Machine Learning

Professor Khaled Rasheed

Lecture Notes

Introduction
What is Machine Learning?

Simon Says:

“Learning denotes changes in the system that enable the system to do the same task ... more effectively next time.”

Example Tasks:

- Classify an object as an instance (or non-instance) of a general concept. (Inductive Concept Learning.)
- Solve a search problem. (Speedup Learning.)
Types of Learning

- Learning by being told:
  
  *Teacher states the rules of grammar.*
  
  *Knowledge Compilation.*

- Learning from examples:
  
  *Teacher shows examples of good and bad sentences.*
  
  *Inductive Learning.*

- Learning by discovery:
  
  *Student talks and teacher corrects his grammar.*
  
  *Automated Theory Formation.*
Inductive Argument

Given:

\[ P(a_1) \land Q(a_1) \quad \neg P(b_1) \land \neg Q(b_1) \quad \neg P(c_1) \land Q(c_1) \]

\[ P(a_2) \land Q(a_2) \quad \neg P(b_2) \land \neg Q(b_2) \quad \neg P(c_2) \land Q(c_2) \]

\[ \ldots \]

\[ P(a_i) \land Q(a_i) \quad \neg P(b_m) \land \neg Q(b_m) \quad \neg P(c_n) \land Q(c_n) \]

Conclude:

\[(\forall x)P(x) \Rightarrow Q(x)\]
The Old Problem of Induction

• Why are inductive arguments justified?
• Hume: Because they have worked in the past.
• Russell: That’s using induction to justify induction.
• Goodman: Inductive arguments are not justified.
How does an intelligent agent choose among many possible inductive generalizations of his observations?
Goodman’s Paradox

Examples:

\[ \text{Emerald}(a) \land \text{Green}(a) \text{ (Sunday)}. \]
\[ \text{Emerald}(b) \land \text{Green}(b) \text{ (Monday)}. \]
\[ \text{Emerald}(c) \land \text{Green}(c) \text{ (Tuesday)}. \]

Generalizations:

\[ (\forall x) \text{Emerald}(x) \Rightarrow \text{Green}(x) \]
\[ (\forall x) \text{Emerald}(x) \Rightarrow \text{Grue}(x) \]

An object is “grue” if it is green on Sunday, Monday and Tuesday, but is blue for the rest of the week.
“Any criteria for choosing one concept description over another, other than strict consistency with the training examples.” (Mitchell)

- E.g., A biased concept description language.
- E.g., A biased learning algorithm.
Occam’s Razor

• Choose the simplest hypothesis.
• That accounts for the observation.
• “Green” is simpler than “Grue”
• So choose a hypothesis involving the term “Green”.
Given Training Data:

- Positive Examples:
  \((ObjectDescription, +Label)\)
- Negative Examples:
  \((ObjectDescription, -Label)\)

Find rule for predicting whether future examples are positive or negative.

“Concept Membership Rule”.
Definitions

• Instance Description Language:
  – Language for describing example objects.
  – E.g., Boolean, Integer or Real Vector.

• Concept Description Language:
  – Language for describing concepts.
  – E.g., Conjunctive: \( \land (v_i = k_i) \).
Decision Trees

- A concept description language.
- For instances represented as feature vectors.
- Each internal node checks the value of a feature.
- Branches are labeled by possible values.
- Leaves are labeled:
  - "+" indicates member of the concept.
  - "-" indicates not a member of the concept.
Generic Decision Tree

Test 1

Test 2

Test N

Label
A Decision Tree for Discrete Feature Vectors

- Color?
  - Green
  - -
  - Red
  - +
  - Blue
  - +

- Size?
  - Big
  - +
  - Small
  - -
• Goal: Find a smallest tree that correctly classifies all the training examples.
• NP Hard: (Hyafil and Rivest, 1976).
• We must use heuristics if CPU time is limited.
Given a set of labeled instances:

1. Find a feature that “best” divides the instances into uniform sets.
2. Recursively call ID3 on each subset.

What does “best” mean?
Using Information Theory

- Let $S$ be a set of unclassified instances.
- Assume we know the fractions of positive and negative instances in $S$:
  \[ p^+ = \text{Fraction of positive instances.} \]
  \[ p^- = \text{Fraction of negative instances.} \]
- Now someone tells us the classification of some instance in set $S$.
- What is the information value of this new fact?
  \[ I(p^+, p^-) = -(p^+)\log_2(p^+) - (p^-)\log_2(p^-) \]
- The information value of the fact is the Entropy of the instances.
Information Content Function

$I(p_+ (1-p_+))$

Fraction $p_+$ of Positive Instances
Finding a “Best” Feature

1. For each feature $f$ do:
   (a) Use $f$ to partition instances into sets $S_1, \ldots, S_n$.
   (b) For each set $S_i$, determine $p_i^+$ and $p_i^-$.
   (c) Let
   \[ Gain(f) = I(p^+, p^-) - \sum_{i=1}^{n} \left( \frac{|S_i|}{|S|} \right) I(p_i^+, p_i^-). \]

2. Choose a feature $f$ with highest value of $Gain(f)$. 
ID3(INSTANCES, FEATURES):

1. If all INSTANCES are positive, then Return(Positive-Leaf).
2. If all INSTANCES are negative, then Return(Negative-Leaf).
3. Let BEST = Maxarg (f in FEATURES) Gain(f).
4. Let R be the root of a decision tree splitting on BEST.
5. For each value V of BEST do:
   a. Let S = Subset of INSTANCES with BEST = V.
   b. Attach to R the subtree ID3(S,FEATURES - {BEST}).
An ID3 Example

<table>
<thead>
<tr>
<th>Color</th>
<th>Shape</th>
<th>Size</th>
<th>Class</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Red</td>
<td>Big</td>
<td>+</td>
</tr>
<tr>
<td>2</td>
<td>Blue</td>
<td>Big</td>
<td>+</td>
</tr>
<tr>
<td>3</td>
<td>Red</td>
<td>Small</td>
<td>-</td>
</tr>
<tr>
<td>4</td>
<td>Green</td>
<td>Small</td>
<td>-</td>
</tr>
<tr>
<td>5</td>
<td>Red</td>
<td>Big</td>
<td>+</td>
</tr>
<tr>
<td>6</td>
<td>Green</td>
<td>Big</td>
<td>-</td>
</tr>
</tbody>
</table>

Initial Call:

$$\text{ID3}\left(\{1,2,3,4,5,6\},\{\text{Color, Shape, Size}\}\right)$$
Finding the Best Feature

\[
Gain(\text{Color}) = I(3/6, 3/6) \\
\quad - (3/6) \cdot I(2/3, 1/3) \\
\quad - (1/6) \cdot I(1/1, 0/1) \\
\quad - (2/6) \cdot I(0/2, 2/2) \\
= 1 - 0.459 = 0.541
\]

\[
Gain(\text{Shape}) = I(3/6, 3/6) \\
\quad - (3/6) \cdot I(2/3, 1/3) \\
\quad - (3/6) \cdot I(1/3, 2/3) \\
= 1 - 0.918 = 0.082
\]

\[
Gain(\text{Size}) = I(3/6, 3/6) \\
\quad - (2/3) \cdot I(3/4, 1/4) \\
\quad - (1/3) \cdot I(0/2, 2/2) \\
= 1 - 0.541 = 0.459
\]

Color is best!
Tree Returned by ID3

```
Color?
   /
  /   
Green - 
/ 
Big +

Red

/ 
+ 

Blue

/ 
+ 

Small -
```
• One rule for each path from root to a leaf node.
• Antecedent: Conjunction of all decisions on path.
• Consequent: Label of the leaf node.

\[
\begin{align*}
\text{Color} = \text{Green} & \Rightarrow - \\
\text{Color} = \text{Blue} & \Rightarrow + \\
\text{Color} = \text{Red} \land \text{Size} = \text{Big} & \Rightarrow + \\
\text{Color} = \text{Red} \land \text{Size} = \text{Small} & \Rightarrow -
\end{align*}
\]
• Similar to handling of discrete feature vectors.
• For each internal, splitting node:
  – Choose best feature.
  – Choose direction \(<\) or \(\geq\) of test.
  – Choose threshold \(k\) of test: \(f < k\) or \(f \geq k\).
• Data Rich: Separate Training and Test Sets.
  – Select a random subset of the training examples.
  – Withold it from the learning algorithm.
  – Use it as an unbiased test set.

• Data Poor: Cross Validation
  – Divide data into $n$ subsets.
  – Learn one concept description for each collection of $n - 1$ subsets.
  – Test each concept description on the corresponding withheld subset.
  – Use average of $n$ error rates as an estimate of the accuracy when learning from all the data.
• Also Known As:
  – Connectionist Machines.
  – Parallel Distributed Processing.

• Early Work: Perceptrons.

• Current Work: Backpropagation.
• Rosenblatt, 1958.
• Very simple computing device.
• Very simple learning device.
• Inspired by rough analogy with neuron.
Perceptron Output Function

\[
A = \begin{cases} 
1 & \text{If } W_1X_1 + \ldots + W_nX_n > T \\
0 & \text{Otherwise.}
\end{cases}
\]
Examples
Thresholds as Weights

Notice that

\[ W_1X_1 + \ldots + W_nX_n > T \]

is equivalent to

\[ W_1X_1 + \ldots + W_nX_n - T > 0 \]

or

\[ W_1X_1 + \ldots + W_nX_n - T \cdot 1 > 0 \]
Thresholds as Weights

- Pretend each node has an extra input whose value is always 1 and whose weight is $-T$, called the "bias".
- Learning updates the bias just like all the other weights.
Thresholds as Weights
• Learn values of weights from I/O pairs.
• Start with random weights.
• Load training example’s input.
• Observe computed output.
• Compare to desired output.
• Modify weights to reduce difference.
• Iterate over all training examples.
• Terminate when weights stop changing.
Perceptron Learning Rule

\[ \Delta W_i = \eta(D - A)X_i \]

\( X_i \) is a node’s input.
\( W_i \) is the corresponding weight.
\( \Delta W_i \) is the change in weight.
\( D \) is the desired output.
\( A \) is the actual observed output.
\( \eta \) is the learning rate.
Learning the *Or* Function

\[ \Delta W_i = 0.2(D - A)X_i \]

\[
\begin{array}{ccc}
A & \quad & \\
\downarrow & \quad & \downarrow \\
W_1 = 0.1 & \quad & W_3 = -0.8 \\
\downarrow & \quad & \downarrow \\
W_2 = 0.5 & \quad & \\
\quad & \quad & X_1 \quad X_2 \quad X_3 = 1
\end{array}
\]
Learning the *Or* Function

<table>
<thead>
<tr>
<th>$X_1$</th>
<th>$X_2$</th>
<th>$D$</th>
<th>$A$</th>
<th>$\Delta W_1$</th>
<th>$W_1$</th>
<th>$\Delta W_2$</th>
<th>$W_2$</th>
<th>$\Delta W_3$</th>
<th>$W_3$</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0.1</td>
<td>0.1</td>
<td>0.5</td>
<td>0.5</td>
<td>-0.8</td>
<td>-0.8</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>0.1</td>
<td>0.2</td>
<td>0.7</td>
<td>0.7</td>
<td>-0.6</td>
<td>-0.6</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0.3</td>
<td>0.3</td>
<td>0.7</td>
<td>0.7</td>
<td>-0.4</td>
<td>-0.4</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>0.3</td>
<td>0.7</td>
<td>0</td>
<td>0</td>
<td>-0.4</td>
<td>-0.4</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0.3</td>
<td>0.7</td>
<td>0</td>
<td>0</td>
<td>-0.4</td>
<td>-0.4</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>0.3</td>
<td>0.7</td>
<td>0</td>
<td>0</td>
<td>-0.4</td>
<td>-0.4</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0.5</td>
<td>0.7</td>
<td>0</td>
<td>0</td>
<td>-0.2</td>
<td>-0.2</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>0.5</td>
<td>0.7</td>
<td>0</td>
<td>0</td>
<td>-0.2</td>
<td>-0.2</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0.5</td>
<td>0.7</td>
<td>0</td>
<td>0</td>
<td>-0.2</td>
<td>-0.2</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>0.5</td>
<td>0.7</td>
<td>0</td>
<td>0</td>
<td>-0.2</td>
<td>-0.2</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>0.5</td>
<td>0.7</td>
<td>0</td>
<td>0</td>
<td>-0.2</td>
<td>-0.2</td>
</tr>
</tbody>
</table>
Perceptron Convergence Theorem

“If a set of I/O pairs is learnable, then the Perceptron Learning Rule will find the necessary weights.”

The output function $W_1X_1 + \ldots + W_nX_n > T$ defines a hyperplane that splits the input space into two half spaces.
Linear Separability: The \textit{XOR} Function

Can a single line separate the two classes?
Solution: Hidden Units

Input Units

Hidden Units

Output Units

I1

I2

I3

O1

O2
• Network contains hidden units.
• Each unit obeys “Sigmoidal Activation Function”.

\[
A = \frac{1}{1+e^{-(W_1X_1 + \ldots + W_nX_n - T)}}
\]
Behavior of Sigmoid Function

The sigmoid function is defined as:

\[ S = W_1 X_1 + ... + W_N X_N \]

where \( S \) is the output, \( W \) are the weights, and \( X \) are the inputs.
Implementation of the $XOR$ Function
Backpropagation Learning Rule for Weights of Edges from Hidden Units to an Output Unit $j$

$$\Delta W_{ij} = \eta A_j(1 - A_j)(D_j - A_j)H_i$$

$H_i$ is a node’s input (a hidden node’s output).
$W_{ij}$ is the corresponding weight.
$\Delta W_{ij}$ is the change in weight.
$D_j$ is the desired output.
$A_j$ is the actual observed output.
$\eta$ is the learning rate.
Backpropagation Learning Rule for Weights of Edges from Input Units to a Hidden Unit $i$

$$\Delta W_{ki} = \eta H_i (1 - H_i) E_i X_k$$

$X_k$ is a node’s input.

$W_{ki}$ is the corresponding weight.

$\Delta W_{ki}$ is the change in weight.

$E_i = \sum_{j=1}^{n} W_{ij} A_j (1 - A_j) (D_j - A_j)$ is the propagated error from the $n$ output units.

$H_i$ is the actual observed output.

$\eta$ is the learning rate.
Motivation for the Backpropagation Learning Rules

- The weights $\bar{W} = (W_1, \ldots, W_n)$ define a point in an $n$-dimensional Euclidean space.
- Each point $\bar{W}$ in the space defines a network.
- Each point $\bar{W}$ has an associated error rate:
  \[ E = \frac{1}{2} \sum_i (D_i - A_i)^2 \]
- We compute the gradient $\nabla E$ with respect to $\bar{W}$.
- Each learning iteration changes $\bar{W}$ to $\bar{W} - \eta \nabla E$.
- A "Gradient Descent" algorithm.
Applications of Backpropagation Networks

- NETtalk converts character strings to phonemes.
- Neurogammon won the 1989 Computer Olympiad.
- ALVINN steers a vehicle along a single lane highway.
Advantages and disadvantages of Neural Networks

Advantages:

• Generalization capability.
• Low sensitivity to noise.

Disadvantages:

• Relative expressiveness.
• Computational efficiency.
• Transparency (black box).
• Hard to use prior knowledge.
• **Noise:** Incorrect data (label errors or attribute errors)

• **Overfitting:** Learner does very well on the training data but poorly on the testing data (unseen examples).

• Noise is the most common cause for overfitting.
Some solutions

- Decision tree pruning: simplify tree by shortening some branches with mostly positive (negative) examples.
- Use of less expressive concept description languages, such as linear classifiers (Perceptrons).
- Stopping the learning early when overfitting is detected.
Case Based Learning

- Very simple model for learning.
- Stores labeled examples.
- Classifies unseen examples based on their similarity to stored examples.
- Uses distance functions to measure similarity.
Nearest Neighbor

- Gives unseen example the label of the nearest stored example.
- Outcome depends on the distance function used.
- **Euclidean Distance** is suitable for continuous features.
- **Hamming Distance** is suitable for binary (discrete) features.

**Problem:** Very sensitive to noise.
K-Nearest Neighbors

• Finds the K nearest neighbors of the unseen example among the stored examples.

• Uses the labels of these neighbors to label the unseen example.

• **Voting K-Nearest Neighbors** labels the unseen example with the majority label among the K nearest neighbors.

• **Weighted K-Nearest Neighbors** uses distances to weigh the votes.
Batch vs. Incremental learning

- **Batch learning**: considers all training examples simultaneously.
- Examples: Backpropagation, ID3.
- **Incremental learning**: tries to update the old hypothesis whenever a new example arrives.
- Examples: Candidate elimination, Case based learning.