CSEP 546: Data Mining
Instructor: Pedro Domingos

Today’s Agenda
- Inductive learning
- Decision trees

Inductive Learning

Supervised Learning
- Given: Training examples \( x, f(x) \) for some unknown function \( f \).
- Find: A good approximation to \( f \).

Example Applications
- Credit risk assessment
  \( f(x) \): Probability of customer and proposed purchase.
- Disease diagnosis
  \( f(x) \): Disease (or maybe, recommended therapy).
- Face recognition
  \( f(x) \): Bitswap picture of person’s face.
  \( f(x) \): Name of the person.
- Automatic steering
  \( f(x) \): Bitswap picture of road surface in front of car.
  \( f(x) \): Degrees to turn the steering wheel.

Appropriate Applications for Supervised Learning
- Situations where there is no human expert
  \( f(x) \): Predicted binding strength to AIDS protease inhibitors.
- Situations where humans can perform the task but can’t describe how they do it.
  \( f(x) \): Bitswap picture of head within character.
- Situations where the desired function is changing frequently
  \( f(x) \): Description of stock prices and trends for last 30 days.
- Situations where each user needs a customized function \( f \)
  \( f(x) \): Importance score for presenting to user (or deleting without presenting).

A Learning Problem

Example: \( x_1, x_2, x_3, x_4, y \)

<table>
<thead>
<tr>
<th>( x_1 )</th>
<th>( x_2 )</th>
<th>( x_3 )</th>
<th>( x_4 )</th>
<th>( y )</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>2</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>3</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>4</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>5</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>6</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>7</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>0</td>
</tr>
</tbody>
</table>
Hypothesis Spaces

- **Complete Ignorance.** There are $2^n$ possible boolean functions for $n$ input features. We can't figure out which one is correct until we've seen every possible input-output pair. After $n$ examples, we still have $2^n$ possibilities.

<table>
<thead>
<tr>
<th>Rule</th>
<th>Counterexample</th>
</tr>
</thead>
<tbody>
<tr>
<td>$x_1 \land \ldots \land x_n$</td>
<td>$y$</td>
</tr>
<tr>
<td>$x_1 \lor \ldots \lor x_n$</td>
<td>$y$</td>
</tr>
<tr>
<td>$x_i$</td>
<td>$y$</td>
</tr>
<tr>
<td>$\neg x_i$</td>
<td>$y$</td>
</tr>
<tr>
<td>$x_i \land \neg x_j$</td>
<td>$y$</td>
</tr>
<tr>
<td>$x_i \lor \neg x_j$</td>
<td>$y$</td>
</tr>
<tr>
<td>$x_i \land \ldots \land x_k$</td>
<td>$y$</td>
</tr>
<tr>
<td>$x_i \lor \ldots \lor x_k$</td>
<td>$y$</td>
</tr>
<tr>
<td>$x_i \land \ldots \land \neg x_k$</td>
<td>$y$</td>
</tr>
<tr>
<td>$x_i \lor \ldots \lor \neg x_k$</td>
<td>$y$</td>
</tr>
<tr>
<td>$\neg x_i \land \ldots \land x_k$</td>
<td>$y$</td>
</tr>
<tr>
<td>$\neg x_i \lor \ldots \lor x_k$</td>
<td>$y$</td>
</tr>
<tr>
<td>$x_i \land \ldots \land \neg x_k$</td>
<td>$y$</td>
</tr>
<tr>
<td>$x_i \lor \ldots \lor \neg x_k$</td>
<td>$y$</td>
</tr>
</tbody>
</table>

No simple rule explains the data. The same is true for simple clauses.

Hypothesis Space (3)

- **m-of-n rules.** There are 32 possible rule (include simple conjunction and clauses).

<table>
<thead>
<tr>
<th>Counterexample</th>
<th>Rule</th>
<th>&quot;And&quot;</th>
<th>&quot;Or&quot;</th>
</tr>
</thead>
<tbody>
<tr>
<td>$x_1 \land \ldots \land x_n$</td>
<td>$y$</td>
<td>$y$</td>
<td>$y$</td>
</tr>
<tr>
<td>$x_1 \lor \ldots \lor x_n$</td>
<td>$y$</td>
<td>$y$</td>
<td>$y$</td>
</tr>
<tr>
<td>$x_i$</td>
<td>$y$</td>
<td>$y$</td>
<td>$y$</td>
</tr>
<tr>
<td>$\neg x_i$</td>
<td>$y$</td>
<td>$y$</td>
<td>$y$</td>
</tr>
<tr>
<td>$x_i \land \neg x_j$</td>
<td>$y$</td>
<td>$y$</td>
<td>$y$</td>
</tr>
<tr>
<td>$x_i \lor \neg x_j$</td>
<td>$y$</td>
<td>$y$</td>
<td>$y$</td>
</tr>
<tr>
<td>$x_i \land \ldots \land x_k$</td>
<td>$y$</td>
<td>$y$</td>
<td>$y$</td>
</tr>
<tr>
<td>$x_i \lor \ldots \lor x_k$</td>
<td>$y$</td>
<td>$y$</td>
<td>$y$</td>
</tr>
<tr>
<td>$x_i \land \ldots \land \neg x_k$</td>
<td>$y$</td>
<td></td>
<td></td>
</tr>
<tr>
<td>$x_i \lor \ldots \lor \neg x_k$</td>
<td>$y$</td>
<td></td>
<td></td>
</tr>
<tr>
<td>$\neg x_i \land \ldots \land x_k$</td>
<td>$y$</td>
<td></td>
<td></td>
</tr>
<tr>
<td>$\neg x_i \lor \ldots \lor x_k$</td>
<td>$y$</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Two Views of Learning

- **Learning is the removal of our remaining uncertainty.** Suppose we know that the unknown function was an m-of-n boolean function, then we could use the training examples to infer which function it is.
- **Learning requires guarding a good, small hypothesis class.** We can start with a very small class and enlarge it until it contains an hypothesis that fits the data.

We could be wrong!

- Our prior knowledge might be wrong
- Our guess of the hypothesis class could be wrong

Example: $x_1 \land x_2 \land x_3 \Rightarrow y$ is also consistent with the training data.
Example: $x_1 \land \neg x_2 \lor x_3$ is also consistent with the training data.

If either of these is the unknown function, then we will make errors when we are given new a value.

Two Strategies for Machine Learning

- **Develop Languages for Expressing Prior Knowledge.** Rule grammars and stochastic models.
- **Develop Flexible Hypothesis Spaces.** Nested collections of hypotheses.
- Decision trees, rules, neural networks, cases.

In either case:

- Develop Algorithms to Find an Hypothesis that Fits the Data

Terminology

- **Training example.** An example of the form $(x^t, f(x^t))$.
- **Target function (target concept).** The true function $f$.
- **Hypothesis.** A proposed function $h$ is believed to be similar to $f$.
- **Concept.** A boolean function. Examples for which $f(x) = 1$ are called positive examples or positive instances of the concept. Examples for which $f(x) = 0$ are called negative examples or negative instances.
- **Classifier.** A discrete-valued function. The possible values $f(x) \in \{1, \ldots, K\}$ are called the classes or class labels.
- **Hypothesis Space.** The space of all hypotheses that can, in principle, be output by a learning algorithm.
- **Version Space.** The space of all hypotheses in the hypothesis space that have not yet been ruled out by a training example.
Key Issues in Machine Learning

- What are good hypothesis spaces?
- Which spaces have been useful in practical applications and why?
- What algorithms can work with these spaces?
  Are there general design principles for machine learning algorithms?
- How can we optimize accuracy on future data points?
  This is sometimes called the “problem of overfitting”.
- How can we have confidence in the results?
  How much training data is required to find accurate hypotheses? (the statistical problem)
- Are some learning problems computationally intractable?
  (the computational problem)
- How can we formulate application problems as machine learning problems?
  (the engineering problem)

A Framework for Hypothesis Spaces

- Size: Does the hypothesis space have a fixed size or variable size?
  Fixed-size spaces are easier to understand, but variable-size spaces are generally more useful. Variable-size spaces introduce the problem of overfitting.
- Randomness: Is each hypothesis deterministic or stochastic?
  This affects how we evaluate hypotheses. With a deterministic hypothesis, a training example is either classified (correctly predicted) or incorrectly classified (incorrectly predicted). With a stochastic hypothesis, a training example is more likely to be misclassified.
- Parameterization: Is each hypothesis described by a set of symbols (discrete) choices or is it described by a set of continuous parameters? If both are required, we say the hypothesis space has a mixed parameterization.
  Discrete parameters must be found by combinatorial search methods; continuous parameters can be found by numerical search methods.

A Framework for Learning Algorithms

- Search Procedure:
  - Directional Search: solve for the hypothesis directly.
  - Local Search: start with an initial hypothesis, make small improvements until a local optimum.
  - Constructive Search: start with an empty hypothesis, gradually add structure to it until a local optimum.
- Timing:
  - Eager: Analyze the training data and construct an explicit hypothesis.
  - Lazy: Store the training data and wait until a test data point is presented, then construct an ad hoc hypothesis to classify that one data point.
- Online vs. Batch (for eager algorithms)
  - Online: Analyze each training example as it is presented.
  - Batch: Collect training examples, analyze them, output an hypothesis.

Decision Trees

- Learning Decision Trees
  - Decision trees provide a very popular and efficient hypothesis space.
  - Variable Size: Any boolean function can be represented.
  - Deterministic
  - Discrete and Continuous Parameters

Decision Tree Hypothesis Space

- Internal nodes test the value of particular features $x_j$ and branch according to the results of this test.
- Leaf nodes specify the class $h(x)$:

```
          Outlook
            /   \
           /     \
    Overcast  Rain
            /   \  /   \  
       Yes    No   No    Yes
```

Suppose the features are Outlook ($x_1$), Temperature ($x_2$), Humidity ($x_3$), and Wind ($x_4$). Then the feature vector $x = [\text{Sunny}, \text{Hot}, \text{High}, \text{Strong}]$ will be classified as No. The Temperature feature is irrelevant.
Decision Tree Hypothesis Space

If the features are continuous, internal nodes may test the value of a feature against a threshold.

Decision Tree Decision Boundaries

Decision trees divide the feature space into axis-parallel rectangles, and label each rectangle with one of the $K$ classes.

Decision Trees Can Represent Any Boolean Function

The tree will in the worst case require exponentially many nodes, however.

Decision Trees Provide Variable-Size Hypothesis Space

As the number of nodes (or depth) of tree increases, the hypothesis space grows:

- depth 1 ("decision stump") can represent any boolean function of one feature.
- depth 2 Any boolean function of two features (or boolean functions involving three features, e.g., $A \land B \lor \neg C$)
- etc.

Learning Algorithm for Decision Trees

The same basic learning algorithm has been discovered by many people independently.

GROWTREE(S)

if $(y = 0 \text{ for all } (x, y) \in S) \text{ return new leaf}(0)$
else if $(y = 1 \text{ for all } (x, y) \in S) \text{ return new leaf}(1)$
else choose best attribute $A_c$

$S_k = \{ (x, y) \in S \mid y = 0 \}$
$S_l = \{ (x, y) \in S \mid y = 1 \}$
return new node($A_c, \text{GROWTREE}(S_k), \text{GROWTREE}(S_l))$

Choosing the Best Attribute

One way to choose the best attribute is to perform a 1-step exhaustive search and choose the attribute that gives the lowest error rate on the training data.

CHOOSEBESTATTRIBUTE(S)

choose $j$ to minimize $J_j$, computed as follows:

$S_0 = \{ (x, y) \in S \mid y = 0 \}$
$S_1 = \{ (x, y) \in S \mid y = 1 \}$
$y_0 = \text{most common value of } y \text{ in } S_0$
$y_1 = \text{most common value of } y \text{ in } S_1$
$J_0 = \text{number of examples } (x, y) \in S_0 \text{ with } y \neq y_0$
$J_1 = \text{number of examples } (x, y) \in S_1 \text{ with } y \neq y_1$
$J = J_0 + J_1 \text{ (total error if we split on this feature)}$
return $j$
Choosing the Best Attribute—An Example

Choosing the Best Attribute (3)

A Better Heuristic From Information Theory

Let $V$ be a random variable with the following probability distribution:

\[
P(V = 0) = 0.2, \quad P(V = 1) = 0.8
\]

The surprise, $S(V = v)$, of each value of $V$ is defined to be

\[
S(V = v) = -\log P(V = v).
\]

An event with probability 1 gives an zero surprise.

An event with probability 0 gives an infinite surprise!

It turns out that the surprise is equal to the number of bits of information that need to be transmitted to a recipient who knows the probabilities of the result. This is also called the description length of $V = v$.

Functional bits only make sense if they are part of a longer message (e.g., describe a whole sequence of mint tosses).

Entropy

The entropy of $V$, denoted $H(V)$, is defined as follows:

\[
H(V) = \sum_{v} -P(V = v) \log P(V = v).
\]

This is the average surprise of describing the result of our “trials” of $V$ (e.g., coin toss).

Entropy can be viewed as a measure of uncertainty.

Mutual Information

Now consider two random variables $A$ and $B$ that are not necessarily independent. The mutual information between $A$ and $B$ is the amount of information we learn about $B$ by knowing the value of $A$ (and vice versa—it is symmetric). It is computed as follows:

\[
I(A; B) = H(B) - \sum_{b} P(B = b) \cdot H(B = b)
\]

In particular, consider the class $Y$ of each training example and the value of feature $x_1$ to be random variables. Then the mutual information quantifies how much $x_1$ tells us about the value of the class $Y$.

\[
I(Y; x_1) = 0.5778
\]

Visualising Heuristics

Visualising Heuristics
Non-Boolean Features

- Features with multiple discrete values
  - Construct a binary split?
  - "Test for one value versus all of the others?"
  - What if there are two disjoint subsets?
- Real-valued features
  - Consider a threshold split using each observed value of the feature.
- Whichever method is used, the mutual information can be computed to choose the best split.

Attributes with Many Values

Problem:
- If attribute has many values, Gain will select it
- Imagine using \( \text{Date} = \text{Jan.3,1996} \) as attribute

One approach: use \( \text{GainRatio} \) instead

\[
\text{GainRatio}(S,A) = \frac{\text{Gain}(S,A)}{\text{SplitInformation}(S,A)}
\]

\[
\text{SplitInformation}(S,A) = - \sum_{v \in S} \frac{|S_v|}{|S|} \log_2 \frac{|S_v|}{|S|}
\]

where \( S_v \) is subset of \( S \) for which \( A \) has value \( v \)

Learning Parity with Noise

When learning non-linear or 3-bit parity, all splits look equally good. If extra random boolean features are included, they also look equally good. Hence, decision tree algorithms cannot distinguish random noisy features from parity features.

\[
\begin{align*}
(4,4) & \\
\text{Yes} & \text{Yes} & \text{Yes}
\end{align*}
\]

Unknown Attribute Values

What if some examples are missing values of \( A \)?

Use training example anyway, sort through tree
- If node \( n \) tests \( A \), assign most common value of \( A \) among other examples sorted to node \( n \)
- Assign most common value of \( A \) among other examples with same target value
- Assign probability \( p_i \) to each possible value \( v_i \) of \( A \)
  - Assign fraction \( p_i \) of example to each descendant in tree
  - Classify new examples in same fashion

Overfitting in Decision Trees

Consider adding a noisy training example:
- Sunny, Hot, Normal, Strong, PlayTennis=No
  - What effect on tree?

Overfitting

Consider error of hypothesis \( h \) over
- training data: \( \text{error}_{\text{train}}(h) \)
- entire distribution \( \mathcal{D} \) of data: \( \text{error}_\mathcal{D}(h) \)

Hypothesis \( h \in H \) overfits training data if there is an alternative hypothesis \( h' \in H \) such that

\[
\text{error}_{\text{train}}(h) < \text{error}_{\text{train}}(h')
\]

and

\[
\text{error}_\mathcal{D}(h) > \text{error}_\mathcal{D}(h')
\]
Overfitting in Decision Tree Learning

Avoiding Overfitting

How can we avoid overfitting?
- Stop growing when data split not statistically significant
- Grow full tree, then post-prune

How to select “best” tree:
- Measure performance over training data
- Measure performance over separate validation data set
- Add complexity penalty to performance measure

Reduced-Error Pruning

Split data into training and validation set

Do until further pruning is harmful:
1. Evaluate impact on validation set of pruning each possible node (plus those below it)
2. Greedily remove the one that most improves validation set accuracy

Effect of Reduced-Error Pruning

Rule Post-Pruning

1. Convert tree to equivalent set of rules
2. Prune each rule independently of others
3. Sort final rules into desired sequence for use

Perhaps most frequently used method (e.g., C4.5)

Converting A Tree to Rules
Summary

• Inductive learning
• Decision trees
  – Representation
  – Tree growth
  – Heuristics
  – Overfitting and pruning
  – Scaling up

Scaling Up

• ID3, C4.5, etc. assume data fits in main memory
  (OK for up to hundreds of thousands of examples)
• SPRINT, SLIQ: multiple sequential scans of data
  (OK for up to millions of examples)
• VFDT: at most one sequential scan
  (OK for up to billions of examples)