Supervised Learning

George Konidaris
gdk@cs.brown.edu

Fall 2018
Machine Learning

Subfield of AI concerned with *learning from data*.

Broadly, using:

- *Experience*
- To Improve *Performance*
- On Some *Task*

*(Tom Mitchell, 1997)*
Supervised Learning

Input:

\[ X = \{x_1, \ldots, x_n\} \] inputs
\[ Y = \{y_1, \ldots, y_n\} \] labels

Learn to predict new labels.
Given \( x \): \( y \)

training data
“A breakthrough in machine learning would be worth ten Microsofts.”

- Bill Gates
Classification vs. Regression

If the set of labels $Y$ is discrete:
- Classification
- Minimize number of errors

If $Y$ is real-valued:
- Regression
- Minimize sum squared error

Today we focus on classification.
Supervised Learning

Formal definition:

Given training data:
\[ X = \{x_1, \ldots, x_n\} \text{ inputs} \]
\[ Y = \{y_1, \ldots, y_n\} \text{ labels} \]

Produce:
Decision function \( f : X \rightarrow Y \)

That minimizes error:
\[ \sum_i \text{err}(f(x_i), y_i) \]
Test/Train Split

Minimize error measured on what?
- Don’t get to see future data.
- Could use test data … but! **may not generalize.**

General principle:
**Do not measure error on the data you train on!**
Test/Train Split

Methodology:
• Split data into **training set** and **test set**.
• Fit $f$ using **training set**.
• Measure error on **test set**.

**Always do this.**
Test/Train Split

What if you choose unluckily?
And aren’t we wasting data?

\textit{k-fold Cross Validation:}
\begin{itemize}
\item Common alternative
\item Repeat \textit{k} times:
  \begin{itemize}
  \item Partition data into train \((n - n/k)\) and test \((n/k)\) data sets
  \item Train on training set, test on test set
  \item Average results across \textit{k} choices of test set.
  \end{itemize}
\end{itemize}
Key Idea: Hypothesis Space

Typically
- Fixed \textit{representation} of classifier.
- Learning algorithm constructed to match.

Representation induces class of functions $F$, from which to find $f$.
- $F$ is known as the \textit{hypothesis space}.
- Tradeoff: power vs. expressibility vs. data efficiency.
- Not every $F$ can represent every function.

$F = \{f_1, f_2, \ldots, f_n\}$
- Set of possible functions that can be returned
- Typically infinite set (not always)
- Learning is finding $f_i \in F$ that minimizes error.
Key Idea: Decision Boundary

Boundary at which label changes
Decision Trees

Let’s assume:

• Two classes (true and false).
• Input: vector of discrete values.

What’s the simplest thing we could do?
How about some if-then rules?

Relatively simple classifier:

• Tree of tests.
• Evaluate test for for each $x_i$, follow branch.
• Leaves are class labels.
Decision Trees

$x_i = [a, b, c]$  
each boolean

- **a?**
  - **true**
    - **b?**
      - **true**
        - $y = 1$
      - **false**
        - $y = 2$
  - **false**
    - **c?**
      - **true**
        - **b?**
          - **true**
            - $y = 2$
          - **false**
            - $y = 1$
      - **false**
        - $y = 1$
Decision Trees

How to make one?

Given
\[ X = \{ x_1, \ldots, x_n \} \]
\[ Y = \{ y_1, \ldots, y_n \} \]

repeat:
• if all the labels are the same, we have a leaf node.
• pick an attribute and split data bases on its value.
• recurse on each half.

If we run out of splits, and data not perfectly in one class, then take a max.
Decision Trees

\[ a? \]

<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th>L</th>
</tr>
</thead>
<tbody>
<tr>
<td>T</td>
<td>F</td>
<td>T</td>
<td>1</td>
</tr>
<tr>
<td>T</td>
<td>T</td>
<td>F</td>
<td>1</td>
</tr>
<tr>
<td>T</td>
<td>F</td>
<td>F</td>
<td>1</td>
</tr>
<tr>
<td>F</td>
<td>T</td>
<td>F</td>
<td>2</td>
</tr>
<tr>
<td>F</td>
<td>T</td>
<td>T</td>
<td>2</td>
</tr>
<tr>
<td>F</td>
<td>T</td>
<td>F</td>
<td>2</td>
</tr>
<tr>
<td>F</td>
<td>F</td>
<td>T</td>
<td>1</td>
</tr>
<tr>
<td>F</td>
<td>F</td>
<td>F</td>
<td>1</td>
</tr>
</tbody>
</table>
### Decision Trees

#### Example

**Decision Tree**
- **Question**: `a?`
- **Answer**: `true` leads to `y=1`

<table>
<thead>
<tr>
<th>$A$</th>
<th>$B$</th>
<th>$C$</th>
<th>$L$</th>
</tr>
</thead>
<tbody>
<tr>
<td>T</td>
<td>F</td>
<td>F</td>
<td>1</td>
</tr>
<tr>
<td>T</td>
<td>T</td>
<td>F</td>
<td>1</td>
</tr>
<tr>
<td>T</td>
<td>F</td>
<td>F</td>
<td>1</td>
</tr>
<tr>
<td>F</td>
<td>T</td>
<td>F</td>
<td>2</td>
</tr>
<tr>
<td>F</td>
<td>T</td>
<td>T</td>
<td>2</td>
</tr>
<tr>
<td>F</td>
<td>T</td>
<td>F</td>
<td>2</td>
</tr>
<tr>
<td>F</td>
<td>F</td>
<td>T</td>
<td>1</td>
</tr>
<tr>
<td>F</td>
<td>F</td>
<td>F</td>
<td>1</td>
</tr>
</tbody>
</table>
Decision Trees

- a?
  - true: \(y=1\)
  - false: b?

### Table

<table>
<thead>
<tr>
<th>A</th>
<th>B</th>
<th>C</th>
<th>L</th>
</tr>
</thead>
<tbody>
<tr>
<td>T</td>
<td>F</td>
<td>T</td>
<td>1</td>
</tr>
<tr>
<td>T</td>
<td>T</td>
<td>F</td>
<td>1</td>
</tr>
<tr>
<td>T</td>
<td>F</td>
<td>F</td>
<td>1</td>
</tr>
<tr>
<td>F</td>
<td>T</td>
<td>F</td>
<td>2</td>
</tr>
<tr>
<td>F</td>
<td>T</td>
<td>T</td>
<td>2</td>
</tr>
<tr>
<td>F</td>
<td>T</td>
<td>F</td>
<td>2</td>
</tr>
<tr>
<td>F</td>
<td>F</td>
<td>T</td>
<td>1</td>
</tr>
<tr>
<td>F</td>
<td>F</td>
<td>F</td>
<td>1</td>
</tr>
</tbody>
</table>
Decision Trees

\[
\begin{array}{c}
\text{a?} \\
\text{true} \\
\text{y=1} \\
\text{false} \\
\text{b?} \\
\text{true} \\
\text{y=2} \\
\end{array}
\]

<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th>L</th>
</tr>
</thead>
<tbody>
<tr>
<td>T</td>
<td>F</td>
<td>T</td>
<td>1</td>
</tr>
<tr>
<td>T</td>
<td>T</td>
<td>F</td>
<td>1</td>
</tr>
<tr>
<td>T</td>
<td>F</td>
<td>F</td>
<td>1</td>
</tr>
<tr>
<td>F</td>
<td>T</td>
<td>F</td>
<td>2</td>
</tr>
<tr>
<td>F</td>
<td>T</td>
<td>T</td>
<td>2</td>
</tr>
<tr>
<td>F</td>
<td>F</td>
<td>F</td>
<td>2</td>
</tr>
<tr>
<td>F</td>
<td>F</td>
<td>T</td>
<td>1</td>
</tr>
<tr>
<td>F</td>
<td>F</td>
<td>F</td>
<td>1</td>
</tr>
</tbody>
</table>

\[a? \rightarrow \text{true} \rightarrow \text{y=1} \]
\[a? \rightarrow \text{false} \rightarrow \text{b?} \]
\[b? \rightarrow \text{true} \rightarrow \text{y=2} \]
Attribute Picking

Key question:
- Which attribute to split over?

Information contained in a data set:

\[ I(A) = -f_1 \log_2 f_1 - f_2 \log_2 f_2 \]

How many “bits” of information do we need to determine the label in a dataset?

Pick the attribute with the max information gain:

\[ Gain(B) = I(A) - \sum_i f_i I(B_i) \]
Example

\[ Gain(B) = I(A) - \sum f_i I(B_i) \]

\[ I(A) = -f_1 \log_2 f_1 - i f_2 \log_2 f_2 \]

<p>| | | | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>A</td>
<td>B</td>
<td>C</td>
<td>L</td>
<td></td>
</tr>
<tr>
<td>---</td>
<td>---</td>
<td>---</td>
<td>---</td>
<td></td>
</tr>
<tr>
<td>T</td>
<td>F</td>
<td>T</td>
<td>1</td>
<td></td>
</tr>
<tr>
<td>T</td>
<td>T</td>
<td>F</td>
<td>1</td>
<td></td>
</tr>
<tr>
<td>T</td>
<td>F</td>
<td>F</td>
<td>1</td>
<td></td>
</tr>
<tr>
<td>F</td>
<td>T</td>
<td>F</td>
<td>2</td>
<td></td>
</tr>
<tr>
<td>F</td>
<td>T</td>
<td>T</td>
<td>2</td>
<td></td>
</tr>
<tr>
<td>F</td>
<td>T</td>
<td>F</td>
<td>2</td>
<td></td>
</tr>
<tr>
<td>F</td>
<td>F</td>
<td>T</td>
<td>1</td>
<td></td>
</tr>
<tr>
<td>F</td>
<td>F</td>
<td>F</td>
<td>1</td>
<td></td>
</tr>
</tbody>
</table>
Decision Trees

What if the inputs are real-valued?
- Have inequalities rather than equalities.
- Can repeat variables.

```
a > 3.1
  true
  y = 1
  false
  b < 0.6?
    true
    y = 2
    false
    y = 1
```
Hypothesis Class

What is the hypothesis class for a decision tree?

- Discrete inputs?
- Real-valued inputs?
The Perceptron

If your input \((x_i)\) is real-valued ... explicit decision boundary?
The Perceptron

If \( x = [x(1), \ldots, x(n)] \):
- Create an \( n \)-d line
- Slope for each \( x(i) \)
- Constant offset

\[
f(x) = \text{sign}(w \cdot x - c)
\]

Gradient: \( \nabla f(x) = w \)
Offset: \( c \)

\[ y = wx + c \]
The Perceptron

Which side of a line are you on?

\[ w \cdot x = ||w|| ||x|| \cos(\theta) \]
The Perceptron

How do you reduce error?

e = (y_i - (w \cdot x_i + c))^2

\frac{\partial e}{\partial w_j} = -2(y_i - (w_i \cdot x_i + c))x_i(j)

descend this gradient to reduce error
The Perceptron Algorithm

Assume you have a *batch* of data:

\[ X = \{x_1, \ldots, x_n\} \]
\[ Y = \{y_1, \ldots, y_n\} \]

set \( w, c \) to 0.

for each \( x_i \):

**predict** \( z_i = \text{sign}(w.x_i + c) \)

if \( z_i \neq y_i \):

\[ w = w + \alpha(y_i - z_i)x_i \]

converges if data is linearly separate

learning rate

https://www.youtube.com/watch?v=KcmIQQ3zWYro
Probabilities

What if you want a **probabilistic classifier**?

Instead of `sign`, squash output of linear sum down to `[0, 1]`:

\[ \sigma(w \cdot x + c) \]

Resulting algorithm: **logistic regression**.
Frank Rosenblatt

Built the *Mark I* in 1960.
Perceptrons

What can’t you do?
Perceptrons

Marvin L. Minsky
Seymour A. Papert

Expanded Edition

1969
Neural Networks

\[ \sigma(w \cdot x + c) \]

logistic regression
Neural Networks

output layer

input layer

hidden layer
Neural Networks

\[
\begin{align*}
\sigma(w_1^{o1}h_1 + w_2^{o1}h_2 + w_3^{o1}h_3 + w_4^{o1}) & \quad \sigma(w_1^{o2}h_1 + w_2^{o2}h_2 + w_3^{o2}h_3 + w_4^{o2}) \\
\text{value computed} & \\
\end{align*}
\]

\[
\begin{align*}
h_1 & = \sigma(w_1^{h1}x_1 + w_2^{h1}x_2 + w_3^{h1}) \\
\text{value computed} & \\
\end{align*}
\]

\[
\begin{align*}
\sigma(w_1^{h2}x_1 + w_2^{h2}x_2 + w_3^{h2}) & \quad \sigma(w_1^{h3}x_1 + w_2^{h3}x_2 + w_3^{h3}) \\
\text{feed forward} & \\
\end{align*}
\]

\[
\begin{align*}
x_1, x_2 & \in [0, 1] \\
\text{input layer} & \\
\end{align*}
\]
Neural Networks

probability of class 1

$\sigma(w_1 x_1 + w_2 x_2 + w_3)$

probability of class 2

$\sigma(w_1 \tilde{x}_1 + w_2 \tilde{x}_2 + w_3)$

weights (parameters)

input data

$x_1, x_2 \in [0, 1]$
Neural Classification

A neural network is just a parametrized function: \( y = f(x, w) \)

How to train it?

Write down an error function:

\[
(y_i - f(x_i, w))^2
\]

Minimize it! (w.r.t. \( w \))
Neural Classification

Recall that the squashing function is defined as:

\[
\sigma(t) = \frac{1}{1 + e^{-t}}
\]

\[
\frac{\partial \sigma(t)}{\partial t} = \sigma(t)(1 - \sigma(t))
\]
Neural Classification

OK, so we can minimize error using gradient descent.

To do so, we must calculate $\frac{\partial e}{\partial w_i}$ for each $w_i$.

How to do so? Easy for output layers:

$$\frac{\partial e}{\partial w_i} = \frac{\partial (y_i - o_i)^2}{\partial w_i} = 2(y_i - o_i) \frac{\partial (y_i - o_i)}{\partial w_i} = 2(o_i - y_i) o_i (1 - o_i)$$

| chain rule |

Interior weights: repeat chain rule application.
Backpropagation

This algorithm is called *backpropagation*.

Bryson and Ho, 1969
Deep Neural Networks

\[
\begin{align*}
&h_{n1} \\
&h_{n2} & & h_{n3}
\end{align*}
\]

\[
\begin{align*}
&o_1 \\
&o_2
\end{align*}
\]

\[
\begin{align*}
&h_{n1} \\
&h_{n2} & & h_{n3}
\end{align*}
\]

\[
\begin{align*}
&h_{11} \\
&h_{12} & & h_{13}
\end{align*}
\]

\[
\begin{align*}
&x_1 \\
&x_2
\end{align*}
\]
Applications

• Fraud detection
• Internet advertising
• Friend or link prediction
• Sentiment analysis
• Face recognition
• Spam filtering
Applications

MNIST Data Set
Training set: 60k digits
Test set: 10k digits