Announcements

• HW1 due today
• Most of last lecture was on the blackboard.
• Gather hand data set today.

Linear Discriminant Functions (Sections 5.1-5.2)

Linear discriminant functions and decisions surfaces

• Definition

It is a function that is a linear combination of the components of x

\[ g(x) = w^T x + w_0 \quad (1) \]

where \( w \) is the weight vector and \( w_0 \) the bias

• A two-category classifier with a discriminant function of the form (1) uses the following rule:

Decide \( \omega_1 \) if \( g(x) > 0 \) and \( \omega_2 \) if \( g(x) < 0 \)

\[ \Leftrightarrow \text{Decide } \omega_1 \text{ if } w^T x > -w_0 \text{ and } \omega_2 \text{ otherwise} \]

If \( g(x) = 0 \) \( \Rightarrow x \) is assigned to either class

• The equation \( g(x) = 0 \) defines the decision surface that separates points assigned to the category \( \omega_1 \) from points assigned to the category \( \omega_2 \)

• When \( g(x) \) is linear, the decision surface is a hyperplane

• Algebraic measure of the distance from \( x \) to the hyperplane (interesting result!)
In conclusion, a linear discriminant function divides the feature space by a hyperplane decision surface. The orientation of the surface is determined by the normal vector \( \mathbf{w} \) and the location of the surface is determined by the bias \( d(0, \mathbf{H}) = \mathbf{w} \). In particular, \( d(0, \mathbf{H}) = \frac{\mathbf{w} \cdot \mathbf{x} - d(0, \mathbf{H})}{\|\mathbf{w}\|} \).

The multi-category case:
- We define \( c \) linear discriminant functions and assign \( x \) to \( \omega_i \) if \( g_i(x) > g_j(x) \) for all \( j \neq i \); in case of ties, the classification is undefined.
- In this case, the classifier is a "linear machine".
- A linear machine divides the feature space into \( c \) decision regions, with \( g(x) \) being the largest discriminant if \( x \) is in the region \( R_i \).
- For a two contiguous regions \( R_i \) and \( R_j \), the boundary that separates them is a portion of hyperplane \( \mathbf{H}_{ij} \) defined by:
  \[ g_i(x) = g_j(x) \iff (\mathbf{w}_i - \mathbf{w}_j)^T \mathbf{x} + (\mathbf{w}_i - \mathbf{w}_j) = 0 \]
- \( \mathbf{w}_i - \mathbf{w}_j \) is normal to \( \mathbf{H}_{ij} \) and the distance is:
  \[ d(x, \mathbf{H}_i) = \frac{R_i - R_j}{\|\mathbf{w}_i - \mathbf{w}_j\|} \]

It is easy to show that the decision regions for a linear machine are convex, this restriction limits the flexibility and accuracy of the classifier.

---

**Perceptron**

Linear, threshold units

\[
o(x_1, \ldots, x_n) = \begin{cases} 1 \text{ if } w_1 x_1 + \cdots + w_n x_n > 0 \\ -1 \text{ otherwise.} \end{cases}
\]

The threshold can be easily forced to 0 by introducing an additional weight input \( W_0 = 0 \)
How powerful is a perceptron?

Threshold = 0

Inverter

Boolean AND

Boolean OR

Boolean XOR

Concept Space & Linear Separability

Linear Separability

Linear Separability

Training Perceptron

Perceptron Training Rule

Converges, if...

...training data linearly separable
...step size \( \eta \) sufficiently small
...no "hidden" units

\[
\Delta w_i = \eta (t - o) x_i
\]

Where:
- \( t \) is target value
- \( o \) is perceptron output
- \( \eta \) is small constant (e.g., 0.1) called learning rate

Gradient Descent

Learn \( \vec{w} \)'s that minimize squared error

\[
E[\vec{w}] = \frac{1}{2} \sum_{d \in D} (t_d - o_d)^2
\]

- \( D \) - training data

Gradient Descent

To find the best direction in the feature space we compute the gradient of \( E \) with respect to each of the components of \( \vec{w} \)

\[
\nabla E(\vec{w}) = \begin{bmatrix}
\frac{\partial E}{\partial w_1} & \frac{\partial E}{\partial w_2} & \cdots & \frac{\partial E}{\partial w_n}
\end{bmatrix}
\]

- This vector specifies the direction the produces the steepest increase in \( E \);
- We want to modify \( \vec{w} \) in the direction of \(-\nabla E(\vec{w})\)

\[
\vec{w} = \vec{w} + \Delta \vec{w}
\]

- Where:

\[
\Delta \vec{w} = -\nabla E(\vec{w})
\]
Batch Learning

- Initialize each $w_i$ to small random value
- Repeat until termination:
  \[ \Delta w_i = 0 \]
  For each training example $d$ do
  \[ o_d \leftarrow \sigma(\sum w_i x_{id}) \]
  \[ \Delta w_i \leftarrow \Delta w_i + \eta (o_d - o_d) o_d (1-o_d) x_{id} \]
  \[ w_i \leftarrow w_i + \Delta w_i \]

Incremental (Online) Learning

- Initialize each $w_i$ to small random value
- Repeat until termination:
  For each training example $d$ do
  \[ \Delta w_i = 0 \]
  \[ o_d \leftarrow \Sigma w_i x_{id} \]
  \[ \Delta w_i \leftarrow \Delta w_i + \eta (o_d - o_d) o_d (1-o_d) x_{id} \]
  \[ w_i \leftarrow w_i + \Delta w_i \]

Summary: Single Layer Networks

- Variety of update rules
  - Multiplicative: $\Delta w_i = R(o_d - o_d)x_{id}$
  - Additive
- Batch and incremental algorithms
- Various convergence and efficiency conditions
- There are other ways to learn linear functions
  - Linear Programming (general purpose)
  - Probabilistic Classifiers (some assumption)
- Although simple and restrictive -- linear predictors perform very well on many realistic problems
- However, the representational restriction is limiting in many applications

Increasing Expressiveness: Multi-Layer Neural Networks

Boolean XOR

<table>
<thead>
<tr>
<th>Input $x_1$</th>
<th>Input $x_2$</th>
<th>Output</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>0</td>
</tr>
</tbody>
</table>

2-layer Neural Net