More Than A Program

- Usually, we think a program is something written by an experienced person.
- Often, the program isn’t complete without “experience” of its own.
- Example: If I had to write a compression program, I could do ok. But, using the character-count statistics from the Gettysburg Address makes it possible to produce the most compact code.
Machine Learning

- *Machine learning* is the idea of writing programs that use data (experience) to create better programs than people can write directly.

- *Supervised learning* is a specific problem in which the experience is in the form of labeled examples and the learning system needs to learn to label novel examples.

Many Names

- classification learning
- statistical learning
- data mining
- concept learning
- discriminative learning
- supervised learning
- pattern recognition
Classification: Idea

• In its simplest form, a learner’s job is to produce a classifier.
• A classifier takes objects as input and assigns each one to a class.
• Most simply, objects are represented as vectors of features and classes are 0/1.

Example Classifier

• **Input**: A high school student
• **Output**: Will the student drop out of college?
• **Vector of Features**: Score on SATs, grades in Math/English/Science, age, parent’s income, years at current address, height
• Such a classifier *might* be useful as a tool for admissions or financial aid.
Learning: The Problem

- **Input**: A *training set* consisting of *labeled instances*, each of which is a feature vector and a desired class (1 = yes, 0 = no).
- **Output**: A classifier, which we hope will accurately assign new feature vectors to classes.
- A *learning algorithm* is a program that addresses this problem.

Wearing White

- My mom told me “never wear white shoes after labor day”.
- Well, she didn’t so much *tell* me as humiliate me if I violated the rule.
- Top part is training set.
- Bottom lines are instances to be classified.

<table>
<thead>
<tr>
<th>month</th>
<th>day</th>
<th>ok?</th>
</tr>
</thead>
<tbody>
<tr>
<td>2</td>
<td>14</td>
<td>1</td>
</tr>
<tr>
<td>7</td>
<td>4</td>
<td>1</td>
</tr>
<tr>
<td>8</td>
<td>30</td>
<td>1</td>
</tr>
<tr>
<td>9</td>
<td>4</td>
<td>1</td>
</tr>
<tr>
<td>9</td>
<td>5</td>
<td>0</td>
</tr>
<tr>
<td>10</td>
<td>31</td>
<td>0</td>
</tr>
<tr>
<td>12</td>
<td>25</td>
<td>0</td>
</tr>
<tr>
<td>10</td>
<td>9</td>
<td>?</td>
</tr>
<tr>
<td>3</td>
<td>30</td>
<td>?</td>
</tr>
</tbody>
</table>
Informal to Formal

• Not a formal problem at this point! Whether it does well on new examples isn’t directly measurable.
• One way to make it formal: find classifier with minimum number of mistakes on the training set.
• Usually not enough to create good classifiers: overfitting.

Learning Algorithms

• Decision trees
• Boosting
• Nearest neighbors
• Support Vector Machines
• Neural networks
• Naive Bayesian classifiers
• etc.
Neural Networks

• Background: http://en.wikipedia.org/wiki/Artificial_neural_network

• Inspired by brains, but ultimately a simple and flexible mathematical representation.

Basic Elements

• Each component of the feature vector becomes an *input unit*.

• An additional input unit is added that always has an input of 1.

• Each input unit is connected to a sum unit with a *weight*. The input value is multiplied by the weight and all are summed.

• If the sum is greater than zero, the output is “1”, else “0”.
In "Math"

- $i$ stands for a feature.
- $x_i$ is the input on feature $i$ (the $i$th component of the feature vector $x$).
- $w_i$ is the weight for feature $i$.
- $y$ is the output of the network.

```python
def classify(w, x):
    return sum([x[i] * w[i] for i in range(len(x))]) > 0
```
Autonomous Driving

- ALVINN could mimic a driver’s decisions and, so, follow roads.

Loan Applications

- Will loan applicant default on the loan?
Character Recognition

- Is it a “9” I see?
- NN systems in place reading roughly half of all checks in US.

Speech Recognition

- Learning systems (sometimes NNs) play an important role in high-quality speech recognizers.
Game Playing

- One of the first uses was in a checker playing program (learn the evaluation function).
- Central part of the world’s best backgammon playing program.

Training A Perceptron

- Given a training set, would like a way to set the weights to correctly classify the labeled instances.
- Networks with one layer of nets are very well understood at this point.
- I’ll describe a demonstrate one simple rule: the perceptron training procedure.
  - http://en.wikipedia.org/wiki/Perceptron
### Start With “Math”

- $i$ is a feature.
- $x$ is the input ($x_i$ is the $i$th component).
- $w_i$ is weight vector.
- $y = (\sum_i x_i \times w_i) > 0$.
- $t$ is the target (right output for $x$).
- $\alpha$ is the learning rate (amount to change the weights).
- $\Delta w_i$ is the amount we plan to change weight $i$.
- For all features $i$:
  - $\Delta w_i = \alpha(t - y)x_i$

### Understanding The Rule

<table>
<thead>
<tr>
<th>$t$</th>
<th>$y$</th>
<th>action of $\Delta w_i = \alpha(t - y)x_i$</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>$t - y$ is zero (right answer). Nothing needs to change.</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>$t - y$ is $-1$ (output too big). If $x_i &gt; 0$, $w_i$ is decreased to make sum smaller.</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>$t - y$ is $1$ (output too small). If $x_i &gt; 0$, $w_i$ is increased to make sum bigger.</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>$t - y$ is zero (right answer). Nothing needs to change.</td>
</tr>
</tbody>
</table>
Using The Rule

- Start off with the weight vector set to an arbitrary value (all zeros, say).
- For each example in the training set, apply the rule, changing the weights as needed.
- If weights have changed, repeat.

Simple Example

```python
def train(d):
    w = [-1 for i in d[0]]
    wNew = [0 for i in d[0]]
    while w != wNew:
        w = wNew
        wNew = perceptronRule(d, w)
    return w

def perceptronRule(d, w):
    for i in d:
        n = len(i)
        t = i[n-1]
        x = i[0:n-1] + [1]
        y = classify(w, x)
        w = [w[i] + (t-y) * x[i] for i in range(n)]
    return w

orSet = [[0,0,0],[0,1,1],[1,0,1],[1,1,1]]
w = train(orSet)
print w
[1, 1, 0]```
Rule Properties

• Stops changing when stops making mistakes.
• If there is a way of setting the weights that makes no mistakes, it will be found eventually.
• Otherwise, might bounce around.
• Very related to gradient descent and hill-climbing: local changes to reduce error score.

More Examples

• Run code on “orSet”, “notASet”, “bDaySet”, “whiteSet”, “hotSet”.
• What about “comfortableSet”, “virgoSet”, and “xorSet”?
Linear Separability

- Perceptron can only achieve perfect score if data is \textit{linearly separable}. That is, if we visualize the instances as points in a high dimensional space, there needs to be linear surface that separates the positive and negative examples.

Not Linearly Separable

- Minsky and Papert pointed out that data sets that are not linearly separable can cause the perceptron much headaches.
Elaborations

• Backprop can train multilayer networks and learn xor.
• Some work on recurrent networks (outputs sent back as inputs).
• Lots of statistically grounded algorithms now available.

Next Time

• Finish discussion of machine learning by looking at reinforcement learning