• Willow Garage: PR2
Mini-Max

Player
0

Opponent
1

Player
2

Opponent
3

Player
4
Learning Algorithm

- Uses training values for the target function to induce a hypothesized definition that fits these examples and hopefully generalizes to unseen examples.
- In statistics, learning to approximate a continuous function is called \textit{regression}.
- Attempts to minimize some measure of error (\textit{loss function}) such as \textit{mean squared error}:

\[
E = \frac{\sum_{b \in B} [V_{\text{train}}(b) - \hat{V}(b)]^2}{|B|}
\]
Least Mean Squares (LMS) Algorithm

- A gradient descent algorithm that incrementally updates the weights of a linear function in an attempt to minimize the mean squared error

Until weights converge:

For each training example \( b \) do:

1) Compute the absolute error:

\[
error(b) = V_{\text{train}}(b) - \tilde{V}(b)
\]

2) For each board feature, \( f_i \), update its weight, \( w_i \):

\[
w_i = w_i + c \cdot f_i \cdot error(b)
\]

for some small constant (learning rate) \( c \)
LMS Discussion

• Intuitively, LMS executes the following rules:
  – If the output for an example is correct, make no change.
  – If the output is too high, lower the weights proportional to the values of their corresponding features, so the overall output decreases
  – If the output is too low, increase the weights proportional to the values of their corresponding features, so the overall output increases.

• Under the proper weak assumptions, LMS can be proven to eventually converge to a set of weights that minimizes the mean squared error.
Lessons Learned about Learning

• Learning can be viewed as using direct or indirect experience to approximate a chosen target function.

• Function approximation can be viewed as a search through a space of hypotheses (representations of functions) for one that best fits a set of training data.

• Different learning methods assume different hypothesis spaces (representation languages) and/or employ different search techniques.
Various Function Representations

• Numerical functions
  – Linear regression
  – Neural networks
  – Support vector machines

• Symbolic functions
  – Decision trees
  – Rules in propositional logic
  – Rules in first-order predicate logic

• Instance-based functions
  – Nearest-neighbor
  – Case-based

• Probabilistic Graphical Models
  – Naïve Bayes
  – Bayesian networks
  – Hidden-Markov Models (HMMs)
  – Probabilistic Context Free Grammars (PCFGs)
  – Markov networks
Various Search Algorithms

- Gradient descent
  - Perceptron
  - Backpropagation

- Dynamic Programming
  - HMM Learning
  - PCFG Learning

- Divide and Conquer
  - Decision tree induction
  - Rule learning

- Evolutionary Computation
  - Genetic Algorithms (GAs)
  - Genetic Programming (GP)
  - Neuro-evolution
Evaluation of Learning Systems

• Experimental
  – Conduct controlled cross-validation experiments to compare various methods on a variety of benchmark datasets.
  – Gather data on their performance, e.g. test accuracy, training-time, testing-time.
  – Analyze differences for statistical significance.

• Theoretical
  – Analyze algorithms mathematically and prove theorems about their:
    • Computational complexity
    • Ability to fit training data
    • Sample complexity (number of training examples needed to learn an accurate function)
Generalization vs. Memorization

• But there’s *No Free Lunch*…
  – David Wolpert & William G. Macready
  – On a particular problem, different search algorithms may obtain different results, but over all problems, they are indistinguishable.
  – If an algorithm achieves superior results on some problems, it must pay with inferiority on other problems.

– What’s *your* bias?
Guess the Number

• Similar to 20 questions…
Decision Trees

- Tree-based classifiers for instances represented as feature-vectors. Nodes test features, there is one branch for each value of the feature, and leaves specify the category.

- Can represent arbitrary conjunction and disjunction. Can represent any classification function over discrete feature vectors.

- Can be rewritten as a set of rules, i.e. disjunctive normal form (DNF).
  - \( \text{red} \land \text{circle} \rightarrow \text{pos} \)
  - \( \text{red} \land \text{circle} \rightarrow \text{A} \)
    - \( \text{blue} \rightarrow \text{B}; \ \text{red} \land \text{square} \rightarrow \text{B} \)
    - \( \text{green} \rightarrow \text{C}; \ \text{red} \land \text{triangle} \rightarrow \text{C} \)
Properties of Decision Tree Learning

- Continuous (real-valued) features can be handled by allowing nodes to split a real valued feature into two ranges based on a threshold (e.g. length < 3 and length $\geq$3).
- Classification trees have discrete class labels at the leaves, *regression trees* allow real-valued outputs at the leaves.
- Algorithms for finding consistent trees are efficient for processing large amounts of training data for data mining tasks.
- Methods developed for handling noisy training data (both class and feature noise).
- Methods developed for handling missing feature values.
Top-Down Decision Tree Induction

- Recursively build a tree top-down by divide and conquer.

![Decision Tree Diagram]

- <big, red, circle>: +
- <small, red, circle>: +
- <small, red, square>: –
- <big, blue, circle>: –
- color
- red
- blue
- green
- <big, red, circle>: +
- <small, red, circle>: +
- <small, red, square>: –
Top-Down Decision Tree Induction

• Recursively build a tree top-down by divide and conquer.

<big, red, circle>: +  <small, red, circle>: +
<small, red, square>: –  <big, blue, circle>: –

<big, red, circle>: +
<small, red, circle>: +
<small, red, square>: –
<big, blue, circle>: –

<big, red, circle>: +  <small, red, circle>: +
<small, red, square>: –
Decision Tree Induction Pseudocode

DTree($examples, features$) returns a tree
   If all $examples$ are in one category, return a leaf node with that category label.
   Else if the set of $features$ is empty, return a leaf node with the category label that is the most common in $examples$.
   Else pick a feature $F$ and create a node $R$ for it
      For each possible value $v_i$ of $F$:
         Let $examples_i$ be the subset of $examples$ that have value $v_i$ for $F$
         Add an out-going edge $E$ to node $R$ labeled with the value $v_i$.
         If $examples_i$ is empty
            then attach a leaf node to edge $E$ labeled with the category that is the most common in $examples$.
            else call DTree($examples_i, features - \{F\}$) and attach the resulting tree as the subtree under edge $E$.
      Return the subtree rooted at $R$. 
Picking a Good Split Feature

• Goal is to have the resulting tree be as small as possible, per Occam’s razor.
• Finding a minimal decision tree (nodes, leaves, or depth) is an NP-hard optimization problem.
• Top-down divide-and-conquer method does a greedy search for a simple tree but does not guarantee to find the smallest.
  – General lesson in ML: “Greed is good.”
• Want to pick a feature that creates subsets of examples that are relatively “pure” in a single class so they are “closer” to being leaf nodes.
• There are a variety of heuristics for picking a good test, a popular one is based on information gain that originated with the ID3 system of Quinlan (1979).
Simple Decision Trees

- What is our definition of simplicity? Fewest tests?
  - Preference for simplicity: Occam’s razor
  - Most likely hypothesis is the simplest one that is consistent with all observations
  - Captures some inherent structure in the problem?

- Allows us to generalize past seen examples:
  - E.g., Temp = hot, Outlook = rain, windy = false, humid = high
- How do we find the simplest decision tree?
- Exhaustive search over all possible decision trees?
Terminology

- Low entropy = Low uncertainty = Low information requirement for getting a correct answer about classifying an object

- High entropy = High uncertainty = High information requirement for getting a correct answer about classifying an object
Entropy

• Entropy (disorder, impurity) of a set of examples, S, relative to a binary classification is:

\[ \text{Entropy}(S) = -p_1 \log_2(p_1) - p_0 \log_2(p_0) \]

where \( p_1 \) is the fraction of positive examples in S and \( p_0 \) is the fraction of negatives.

• If all examples are in one category, entropy is zero (we define 0·log(0)=0)

• If examples are equally mixed (\( p_1=p_0=0.5 \)), entropy is a maximum of 1.

• Entropy can be viewed as the number of bits required on average to encode the class of an example in S where data compression (e.g. Huffman coding) is used to give shorter codes to more likely cases.

• For multi-class problems with \( c \) categories, entropy generalizes to:

\[ \text{Entropy}(S) = \sum_{i=1}^{c} - p_i \log_2(p_i) \]
Entropy Plot for Binary Classification
Information Gain

- The information gain of a feature $F$ is the expected reduction in entropy resulting from splitting on this feature.

$$Gain(S, F) = Entropy(S) - \sum_{v \in Values(F)} \frac{|S_v|}{|S|} Entropy(S_v)$$

where $S_v$ is the subset of $S$ having value $v$ for feature $F$.
- Entropy of each resulting subset weighted by its relative size.
- Example:
  - <big, red, circle>: +
  - <small, red, circle>: +
  - <small, red, square>: -
  - <big, blue, circle>: -

<table>
<thead>
<tr>
<th>Values</th>
<th>Entropy</th>
<th>Gain</th>
</tr>
</thead>
<tbody>
<tr>
<td>size</td>
<td></td>
<td></td>
</tr>
<tr>
<td>big</td>
<td>1+</td>
<td>1+</td>
</tr>
<tr>
<td>small</td>
<td>1+</td>
<td>1+</td>
</tr>
<tr>
<td>E=1</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Gain=1-(0.5\cdot1+0.5\cdot1) = 0</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

| size     |         |      |
| big      | 1+      | 1+   |
| small    | 1+      | 1+   |
| E=1      |         |      |
| Gain=1-(0.5\cdot1+0.5\cdot1) = 0 |

| color    |         |      |
| red      | 2+      | 0+   |
| blue     | 1+      | 1-   |
| E=0.918  |         |      |
| Gain=1-(0.75\cdot0.918 + 0.25\cdot0) = 0.311 |

| color    |         |      |
| red      | 2+      | 0+   |
| blue     | 1+      | 1-   |
| E=0.918  |         |      |
| Gain=1-(0.75\cdot0.918 + 0.25\cdot0) = 0.311 |

| shape    |         |      |
| circle   | 2+      | 0+   |
| square   | 1-      | 1-   |
| E=0.918  |         |      |
| Gain=1-(0.75\cdot0.918 + 0.25\cdot0) = 0.311 |
Hypothesis Space Search

- Performs *batch learning* that processes all training instances at once rather than *incremental learning* that updates a hypothesis after each example.
- Performs hill-climbing (greedy search) that may only find a locally-optimal solution. Guaranteed to find a tree consistent with any conflict-free training set (i.e. identical feature vectors always assigned the same class), but not necessarily the simplest tree.
- Finds a single discrete hypothesis, so there is no way to provide confidences or create useful queries.
Bias in Decision-Tree Induction

- Information-gain gives a bias for trees with minimal depth.
- Implements a search (preference) bias instead of a language (restriction) bias.
Computational Complexity

- Worst case builds a complete tree where every path test every feature. Assume $n$ examples and $m$ features.

  Maximum of $n$ examples spread across all nodes at each of the $m$ levels

- At each level, $i$, in the tree, must examine the remaining $m - i$ features for each instance at the level to calculate info gains.

  $\sum_{i=1}^{m} i \cdot n = O(nm^2)$

- However, learned tree is rarely complete (number of leaves is $\leq n$). In practice, complexity is linear in both number of features ($m$) and number of training examples ($n$).
Overfitting

- Learning a tree that classifies the training data perfectly may not lead to the tree with the best generalization to unseen data.
  - There may be noise in the training data that the tree is erroneously fitting.
  - The algorithm may be making poor decisions towards the leaves of the tree that are based on very little data and may not reflect reliable trends.
- A hypothesis, $h$, is said to overfit the training data is there exists another hypothesis which, $h'$, such that $h$ has less error than $h'$ on the training data but greater error on independent test data.

![Graph showing the relationship between hypothesis accuracy and complexity, with one line indicating performance on training data and another showing performance on test data.](image-url)
Overfitting Example

Testing Ohms Law: \( V = IR \)  \( (I = (1/R)V) \)

Experimentally measure 10 points

Fit a curve to the Resulting data.

Perfect fit to training data with an 9\textsuperscript{th} degree polynomial (can fit \( n \) points exactly with an \( n-1 \) degree polynomial)
Overfitting Example

Testing Ohms Law: \( V = IR \) \( (I = (1/R)V) \)

Better generalization with a linear function that fits training data less accurately.
Category or feature noise can easily cause overfitting.

- Add noisy instance <medium, blue, circle>: pos (but really neg)
Overfitting Noise in Decision Trees

- Category or feature noise can easily cause overfitting.
  - Add noisy instance <medium, blue, circle>: pos (but really neg)

- Noise can also cause different instances of the same feature vector to have different classes. Impossible to fit this data and must label leaf with the majority class.
  - <big, red, circle>: neg (but really pos)

- Conflicting examples can also arise if the features are incomplete and inadequate to determine the class or if the target concept is non-deterministic.
Overfitting Prevention (Pruning) Methods

- Two basic approaches for decision trees
  - **Prepruning**: Stop growing tree as some point during top-down construction when there is no longer sufficient data to make reliable decisions.
  - **Postpruning**: Grow the full tree, then remove subtrees that do not have sufficient evidence.

- Label leaf resulting from pruning with the majority class of the remaining data, or a class probability distribution.

- Method for determining which subtrees to prune:
  - **Cross-validation**: Reserve some training data as a hold-out set (*validation set, tuning set*) to evaluate utility of subtrees.
  - **Statistical test**: Use a statistical test on the training data to determine if any observed regularity can be dismissed as likely due to random chance.
  - **Minimum description length (MDL)**: Determine if the additional complexity of the hypothesis is less complex than just explicitly remembering any exceptions resulting from pruning.