Learning
Learning is improvement of performance (time, accuracy).

In supervised learning, from training examples of input-output pairs, predict the output of a new input.

In unsupervised learning, examples do not have outputs. The most common task is clustering.

In semi-supervised learning, some examples have outputs. For example, in reinforcement learning, an input is a series of actions, and the output is intermittent feedback.
Assume the learner is given the following:

- a set of input features, $X_1, \ldots, X_n$;
- a target feature, $Y$;
- a set of *training examples*, each with values for the $X$’s and $Y$;
- a set of *test examples*, each with values only for the $X$’s.

The learner finds a *hypothesis* $h$ to predict the target from the inputs.

Usually, $h$ is restricted to be an element from a *hypothesis space*.

*Regression* is when the target is continuous.

*Classification* is when the target is discrete.
# Example of Examples

<table>
<thead>
<tr>
<th>No.</th>
<th>Input Features</th>
<th>Target</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Outlook</td>
<td>Temp</td>
</tr>
<tr>
<td>1</td>
<td>sunny</td>
<td>hot</td>
</tr>
<tr>
<td>2</td>
<td>sunny</td>
<td>hot</td>
</tr>
<tr>
<td>3</td>
<td>overcast</td>
<td>hot</td>
</tr>
<tr>
<td>4</td>
<td>rain</td>
<td>mild</td>
</tr>
<tr>
<td>5</td>
<td>rain</td>
<td>cool</td>
</tr>
<tr>
<td>6</td>
<td>rain</td>
<td>cool</td>
</tr>
<tr>
<td>7</td>
<td>overcast</td>
<td>cool</td>
</tr>
<tr>
<td>8</td>
<td>sunny</td>
<td>mild</td>
</tr>
<tr>
<td>9</td>
<td>sunny</td>
<td>cool</td>
</tr>
<tr>
<td>10</td>
<td>rain</td>
<td>mild</td>
</tr>
<tr>
<td>11</td>
<td>sunny</td>
<td>mild</td>
</tr>
<tr>
<td>12</td>
<td>overcast</td>
<td>mild</td>
</tr>
<tr>
<td>13</td>
<td>overcast</td>
<td>hot</td>
</tr>
<tr>
<td>14</td>
<td>rain</td>
<td>mild</td>
</tr>
</tbody>
</table>
Evaluating Predictions

- Let $y_e$ be the target value for example $e$.
- Let $\hat{y}_e$ be the predicted value.
- **Error** (or **loss**) measures how close $\hat{y}_e$ is to $y_e$.
- **Zero-One Error**: if $y_e \neq \hat{y}_e$, then 1, else 0
- **Absolute Error**: $\left| y_e - \hat{y}_e \right|$ 
- **Squared Error**: $(y_e - \hat{y}_e)^2$
- **Entropy**: $-(y_e \log \hat{y}_e + (1 - y_e) \log (1 - \hat{y}_e))$
  (assumes $y_e$ and $\hat{y}_e$ are probabilities.)
- and many variations.

- For classification, use $y_e \in \{0, 1\}$ or $\{-1, 1\}$.
- Secret of machine learning: update hypothesis to reduce error.
Decision trees are a representation for classification.

- Each nonleaf is labeled by a feature.
- Edges from nonleaf to children are labeled by feature values.
- Each leaf is labeled by a prediction.

Typical Algorithm: Construct the tree top-down.

- Find the “best” feature.
- Split examples based on feature’s values.
Example of a Decision Tree

Learning
Decision Trees
Definition
Example
Learning Trees
Selecting a Feature
Information Plot
Gain Plot
Selection I
Selection II
Other Choices
Special Cases
Iris Dataset
Naive Bayes
Linear Regression
and Classification
Algorithm for Learning Decision Trees

Procedure $DTLearner(X, Y, E)$

Input: $X$: set of input features, $X = \{X_1, \ldots, X_n\}$
$Y$: target feature
$E$: set of training examples

if stopping criterion is true then
    return a leaf labeled with prediction of $Y$

Select feature $X_i \in X$, with domain $V$

let $T = $ nonleaf node labeled $X_i$

for each $v \in V$
    let $E' = \{e \in E : X_i = v\}$
    let $T' = DTLearner(X, Y, E')$
    add edge from $T$ to $T'$ labeled $v$

return $T$
Selecting a Feature: Information Gain

- $p$ positive examples and $n$ negative examples
- The information contained is:
  \[
  I(p, n) = -\frac{p}{p+n} \log_2 \frac{p}{p+n} - \frac{n}{p+n} \log_2 \frac{n}{p+n}
  \]
- Feature $X_i$ has $v$ values, $p_j$ positive examples and $n_j$ negative examples when $X_i = v_j$
- The **Remainder** of $X_i$ is:
  \[
  \text{Remainder}(X_i) = \sum_{j=1}^{v} \frac{p_j + n_j}{p+n} I(p_j, n_j)
  \]
- The information gain of $X_i$ is:
  \[
  \text{Gain}(X_i) = I(p, n) - \text{Remainder}(X_i)
  \]
$p$ positive examples and $n$ negative examples

$I(p, n=100-p)$
$p_1$ positive and $n_1$ negative exs. when $X_i = v_1$

$p_2$ positive and $n_2$ negative exs. when $X_i = v_2$

$\text{gain}(p_1, n_1=50-p_1, p_2, n_2=50-p_2)$
Refer to Example of Examples earlier.

\[ Gain(\text{Outlook}) \approx 0.246 \]
\[ Gain(\text{Temp}) \approx 0.029 \]
Outlook has the highest gain.
Overcast branch is pure.
Need to construct DTs for two branches.
Other Choices

- When to stop:
  - all examples are classified the same
  - all examples have the same feature values
  - too few examples

- Overfitting occurs when the algorithm tries to fit noise in the training data (outliers, random fluctuations, approx. decision boundary).

- Handling overfitting: use part of training set as a validation set.
  - create decision tree with training set
  - prune decision tree with validation set
Special Cases in Decision Trees

- Feature $X_i$ is numeric.
  - Find best $X_i \leq v$ test. Requires sorting.
  - Or: Discretization. Partition $X_i$ into ranges.

- Feature $X_i$ has missing values.
  - Pretend missing is just another value.
  - Or: Ignore missing values. Split examples with missing values across branches.

- Feature $X_i$ has many discrete values.
  - Find best $X_i = v$ test. Forms binary tree.
  - Or: Partition values into subsets.
## Iris Dataset

<table>
<thead>
<tr>
<th>No.</th>
<th>Input Features</th>
<th>Target</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Sepal length</td>
<td>Sepal width</td>
</tr>
<tr>
<td>1</td>
<td>5.8</td>
<td>4.0</td>
</tr>
<tr>
<td>2</td>
<td>5.7</td>
<td>4.4</td>
</tr>
<tr>
<td>3</td>
<td>4.8</td>
<td>3.4</td>
</tr>
<tr>
<td>4</td>
<td>5.0</td>
<td>3.5</td>
</tr>
<tr>
<td>5</td>
<td>6.7</td>
<td>3.0</td>
</tr>
<tr>
<td>6</td>
<td>6.0</td>
<td>2.7</td>
</tr>
<tr>
<td>7</td>
<td>5.9</td>
<td>3.2</td>
</tr>
<tr>
<td>8</td>
<td>6.0</td>
<td>3.4</td>
</tr>
<tr>
<td>9</td>
<td>7.6</td>
<td>3.0</td>
</tr>
<tr>
<td>10</td>
<td>4.9</td>
<td>2.5</td>
</tr>
<tr>
<td>11</td>
<td>7.3</td>
<td>2.9</td>
</tr>
<tr>
<td>12</td>
<td>6.7</td>
<td>2.5</td>
</tr>
</tbody>
</table>

---

Learning

- Decision Trees
- Naive Bayes
- Linear Regression and Classification

**Learning Trees**

- Definition
- Example
- Selecting a Feature
- Information Plot
- Gain Plot
- Selection I
- Selection II
- Other Choices

**Special Cases**

- Iris Dataset

**Naive Bayes**

- Standard Form
- Independence Assumption
- Conditional Independence

**Linear Regression and Classification**

- Linear Regression
- Classification
Numerical learning methods learn the parameters or weights of a model, often by optimizing an error function. Examples include:

- Calculate the parameters of a probability distribution.
- Separate positive from negative examples by a decision boundary.
- Find points close to positive but far from negative examples.
- Update parameters to decrease error.
For target class $Y$ and features $X_i$, assume:

$$P(Y, X_1, \ldots, X_n) = P(Y)P(X_1|Y)\ldots P(X_n|Y)$$

This corresponds to a Bayesian network where $Y$ is the sole parent of each $X_i$.

To calculate the belief in $Y$:

$$P(Y \mid X_1, \ldots, X_n) = \frac{P(Y, X_1, \ldots, X_n)}{P(X_1, \ldots, X_n)}$$

The denominator is the same for all values of $Y$, so to compare only the numerator needs to be calculated.
Estimate prior and conditional probabilities by counting, e.g.,
- \( Y = \text{pos} \) in 9 of the 14 examples
- \( X_1 = \text{sunny} \) in 2 examples where \( Y = \text{pos} \).

If an outcome occurs \( m \) times out of \( n \) examples, Laplace’s law of succession recommends the estimate \( (m + 1)/(n + k) \) where \( k \) is the number of outcomes.

Estimate
- \( P(Y = \text{pos}) = (9 + 1)/(14 + 2) = 10/16 \)
- Estimate \( P(X_1 = \text{sunny} \mid Y = \text{pos}) = (2 + 1)/(9 + 3) = 3/12 \)
Using Laplace’s law of succession on the 14 examples:

\[
P(Y = \text{pos}) = \frac{9 + 1}{14 + 2} = \frac{10}{16}
\]

\[
P(Y = \text{neg}) = \frac{5 + 1}{14 + 2} = \frac{6}{16}
\]

\[
P(X_1 = \text{sunny} | Y = \text{pos}) = \frac{2 + 1}{9 + 3} = \frac{3}{12}
\]

\[
P(X_1 = \text{overcast} | Y = \text{pos}) = \frac{4 + 1}{9 + 3} = \frac{5}{12}
\]

\[
P(X_1 = \text{rain} | Y = \text{pos}) = \frac{3 + 1}{9 + 3} = \frac{4}{12}
\]
For the first example:

\[
P(Y = \text{pos} \mid \text{sunny, hot, high, false})
\]

\[
= \alpha \frac{10}{16} \frac{3}{12} \frac{3}{12} \frac{4}{11} \frac{7}{11}
\]

\[
\approx \alpha 0.00904
\]

\[
P(Y = \text{neg} \mid \text{sunny, hot, high, false})
\]

\[
= \alpha \frac{6}{16} \frac{4}{8} \frac{3}{8} \frac{5}{7} \frac{3}{7}
\]

\[
\approx \alpha 0.02152
\]

\[
\approx \frac{0.02152}{0.00904 + 0.02152} \approx 0.704
\]
A linear function of the input features is a dot product of the weights and the inputs.

Inputs: \( \mathbf{x} = (1.0, x_1, \ldots, x_n) \)
Weights: \( \mathbf{w} = (w_0, w_1, \ldots, w_n) \)
Dot product: \( \mathbf{w} \cdot \mathbf{x} = w_0 + w_1 x_1 + \ldots + w_n x_n \)

If \( y \) is the target and \( \hat{y} = \mathbf{w} \cdot \mathbf{x} \):

Regression:
- Squared error: \( (y - \hat{y})^2 \)
- Absolute error: \( |y - \hat{y}| \)

Classification (assume \( y \in \{-1, 1\} \)):
- Hinge loss: \( \max(0, 1 - y\hat{y}) \)
- Logistic loss: \( -\log(1/(1 + e^{-y\hat{y}})) \)
Why $y \neq \hat{y}$ all over the place?

- The goal of linear regression is $y = \hat{y}$.
- The goal of linear classification is not $y = \hat{y}$, but $\text{sign}(y) = \text{sign}(\hat{y})$.
- If $y$ and $\hat{y}$ have the same sign, then $y\hat{y} > 0$.
- The hinge loss includes a margin. Its goal is $y\hat{y} \geq 1$.
- The logistic loss is for interpreting $\hat{y}$ probabilistically (as a log-likelihood). It has larger values for negative $y\hat{y}$.
- The global minimum can be found for the hinge and logistic loss.
## Example of Numeric Examples

<table>
<thead>
<tr>
<th>No.</th>
<th>Input Features</th>
<th>Target Feature</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Sunny</td>
<td>Rainy</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>2</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>3</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>4</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>5</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>6</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>7</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>8</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>9</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>10</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>11</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>12</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>13</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>14</td>
<td>0</td>
<td>1</td>
</tr>
</tbody>
</table>
Generic Linear Learning Algorithm

Procedure *LinearLearner*(\(X, Y, E, \eta\))

Inputs \(X\): set of input features, \(X = \{X_1, \ldots, X_n\}\)

\(Y\): target feature

\(E\): set of training examples

\(\eta\): learning rate

initialize all weights \(w_0, w_1, \ldots, w_n\) to zero

repeat until termination

for each example \(e = (x, y) \in E\)

\(\hat{y} \leftarrow w \cdot x\)

\(\delta \leftarrow \) update based on \(y\) and \(\hat{y}\)

\(w \leftarrow w + \eta \delta x\)

return \(w\)
### Updates

<table>
<thead>
<tr>
<th>Name</th>
<th>( \delta ) ←</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Regression:</strong></td>
<td></td>
</tr>
<tr>
<td>Squared error</td>
<td>( y - \hat{y} )</td>
</tr>
<tr>
<td>Absolute error</td>
<td>( \text{sign}(y - \hat{y}) )</td>
</tr>
<tr>
<td><strong>Classification:</strong></td>
<td></td>
</tr>
<tr>
<td>Perceptron</td>
<td>if ( yy \leq 0 ) then ( y ) else 0</td>
</tr>
<tr>
<td>Hinge loss</td>
<td>if ( yy &lt; 1 ) then ( y ) else 0</td>
</tr>
<tr>
<td>Logistic loss</td>
<td>( y/(1 + e^{yy}) )</td>
</tr>
</tbody>
</table>

Except for perceptron, the update is based on the derivative of the error wrt the weights.

Note: For squared error, the optimal solution can be directly computed.
Using the learning rate $\eta = 1$:

<table>
<thead>
<tr>
<th>Features</th>
<th>$X_0$</th>
<th>$X_1$</th>
<th>$X_2$</th>
<th>$X_3$</th>
<th>$X_4$</th>
<th>$Y$</th>
<th>$\hat{y}$</th>
<th>$\hat{y}Y$</th>
<th>$w_0$</th>
<th>$w_1$</th>
<th>$w_2$</th>
<th>$w_3$</th>
<th>$w_4$</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>-1</td>
<td>0</td>
<td>0</td>
<td>-1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td></td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>-1</td>
<td>-1</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>-1</td>
</tr>
<tr>
<td></td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>2</td>
<td>2</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>-1</td>
</tr>
<tr>
<td></td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>-1</td>
<td>0</td>
<td>0</td>
<td>-1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td></td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>-1</td>
<td>-1</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>-2</td>
</tr>
<tr>
<td></td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>-1</td>
<td>-1</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>-2</td>
</tr>
<tr>
<td></td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>-2</td>
</tr>
<tr>
<td></td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>-1</td>
<td>-1</td>
<td>1</td>
<td>2</td>
<td>1</td>
<td>1</td>
<td>-1</td>
</tr>
<tr>
<td></td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>-1</td>
<td>2</td>
<td>-2</td>
<td>0</td>
<td>2</td>
<td>0</td>
<td>1</td>
<td>-1</td>
</tr>
</tbody>
</table>
The perceptron can learn *linearly separable* examples with zero error. Linearly separable = exists $w$ with zero error on all examples.

<table>
<thead>
<tr>
<th>or</th>
<th>and</th>
<th>xor</th>
</tr>
</thead>
<tbody>
<tr>
<td>1 + + 1 - + 1 + -</td>
<td></td>
<td></td>
</tr>
<tr>
<td>0 - + 0 - - 0 - +</td>
<td></td>
<td></td>
</tr>
<tr>
<td>0 1 0 1 0 1</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Usually, many *epochs* (passes over the training examples) are needed until convergence.

If zero error is not possible, use hinge/logistic loss and $\eta \approx 0.1/n$, where $n$ is $\max x \cdot x$. 