Learning

Learning is the ability to improve one’s behavior based on experience.
- The range of behaviors is expanded: the agent can do more.
- The accuracy on tasks is improved: the agent can do things better.
- The speed is improved: the agent can do things faster.

Components of a learning problem

The following components are part of any learning problem:
- **task**: The behavior or task that’s being improved. For example: classification, acting in an environment
- **data**: The experiences that are being used to improve performance in the task.
- **measure of improvement**: How can the improvement be measured? For example: increasing accuracy in prediction, new skills that were not present initially, improved speed.

Types of learning

- Draw a conclusion from a knowledge base: **deduction** (top-down)
- Infer a representation from data: **induction** (bottom-up)
- The richer (more complex) the representation, the more useful it is for subsequent problem solving.
- The richer the representation, the more difficult it is to learn.

Common Learning Tasks

- **Supervised classification**: Given a set of pre-classified training examples, classify a new instance.
- **Unsupervised learning**: Find natural classes for examples.
- **Reinforcement learning**: Determine what to do based on rewards and punishments.
- **Transfer Learning**: Learning from an expert
- **Active Learning**: Learner actively seeks to learn
- **Inductive logic programming**: Build richer models in terms of logic programs.

Feedback

Learning tasks can be characterized by the feedback given to the learner.
- **Supervised learning**: What has to be learned is specified for each example.
- **Unsupervised learning**: No classifications are given; the learner has to discover categories and regularities in the data.
- **Reinforcement learning**: Feedback occurs after a sequence of actions. Credit assignment problem. Is a form of Supervised Learning.
Measuring Success

- The measure of success is not how well the agent performs on the training examples, but how well the agent performs for new examples.
- Consider two agents solving a binary classification task:
  - $P$ claims the negative examples seen are the only negative examples. Every other instance is positive.
  - $N$ claims the positive examples seen are the only positive examples. Every other instance is negative.
- Both agents correctly classify every training example, but disagree on every other example.

Bias

- The tendency to prefer one hypothesis over another is called a bias.
- A bias is necessary to make predictions on unseen data.
- Saying a hypothesis is better than $N$’s or $P$’s hypothesis isn’t something that’s obtained from the data.
- To have any inductive process make predictions on unseen data, you need a bias.
- What constitutes a good bias is an empirical question about which biases work best in practice.

Learning as search

- Given a representation and a bias, the problem of learning can be reduced to one of search.
- Learning is search through the space of possible representations looking for the representation or representations that best fits the data, given the bias.
- These search spaces are typically prohibitively large for systematic search.
- A learning algorithm is made of a search space, an evaluation function, and a search method.

Supervised Learning

- Given:
  - a set of input features $X_1, \ldots, X_n$
  - a set of target features $Y_1, \ldots, Y_k$
  - a set of training examples where the values for the input features and the target features are given for each example
  - a set of test examples, where only the values for the input features are given
- predict the values for the target features for the test examples.
  - classification when the $Y_i$ are discrete
  - regression when the $Y_i$ are continuous
- Very important: keep training and test sets separate!

Noise

- Data isn’t perfect:
  - some of the features are assigned the wrong value
  - the features given are inadequate to predict the classification
  - there are examples with missing features
- overfitting occurs when a distinction appears in the data, but doesn’t appear in the unseen examples. This occurs because of random correlations in the training set.

Evaluating Predictions

- Suppose $Y$ is a feature and $e$ is an example:
  - $Y(e)$ is the value of feature $Y$ for example $e$.
  - $\hat{Y}(e)$ is the predicted value of feature $Y$ for example $e$.
  - The error of the prediction is a measure of how close $\hat{Y}(e)$ is to $Y(e)$.
  - There are many possible errors that could be measured.
Measures of error

\[ E \] is the set of examples. \( T \) is the set of target features.

- **absolute error**
  \[ \sum_{e \in E} \sum_{Y \in T} \left| Y(e) - \hat{Y}(e) \right| \]

- **sum of squares error**
  \[ \sum_{e \in E} \sum_{Y \in T} (Y(e) - \hat{Y}(e))^2 \]

- **worst-case error**:
  \[ \max_{e \in E} \max_{Y \in T} \left| Y(e) - \hat{Y}(e) \right| \]

A **cost-based error** takes into account costs of various errors.

Measures of error (cont.)

When target features are \( Y(e) \in \{0, 1\} \) and predicted features are \( \hat{Y}(e) \in [0, 1] \) (predicted features: probability the target is 1):

- **likelihood of the data** (maximize this)
  \[
  \prod_{e \in E} \prod_{Y \in T} P(\hat{Y}(e) \mid Y(e)) \prod_{e \in E} \prod_{Y \in T} \hat{Y}(e)^{Y(e)} (1 - \hat{Y}(e))^{(1 - Y(e))}
  \]

- **entropy** or **negative log likelihood** (minimize this: a cost)
  \[
  - \sum_{e \in E} \sum_{Y \in T} [Y(e) \log \hat{Y}(e) - (1 - Y(e)) \log(1 - \hat{Y}(e))]
  \]
Not all errors are equal, e.g. predict:
- a patient has a disease when they do not
- a patient doesn’t have a disease when they do

need to map out both kinds of errors to find the best trade-off

<table>
<thead>
<tr>
<th>actual</th>
<th>T</th>
<th>F</th>
</tr>
</thead>
<tbody>
<tr>
<td>T</td>
<td>true positive (TP)</td>
<td>false negative (FN)</td>
</tr>
<tr>
<td>F</td>
<td>false positive (FP)</td>
<td>true negative (TN)</td>
</tr>
</tbody>
</table>

recall = sensitivity = TP/(TP+FN)
specificity = TN/(TN+FP)
precision = TP/(TP+FP)
F-measure = 2*Precision*Recall/(Precision+Recall)

Many learning algorithms can be seen as deriving from:
- decision trees
- linear classifiers
- Bayesian classifiers

Example: user discussion board behaviors

<table>
<thead>
<tr>
<th>example</th>
<th>author</th>
<th>thread</th>
<th>length</th>
<th>where read</th>
<th>user's action</th>
</tr>
</thead>
<tbody>
<tr>
<td>e1</td>
<td>known</td>
<td>new</td>
<td>long</td>
<td>home</td>
<td>skips</td>
</tr>
<tr>
<td>e2</td>
<td>unknown</td>
<td>new</td>
<td>short</td>
<td>work</td>
<td>reads</td>
</tr>
<tr>
<td>e3</td>
<td>unknown</td>
<td>new</td>
<td>short</td>
<td>follow up</td>
<td>skips</td>
</tr>
<tr>
<td>e4</td>
<td>known</td>
<td>follow up</td>
<td>long</td>
<td>work</td>
<td>skips</td>
</tr>
<tr>
<td>e5</td>
<td>known</td>
<td>new</td>
<td>short</td>
<td>home</td>
<td>reads</td>
</tr>
<tr>
<td>e6</td>
<td>known</td>
<td>follow up</td>
<td>long</td>
<td>work</td>
<td>skips</td>
</tr>
<tr>
<td>e7</td>
<td>unknown</td>
<td>follow up</td>
<td>short</td>
<td>work</td>
<td>skips</td>
</tr>
<tr>
<td>e8</td>
<td>unknown</td>
<td>new</td>
<td>short</td>
<td>work</td>
<td>skips</td>
</tr>
<tr>
<td>e9</td>
<td>known</td>
<td>new</td>
<td>short</td>
<td>follow up</td>
<td>reads</td>
</tr>
<tr>
<td>e10</td>
<td>known</td>
<td>new</td>
<td>long</td>
<td>work</td>
<td>skips</td>
</tr>
<tr>
<td>e11</td>
<td>unknown</td>
<td>new</td>
<td>short</td>
<td>follow up</td>
<td>skips</td>
</tr>
<tr>
<td>e12</td>
<td>known</td>
<td>new</td>
<td>long</td>
<td>work</td>
<td>reads</td>
</tr>
<tr>
<td>e13</td>
<td>known</td>
<td>new</td>
<td>short</td>
<td>follow up</td>
<td>reads</td>
</tr>
<tr>
<td>e14</td>
<td>known</td>
<td>new</td>
<td>short</td>
<td>work</td>
<td>reads</td>
</tr>
<tr>
<td>e15</td>
<td>known</td>
<td>new</td>
<td>short</td>
<td>home</td>
<td>reads</td>
</tr>
<tr>
<td>e16</td>
<td>known</td>
<td>new</td>
<td>short</td>
<td>follow up</td>
<td>reads</td>
</tr>
<tr>
<td>e17</td>
<td>known</td>
<td>new</td>
<td>short</td>
<td>home</td>
<td>reads</td>
</tr>
<tr>
<td>e18</td>
<td>unknown</td>
<td>new</td>
<td>short</td>
<td>work</td>
<td>reads</td>
</tr>
<tr>
<td>e19</td>
<td>unknown</td>
<td>new</td>
<td>long</td>
<td>work</td>
<td>?</td>
</tr>
</tbody>
</table>

Learning Decision Trees
- Representation is a decision tree.
- Bias is towards simple decision trees.
- Search through the space of decision trees, from simple decision trees to more complex ones.
Simple, successful technique for supervised learning from discrete data
Learn a decision tree from data:
- Nodes are input attributes/features
- Branches are labeled with input feature value(s)
- Leaves are predictions for target features (point estimates)
- Can have many branches per node
- Branches can be labeled with multiple feature values

Learning a decision tree
- Incrementally split the training data
- Recursively solve sub-problems
- Hard part: how to split the data?
- Criteria for a good decision tree (bias):
  - small decision tree,
  - good classification (low error on training data),
  - good generalisation (low error on test data)

Decision tree learning: pseudocode
//X is input features, Y is output features,
//E is training examples
//output is a decision tree, which is either
// - a point estimate of Y, or
// - of the form <X_i, T_1, . . . , T_N> where
//  X_i is an input feature and T_1, . . . , T_N are decision trees

procedure DecisionTreeLearner(X,Y,E)
if stopping criteria is met then
  return pointEstimate(Y,E)
else
  select feature X_i \in X
  for each value x_i of X_i do
    E_i = all examples in E where X_i = x_i
    T_i = DecisionTreeLearner(X\{X_i}, Y, E_i)
  end for
  return <X_i, T_1, . . . , T_N>
end procedure

Decision tree classification: pseudocode
//X is is input features, Y is output features,
//e is test example
//DT is a decision tree
//output is a prediction of Y for e

procedure ClassifyExample(e,X,Y,DT)
S ← DT
while S is internal node of the form <X_i, T_1, . . . , T_N> do
  j ← X_i(e)
  S ← T_j
end while
return S
end procedure

Remaining issues
- Stopping criteria
- Selection of features
- Point estimate (final return value at leaf)
- Reducing number of branches (partition of domain)
Stopping Criteria

- How do we decide to stop splitting?
- The stopping criteria is related to the final return value
- Depends on what we will need to do
- Possible stopping criteria:
  - No more features
  - Performance on training data is good enough

Feature Selection

- Ideal: choose sequence of features that result in smallest tree
- Actual: myopically split - as if only allowed one split, which feature would give best performance?
- heuristics for best performing feature:
  - Most even split
  - Maximum information gain
  - GINI index
  - ... others domain dependent ...

Information Theory

- A bit is a binary digit: 0 or 1
- n bits can distinguish \(2^n\) items
- Can do better by taking probabilities into account

Example:

distinguish \(\{a, b, c, d\}\) with
\[P(a) = 0.5, P(b) = 0.25, P(c) = P(d) = 0.125\]

If we encode
\[
a:0 \quad b:01 \quad c:10 \quad d:11
\]
uses on average \(2\) bits

but if we encode
\[
a:0 \quad b:10 \quad c:110 \quad d:111
\]
uses on average
\[
P(a) \times 1 + P(b) \times 2 + P(c) \times 3 + P(d) \times 3 = 1.75\text{ bits}
\]
Information gain

Given a set $E$ of $N$ training examples, if the number of examples with output feature $Y = y_i$ is $N_i$, then

$$P(Y = y_i) = P(y_i) = \frac{N_i}{N}$$

(the point estimate)

Total information content for the set $E$ is:

$$I(E) = - \sum_{y_i \in Y} P(y_i) \log P(y_i)$$

So, after splitting $E$ up into $E_1$ and $E_2$ (size $N_1$, $N_2$) based on input attribute $X_i$, the information content

$$I(E_{split}) = \frac{N_1}{N} I(E_1) + \frac{N_2}{N} I(E_2)$$

and we want the $X_i$ that maximises the information gain:

$$I(E) - I(E_{split})$$

Final return value

- Point estimate of $Y$ (output features) over all examples
- Point estimate is just a prediction of target features
  - mean value,
  - median value,
  - most likely classification,
  - etc.

  e.g.

  $$P(Y = y_i) = \frac{N_i}{N}$$

  where
  - $N_i$ is the number of training samples at the leaf with $Y = Y_i$
  - $N$ is the total number of training samples at the leaf.

Example: user discussion board behaviors

table

<table>
<thead>
<tr>
<th>example</th>
<th>author</th>
<th>thread</th>
<th>length</th>
<th>where read</th>
<th>user's action</th>
</tr>
</thead>
<tbody>
<tr>
<td>e1</td>
<td>known</td>
<td>new</td>
<td>long</td>
<td>home</td>
<td>skips</td>
</tr>
<tr>
<td>e2</td>
<td>unknown</td>
<td>new</td>
<td>short</td>
<td>work</td>
<td>reads</td>
</tr>
<tr>
<td>e3</td>
<td>unknown</td>
<td>follow up</td>
<td>long</td>
<td>work</td>
<td>skips</td>
</tr>
<tr>
<td>e4</td>
<td>known</td>
<td>follow up</td>
<td>long</td>
<td>home</td>
<td>skips</td>
</tr>
<tr>
<td>e5</td>
<td>known</td>
<td>new</td>
<td>short</td>
<td>home</td>
<td>reads</td>
</tr>
<tr>
<td>e6</td>
<td>known</td>
<td>follow up</td>
<td>long</td>
<td>work</td>
<td>skips</td>
</tr>
<tr>
<td>e7</td>
<td>unknown</td>
<td>follow up</td>
<td>short</td>
<td>work</td>
<td>skips</td>
</tr>
<tr>
<td>e8</td>
<td>unknown</td>
<td>new</td>
<td>short</td>
<td>work</td>
<td>reads</td>
</tr>
<tr>
<td>e9</td>
<td>known</td>
<td>follow up</td>
<td>long</td>
<td>home</td>
<td>skips</td>
</tr>
<tr>
<td>e10</td>
<td>known</td>
<td>new</td>
<td>long</td>
<td>work</td>
<td>skips</td>
</tr>
<tr>
<td>e11</td>
<td>unknown</td>
<td>follow up</td>
<td>short</td>
<td>home</td>
<td>skips</td>
</tr>
<tr>
<td>e12</td>
<td>known</td>
<td>new</td>
<td>long</td>
<td>work</td>
<td>skips</td>
</tr>
<tr>
<td>e13</td>
<td>known</td>
<td>follow up</td>
<td>short</td>
<td>home</td>
<td>reads</td>
</tr>
<tr>
<td>e14</td>
<td>known</td>
<td>new</td>
<td>short</td>
<td>work</td>
<td>reads</td>
</tr>
<tr>
<td>e15</td>
<td>known</td>
<td>new</td>
<td>short</td>
<td>home</td>
<td>reads</td>
</tr>
<tr>
<td>e16</td>
<td>known</td>
<td>follow up</td>
<td>short</td>
<td>work</td>
<td>reads</td>
</tr>
<tr>
<td>e17</td>
<td>known</td>
<td>new</td>
<td>short</td>
<td>home</td>
<td>reads</td>
</tr>
<tr>
<td>e18</td>
<td>unknown</td>
<td>new</td>
<td>short</td>
<td>work</td>
<td>reads</td>
</tr>
<tr>
<td>e19</td>
<td>unknown</td>
<td>new</td>
<td>long</td>
<td>work</td>
<td>?</td>
</tr>
</tbody>
</table>

Using a Priority Queue to Learn the DT

- The “vanilla” version we saw grows all branches for a node
- But there might be some branches that are more worthwhile to expand
- Idea: sort the leaves using a priority queue ranked by how much information can be gained with the best feature at that leaf
- always expand the leaf at the top of the queue

Decision tree learning: pseudocode V2

procedure DecisionTreeLearner(X,Y,E)
Start with a single node (index 0) with
- whole data set $E_0 \equiv E$,
- ... point estimate,
- best next feature to split on (of all the remaining features), and
- information gain for that split

repeat until a stopping criteria is reached
- find the tree leaf (index $i$) with the highest information-Gain $\Delta I_i$:
  - $i$ is the next split to do.
- Split the data at that leaf ($E_i$) according to the Best-Feature $X_i$
  - two datasets $E_{i+}$ and $E_{i-}$
- Add 2 children to node $i$, one with $E_{i+}$ and one with $E_{i-}$
- for each new child: compute and store in the child nodes:
  - point estimate,
  - best next feature to split on (of all the remaining features), and
  - information gain for that split
- insert $\{T_i, E_i, X_i, \Delta I_i\}$ into $PQ$ according to $\Delta I_i$
end repeat
return $DT$
end procedure
Overfitting

Sometimes the decision tree is “too good” at classifying the training data, and will not generalise very well. This often occurs when there is not much data.

3 attributes: bad weather, burnt toast, train late

 training data:
A: true, true, true;
A: false, false, false;
A: true, false, false;
A: false, true, false;
A: true, false, true;
A: false, true, false;
A: true, true, true;
A: true, false, true;
A: false, true, false;
A: false, false, false;
...

Some methods to avoid overfitting
- Regularization: e.g. Prefer small decision trees over big ones, so add a ‘complexity’ penalty to the stopping criteria - stop early.
- Pseudocounts: add some data based on prior knowledge.
- Cross validation.

Test set errors caused by:
- bias: the error due to the algorithm finding an imperfect model.
  - representation bias: model is too simple
  - search bias: not enough search
- variance: the error due to lack of data.
- noise: the error due to the data depending on features not modeled or because the process generating the data is inherently stochastic.
- bias-variance trade-off:
  - Complicated model, not enough data (low bias, high variance)
  - Simple model, lots of data (high bias, low variance)

Cross Validation

Split training data into a training and a validation set.
Use the validation set as a “pretend” test set.
Optimise the decision maker to perform well on the validation set, not the training set.
Can do this multiple times with different validation sets.

Capacity of a model is its ability to fit a wide variety of functions.
Capacity is like the inverse of bias - a high capacity model has low bias and vice-versa.
Next:

- Supervised Learning II (Poole & Mackworth (2nd ed.) Chapter 7.3.2, 7.5-7.6)
- Uncertainty (Poole & Mackworth (2nd ed.) Chapter 8)