiquitous

- Examples of systems that employ ML?
Examples of learning tasks

- OCR (Optical Character Recognition)
- Loan risk diagnosis
- Medical diagnosis
- Credit card fraud detection
- Speech recognition (e.g., in automatic call handling systems)
- Spam filtering
- Collaborative filtering (recommender systems)
- Biometric identification (fingerprints, iris scan, face)
- Information retrieval (incl. web searching)
- Data mining, e.g. customer purchase behavior
- Customer retention
- Bioinformatics: prediction of properties of genes and proteins.
Learning

- The agent tries to learn from the data (examples) provided to it.
- The agent receives feedback that tells it how well it is doing.
- There are several learning scenarios according to the type of feedback:
  - **Supervised learning**: correct answers for each example
  - **Unsupervised learning**: correct answers not given
  - **Reinforcement learning**: occasional rewards (e.g. learning to play a game).
- Each scenario has appropriate learning algorithms
ML tasks

Classification: discrete/categorical labels

Regression: continuous labels

Clustering: no labels
Inductive learning

- Construct a function that agrees with the training data
- Example: curve fitting
Inductive learning

- Construct a function that agrees with the training data
- Example: curve fitting
Inductive learning

- Construct a function that agrees with the training data
- Example: curve fitting
Inductive learning

- Construct a function that agrees with the training data
- Example: curve fitting

Which of these models do you think is better?
Inductive learning

- Construct a function that agrees with the training data
- Example: curve fitting

Learning is concerned with accurate prediction of future data, *not* accurate prediction of training data.

Occam’s razor: prefer the simplest hypothesis consistent with data.
Occam’s Razor

Savage Chickens

It’s So Simple!

NEW?

Just One Blade!

OCCAM’S RAZOR!

Occam’s Razor

Pluralitas non est ponenda sine necessitate.
(Plurality should not be posited without necessity.)
- William of Ockham

Everything should be made as simple as possible, but not simpler.
- Albert Einstein

Keep It Simple, Stupid!

http://old.aitopics.org/AlToons
Overfitting in classification
Supervised Learning

Example: want to classify versus

Data: Labeled images

\[ \mathcal{D} = \{(x_i, y_i)\}_{i=1}^n \]

\[ y_i \in \{\text{monkey, human}\} \]

\[ x_i \text{ is a vector of features that represents the image} \]

Task: Here is a new image:
What species is it?
**The Nearest Neighbor Method**

(your first classification algorithm!)

**NN(image):**

1. Find the image in the training data which is closest to the query image.
2. Return its label.

---

*query*  
[Image of a person]  
[Image of a monkey]

*closest image*
Distance measures

- How to measure closeness?
Distance measures

- How to measure closeness?

- Discrete data: Hamming distance
- Continuous data: Euclidean distance
- Sequence data: edit distance

- Alternative: use a similarity measure (or dot product) rather than a distance
**k-NN**

- Use the closest k neighbors to make a decision instead of a single nearest neighbor.
- Why do you expect this to work better?
Remarks on Nearest Neighbor

- Very easy to implement
- No training required. All the computation performed in classifying an example (complexity: $O(n)$)
- Need to store the whole training set (memory inefficient).
- Flexible, no prior assumptions (a type of non parametric classifier: does not assume anything about the data).
- Curse of dimensionality: if data has many features that are irrelevant/noisy distances are always large.
Take home question

- How would you convert the k-nearest-neighbor classification method to a regression method?
Measuring classifier performance

- Or how accurate is my classifier.
Measuring classifier performance

- The error-rate on a set of examples
  \[ \mathcal{D} = \{(x_i, y_i)\}_{i=1}^{n} : \]

  \[
  \frac{1}{n} \sum_{i=1}^{n} I(f(x_i) \neq y_i)
  \]

  *I* is the indicator function that returns 1 if its argument is True and zero otherwise.

- What is the error rate of a nearest neighbor classifier applied to its training set?
- Accuracy is 1-(error-rate)
Measuring classifier performance

- The error-rate on a set of examples
  \[ \mathcal{D} = \{(x_i, y_i)\}_{i=1}^{n} : \]

  \[
  \frac{1}{n} \sum_{i=1}^{n} I(f(x_i) \neq y_i)
  \]

  *I* is the indicator function that returns 1 if its argument is True and zero otherwise.

- Report error rates computed on an independent **test set** (classifier was trained using **training set**): classifier performance on the training set is not indicative of performance on unseen data.
Measuring classifier performance

- The error-rate on a set of examples
  \[ \mathcal{D} = \{(x_i, y_i)\}_{i=1}^n : \]

  \[ \frac{1}{n} \sum_{i=1}^{n} I(f(x_i) \neq y_i) \]

  \( I \) is the indicator function that returns 1 if its argument is True and zero otherwise.

- Issue when classes are imbalanced.
- There are other measures of performance that address this.
Measuring classifier performance

- Split data into training set and test set (say 70%, 30%).
- Compare several classifiers trained on this split.
- Train final best classifier on the full dataset.

- A better method: **cross-validation**
Cross-validation

- Split data into k parts \((E_1, \ldots, E_k)\)
  - for \(i = 1, \ldots, k:\)
    - training set = \(D \setminus E_i\)
    - test set = \(E_i\)
    - classifier.train(training set)
    - accumulate results of classifier.test(test set)

- This is called k-fold cross-validation
- Extreme version: Leave-One-Out
- Assumptions?
Uses of CV

Cross Validation is used to choose:

- Classifier parameters
  - $k$ for $k$-NN
- Normalization method
- Which classifier
- Feature selection (which features provide best performance).
- This is called model selection
CV-based model selection

- We’re trying to determine which classifier to use

<table>
<thead>
<tr>
<th>Classifier</th>
<th>Training error</th>
<th>CV-error</th>
<th>choice</th>
</tr>
</thead>
<tbody>
<tr>
<td>f₁</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>f₂</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>f₃</td>
<td></td>
<td></td>
<td>✔</td>
</tr>
<tr>
<td>f₄</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>f₅</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>f₆</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
CV-based model selection

- Example: choosing $k$ for the $k$-NN algorithm:

<table>
<thead>
<tr>
<th>Classifier</th>
<th>Training error</th>
<th>CV-error</th>
<th>choice</th>
</tr>
</thead>
<tbody>
<tr>
<td>$K = 1$</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>$K = 2$</td>
<td><img src="image" alt="Training Error" /></td>
<td><img src="image" alt="CV-error" /></td>
<td></td>
</tr>
<tr>
<td>$K = 3$</td>
<td><img src="image" alt="Training Error" /></td>
<td><img src="image" alt="CV-error" /></td>
<td>✔</td>
</tr>
<tr>
<td>$K = 4$</td>
<td><img src="image" alt="Training Error" /></td>
<td><img src="image" alt="CV-error" /></td>
<td></td>
</tr>
<tr>
<td>$K = 5$</td>
<td><img src="image" alt="Training Error" /></td>
<td><img src="image" alt="CV-error" /></td>
<td></td>
</tr>
<tr>
<td>$K = 6$</td>
<td><img src="image" alt="Training Error" /></td>
<td><img src="image" alt="CV-error" /></td>
<td></td>
</tr>
</tbody>
</table>
The general workflow

- Formulate problem
- Get data
- Decide on a representation (what features to use)
- Visualize the data
- Choose a classifier
- Assess the performance of the classifier
- Depending on the results: modify the representation, classifier, or look for more data
Decision Tree Learning

Russell and Norvig Chapter 18.3
Learning decision trees

Problem: decide whether to wait for a table at a restaurant, based on the following attributes:

1. Alternate: is there an alternative restaurant nearby?
2. Bar: is there a comfortable bar area to wait in?
3. Fri/Sat: is today Friday or Saturday?
4. Hungry: are we hungry?
5. Patrons: number of people in the restaurant (None, Some, Full)
6. Price: price range ($, $$, $$$)
7. Raining: is it raining outside?
8. Reservation: have we made a reservation?
9. Type: kind of restaurant (French, Italian, Thai, Burger)
10. WaitEstimate: estimated waiting time (0-10, 10-30, 30-60, >60)
Attribute-based representations

- Examples described by **attribute values** (Boolean, discrete, continuous)
- Example: situations where I will/won't wait for a table:

<table>
<thead>
<tr>
<th>Example</th>
<th>Alt</th>
<th>Bar</th>
<th>Fri</th>
<th>Hun</th>
<th>Pat</th>
<th>Price</th>
<th>Rain</th>
<th>Res</th>
<th>Type</th>
<th>Est</th>
<th>Wait</th>
</tr>
</thead>
<tbody>
<tr>
<td>X_1</td>
<td>T</td>
<td>F</td>
<td>F</td>
<td>T</td>
<td>Some</td>
<td>$$$$</td>
<td>T</td>
<td>T</td>
<td>French</td>
<td>0–10</td>
<td>T</td>
</tr>
<tr>
<td>X_2</td>
<td>T</td>
<td>F</td>
<td>F</td>
<td>T</td>
<td>Full</td>
<td>$</td>
<td>F</td>
<td>F</td>
<td>Thai</td>
<td>30–60</td>
<td>F</td>
</tr>
<tr>
<td>X_3</td>
<td>F</td>
<td>T</td>
<td>F</td>
<td>F</td>
<td>Some</td>
<td>$</td>
<td>F</td>
<td>F</td>
<td>Burger</td>
<td>0–10</td>
<td>T</td>
</tr>
<tr>
<td>X_4</td>
<td>T</td>
<td>F</td>
<td>T</td>
<td>T</td>
<td>Full</td>
<td>$</td>
<td>F</td>
<td>F</td>
<td>Thai</td>
<td>10–30</td>
<td>T</td>
</tr>
<tr>
<td>X_5</td>
<td>T</td>
<td>F</td>
<td>T</td>
<td>F</td>
<td>Full</td>
<td>$$$$</td>
<td>F</td>
<td>T</td>
<td>French</td>
<td>&gt;60</td>
<td>F</td>
</tr>
<tr>
<td>X_6</td>
<td>F</td>
<td>T</td>
<td>F</td>
<td>T</td>
<td>Some</td>
<td>$$</td>
<td>T</td>
<td>T</td>
<td>Italian</td>
<td>0–10</td>
<td>T</td>
</tr>
<tr>
<td>X_7</td>
<td>F</td>
<td>T</td>
<td>F</td>
<td>F</td>
<td>None</td>
<td>$</td>
<td>T</td>
<td>F</td>
<td>Burger</td>
<td>0–10</td>
<td>F</td>
</tr>
<tr>
<td>X_8</td>
<td>F</td>
<td>F</td>
<td>F</td>
<td>T</td>
<td>Some</td>
<td>$$</td>
<td>T</td>
<td>T</td>
<td>Thai</td>
<td>0–10</td>
<td>T</td>
</tr>
<tr>
<td>X_9</td>
<td>F</td>
<td>T</td>
<td>T</td>
<td>F</td>
<td>Full</td>
<td>$</td>
<td>T</td>
<td>F</td>
<td>Burger</td>
<td>&gt;60</td>
<td>F</td>
</tr>
<tr>
<td>X_10</td>
<td>T</td>
<td>T</td>
<td>T</td>
<td>T</td>
<td>Full</td>
<td>$$$$</td>
<td>F</td>
<td>T</td>
<td>Italian</td>
<td>10–30</td>
<td>F</td>
</tr>
<tr>
<td>X_11</td>
<td>F</td>
<td>F</td>
<td>F</td>
<td>F</td>
<td>None</td>
<td>$</td>
<td>F</td>
<td>F</td>
<td>Thai</td>
<td>0–10</td>
<td>F</td>
</tr>
<tr>
<td>X_12</td>
<td>T</td>
<td>T</td>
<td>T</td>
<td>T</td>
<td>Full</td>
<td>$</td>
<td>F</td>
<td>F</td>
<td>Burger</td>
<td>30–60</td>
<td>T</td>
</tr>
</tbody>
</table>

- Examples are **labeled** as positive (T) or negative (F)
Decision trees

- Decision trees: a form of representation for hypotheses (classification rules)
- Example: the “true” tree for deciding whether to wait:
Expressiveness

- Decision trees can express any function of the input attributes.
- E.g., for Boolean functions, truth table row $\rightarrow$ path to leaf:

<table>
<thead>
<tr>
<th>A</th>
<th>B</th>
<th>A xor B</th>
</tr>
</thead>
<tbody>
<tr>
<td>F</td>
<td>F</td>
<td>F</td>
</tr>
<tr>
<td>F</td>
<td>T</td>
<td>T</td>
</tr>
<tr>
<td>T</td>
<td>F</td>
<td>T</td>
</tr>
<tr>
<td>T</td>
<td>T</td>
<td>F</td>
</tr>
</tbody>
</table>

- Trivially, there is a consistent decision tree for any training set with one path to leaf for each example but it probably won't generalize to new examples
- Prefer to find compact decision trees
Decision tree learning

- Aim: find a small tree consistent with the training examples
- Idea: (recursively) choose "most significant" attribute as root of (sub)tree

```
function DTL(examples, attributes, default) returns a decision tree
    if examples is empty then return default
    else if all examples have the same classification then return the classification
    else if attributes is empty then return MODE(examples)
    else
        best ← CHOOSE-ATTRIBUTE(attributes, examples)
        tree ← a new decision tree with root test best
        for each value $v_i$ of best do
            $examples_i ← \{\text{elements of examples with best } = v_i\}$
            $subtree ← DTL(examples_i, attributes − best, MODE(examples))$
            add a branch to tree with label $v_i$ and subtree $subtree$
    return tree
```
Choosing an attribute

- Idea: a good attribute splits the examples into subsets that are (ideally) "all positive" or "all negative"

- **Patrons?** is a better choice
- Need a measure of quality for an attribute
I am thinking of an integer between 0 and 1,023. You want to guess it using the fewest number of questions.

Most of us would ask “is it between 0 and 512?”

This is a good strategy because it provides the most information about the unknown number.

It provides the first binary digit of the number.

Initially you need to obtain $\log_2(1024) = 10$ bits of information. After the first question you only need $\log_2(512) = 9$ bits.
Information theory (cont).

- Consider a question that splits the set of numbers 0,…, 1023 into two sets.
- The average information left after a split of $N$ numbers into two sets of size $n$ and $p$:
  \[
  \frac{n}{N} \log n + \frac{p}{N} \log p
  \]
- The average information provided by a question:
  \[
  \log N - \left( \frac{n}{N} \log n + \frac{p}{N} \log p \right) =
  - \frac{n}{N} \log \frac{n}{N} - \frac{p}{N} \log \frac{p}{N} = \text{Entropy}(n/N, p/N)
  \]
- The definition for a multinomial distribution:
  \[
  \text{Entropy}(p_1,\ldots,p_k) = I(p_1,\ldots,p_k) = - \sum p_i \log_2(p_i)
  \]
Entropy

- Graph of the entropy function for a binomial distribution:

- Maximal when $p = 1/2$. 
Example: triangles and squares

<table>
<thead>
<tr>
<th>#</th>
<th>Attribute</th>
<th>Shape</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Color</td>
<td>Outline</td>
</tr>
<tr>
<td>1</td>
<td>green</td>
<td>dashed</td>
</tr>
<tr>
<td>2</td>
<td>green</td>
<td>dashed</td>
</tr>
<tr>
<td>3</td>
<td>yellow</td>
<td>dashed</td>
</tr>
<tr>
<td>4</td>
<td>red</td>
<td>dashed</td>
</tr>
<tr>
<td>5</td>
<td>red</td>
<td>solid</td>
</tr>
<tr>
<td>6</td>
<td>red</td>
<td>solid</td>
</tr>
<tr>
<td>7</td>
<td>green</td>
<td>solid</td>
</tr>
<tr>
<td>8</td>
<td>green</td>
<td>dashed</td>
</tr>
<tr>
<td>9</td>
<td>yellow</td>
<td>solid</td>
</tr>
<tr>
<td>10</td>
<td>red</td>
<td>solid</td>
</tr>
<tr>
<td>11</td>
<td>green</td>
<td>solid</td>
</tr>
<tr>
<td>12</td>
<td>yellow</td>
<td>dashed</td>
</tr>
<tr>
<td>13</td>
<td>yellow</td>
<td>solid</td>
</tr>
<tr>
<td>14</td>
<td>red</td>
<td>dashed</td>
</tr>
</tbody>
</table>
Entropy

- 5 triangles
- 9 squares
- class probabilities

\[ p(\square) = \frac{9}{14} \]
\[ p(\triangle) = \frac{5}{14} \]

- entropy

\[ I = - \frac{9}{14} \log_2 \frac{9}{14} - \frac{5}{14} \log_2 \frac{5}{14} = 0.940 \text{ bits} \]
Entropy after partitioning

Reduction in entropy by partitioning

\[ I(red) = -\frac{3}{5} \log_2 \frac{3}{5} - \frac{2}{5} \log_2 \frac{2}{5} = 0.971 \text{ bits} \]

\[ I(green) = -\frac{2}{5} \log_2 \frac{2}{5} - \frac{3}{5} \log_2 \frac{3}{5} = 0.971 \text{ bits} \]

\[ I(yellow) = -\frac{4}{4} \log_2 \frac{4}{4} - \frac{0}{4} \log_2 \frac{0}{4} = 0.0 \text{ bits} \]
Entropy after partitioning

\[ I(Color) = \sum_v p(v)I(v) = \frac{5}{14} \times 0.971 + \frac{5}{14} \times 0.971 + \frac{4}{14} \times 0 = 0.694 \text{ bits} \]
Information Gain

\[ I(Green) = 0.971 \text{ bits} \]

\[ I(Yellow) = 0.0 \text{ bits} \]

\[ I(red) = 0.971 \text{ bits} \]

\[ p(red) = \frac{5}{14} \]

\[ p(yellow) = \frac{4}{14} \]

\[ p(green) = \frac{5}{14} \]

\[ IG(Color) = I - I(Color) = I - \sum_v p(v)I(v) = 0.940 - 0.694 = 0.246 \text{ bits} \]
Information Gain

- Attributes
  - IG(Color) = 0.246
  - IG(Outline) = 0.151
  - IG(Dot) = 0.048

- Heuristic: attribute with the highest gain is chosen for making a split
Example

Gain(Outline) = 0.971 - 0 = 0.971 bits
Gain(Dot) = 0.971 - 0.951 = 0.020 bits
Example (cont.)

Gain(Outline) = 0.971 - 0.951 = 0.020 bits
Gain(Dot) = 0.971 - 0 = 0.971 bits
Example (cont.)

Color:
- red
- yellow
- green

Dot?
- yes
- no

Outline?
- dashed
- solid
Issue with IG

- IG favors attributes with many values
- Such attribute splits S to many subsets, and if these are small, they will tend to be pure anyway
- One way to rectify this is through a corrected measure of information gain ratio:
  \[
  \text{GainRatio}(A) = \frac{\text{IG}(A)}{\text{IntrinsicInfo}(A)}
  \]

\text{IntrinsicInfo}(A) is the entropy associated with the distribution of the attribute: \( I(p_1, \ldots, p_k) \) where \( p_i \) is the probability of observing the \( i \)th value of A.
### Information Gain and Information Gain Ratio

|     | \( |v(A)| \) | Gain(A) | GainRatio(A) |
|-----|----------|--------|------------|
| Color | 3        | 0.247  | 0.156      |
| Outline | 2       | 0.152  | 0.152      |
| Dot  | 2        | 0.048  | 0.049      |
The Gini Index

- Another sensible measure of impurity (i and j are classes)

\[
Gini = \sum_{i \neq j} p(i)p(j)
\]

- After applying a split on attribute A, the resulting Gini index is

\[
Gini(A) = \sum_v p(v) \sum_{i \neq j} p(i|v)p(j|v)
\]

- Gini can be interpreted as expected error rate
The Gini Index: example

\[ p(\Box) = \frac{9}{14} \]

\[ p(\Delta) = \frac{5}{14} \]

\[ Gini = \sum_{i \neq j} p(i)p(j) \]

\[ Gini = \frac{9}{14} \times \frac{5}{14} = 0.230 \]
The Gini Index: example (cont.)

\[ p(\text{red}) = \frac{5}{14} \]

\[ p(\text{green}) = \frac{5}{14} \]

\[ p(\text{yellow}) = \frac{4}{14} \]

\[ Gini(A) = \sum_v \ p(v) \ \sum_{i \neq j} \ p(i|v) p(j|v) \]

\[ Gini(\text{Color}) = \frac{5}{14} \times \left( \frac{3}{5} \times \frac{2}{5} \right) + \frac{5}{14} \times \left( \frac{2}{5} \times \frac{3}{5} \right) + \frac{4}{14} \times \left( \frac{4}{4} \times \frac{0}{4} \right) = 0.171 \]
The Gini Gain Index

\[ Gini = \frac{9}{14} \times \frac{5}{14} = 0.230 \]

\[ Gini(\text{Color}) = \frac{5}{14} \times \left( \frac{3}{5} \times \frac{2}{5} \right) + \frac{5}{14} \times \left( \frac{2}{5} \times \frac{3}{5} \right) + \frac{4}{14} \times \left( \frac{4}{4} \times \frac{0}{4} \right) = 0.171 \]

\[ GiniGain(\text{Color}) = 0.230 - 0.171 = 0.058 \]
Three Impurity Measures

<table>
<thead>
<tr>
<th>A</th>
<th>Gain(A)</th>
<th>GainRatio(A)</th>
<th>GiniGain(A)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Color</td>
<td>0.247</td>
<td>0.156</td>
<td>0.058</td>
</tr>
<tr>
<td>Outline</td>
<td>0.152</td>
<td>0.152</td>
<td>0.046</td>
</tr>
<tr>
<td>Dot</td>
<td>0.048</td>
<td>0.049</td>
<td>0.015</td>
</tr>
</tbody>
</table>

These impurity measures assess the effect of a single attribute. The criterion “most informative” that they define is local (and “myopic”). It does not reliably predict the effect of several attributes applied jointly.
Back to the restaurant example

- Decision tree learned from the 12 examples:

  Substantially simpler than “true” tree - a more complex hypothesis isn’t justified by small amount of data.
Continuous variables

- If variables are continuous we can bin them.
- Alternative: learn a simple classifier on a single dimension, e.g. find decision point, classifying all data to the left in one class and all data to the right in the other.

So, we choose an attribute, a sign “+/-” and a threshold. This determines a half space to be +1 and the other half -1.
When to Stop?

- If we keep splitting until perfect classification we might over-fit.

- Heuristics:
  - Stop when splitting criterion is below some threshold
  - Stop when number of examples in each leaf node is below some threshold

- Alternative: **prune** the tree; potentially better than stopped splitting, since split may be useful at a later point.
Comments on decision trees

- Fast training (complexity?)
- Established commercial software (e.g. CART, C4.5, C5.0)
- Users like them because they produce rules which can supposedly be interpreted (but decision trees are very sensitive with respect to training data)
- Able to handle missing data (how?)
When are decision trees useful?

- Limited accuracy for problems with many variables. Why?
Classification by committee

- An example of a *committee classifier*: a classifier that bases its prediction on a set of classifiers.
- Output: a majority vote
- If the errors made by the individual classifiers are independent the committee will perform well.
Random Forests

- A committee algorithm that combines decision trees (Breiman, 2001)
- To train a tree in a random forest:
  - Choose a training set by picking $n$ examples with replacement from the $n$ training examples (a bootstrap sample).
  - For each node of the tree, randomly choose $m$ variables on which to base variable choice at that node.
Advantages of random forests

- State of the art performance
- Works well for high dimensional data.
- Estimates the importance of variables in determining classification.
- It generates an estimate of the generalization error as the forest building progresses.
- Handles missing data well.
- Scalable to large datasets.
Bagging

- The idea behind random forests can be applied to any classifier and is called **bagging**:
  - Choose M bootstrap samples.
  - Train M different classifiers on these bootstrap samples.
  - For a new query, average or take a majority vote.

- Other committee methods: Boosting (Freund & Schapire, 1995)
Classification: summary

- Classifiers we learned:
  - Nearest neighbors
  - Decision trees
  - Random forests
More classifiers:
- Naïve Bayes