Linear Regression

Linear regression is a model in which the output is a linear function of the input features.

\[ \hat{Y}(e) = w_0 + w_1 X_1(e) + \cdots + w_n X_n(e) \]

where \( \vec{w} = \langle w_0, w_1, \ldots w_n \rangle \). We invent a new feature \( X_0 \equiv 1 \), to make it not a special case.

The sum of squares error on examples \( E \) for output \( Y \) is:

\[ \text{Error}(E, \vec{w}) = \sum_{e \in E} (Y(e) - \hat{Y}(e))^2 \]

\[ = \sum_{e \in E} (Y(e) - \sum_{i=0}^{n} w_i X_i(e))^2 \]

Goal: find weights that minimize \( \text{Error}(E, \vec{w}) \).

Finding weights that minimize \( \text{Error}(E, \vec{w}) \)

Find the minimum analytically. Effective when it can be done (e.g., for linear regression). If

- \( \vec{y} = [Y(e_1), Y(e_2), \ldots, Y(e_M)] \) is a vector of the output features for the \( M \) examples
- \( X \) is a matrix where the \( j^{th} \) column is the values of the input features for the \( j^{th} \) example
- \( \vec{w} = [w_0, w_1, \ldots, w_n] \) is a vector of the weights then,

\[ \vec{y}^T = \vec{w} X \]

\[ \vec{y}^T X^T (XX^T)^{-1} = \vec{w} \]

\( (XX^T)^{-1} \) is the pseudo-inverse.
Finding weights that minimize \( \text{Error}_E(\vec{w}) \)

Find the minimum iteratively. Works for larger classes of problems (not just linear).

**Gradient descent**:

\[
    w_i \leftarrow w_i - \eta \frac{\partial \text{Error}(E, \vec{w})}{\partial w_i}
\]

\( \eta \) is the gradient descent step size, the learning rate. If

\[
    \text{Error}(E, \vec{w}) = \sum_{e \in E} (Y(e) - \hat{Y}(\vec{w})(e))^2 = \sum_{e \in E} \left( Y(e) - \sum_{i=0}^{n} w_i X_i(e) \right)^2
\]

then update rule:

\[
    w_i \leftarrow w_i + \eta \sum_{e \in E} \left( Y(e) - \sum_{i=0}^{n} w_i X_i(e) \right) X_i(e)
\]

where we have set \( \eta \rightarrow 2\eta \) (arbitrary scale)

### Incremental Gradient Descent for Linear Regression

1. **procedure** `LinearLearner(X, Y, E, \eta)`
2. **Inputs**
   - \( X \): set of input features, \( X = \{X_1, \ldots, X_n\} \)
   - \( Y \): output feature
   - \( E \): set of examples from which to learn
   - \( \eta \): learning rate
3. initialize \( w_0, \ldots, w_n \) randomly
4. repeat
5. for each example \( e \) in \( E \) do
6. \( \delta \leftarrow Y(e) - \sum_{i=0}^{n} w_i X_i(e) \)
7. for each \( i \in [0, n] \) do
8. \( w_i \leftarrow w_i + \eta \delta X_i(e) \)
9. until some stopping criteria is true
10. return \( w_0, \ldots, w_n \)

### Stochastic and Batched Gradient Descent

- Algorithm on the last slide is **incremental** gradient descent
- If examples are chosen randomly at line 8 then its **stochastic gradient descent**.
- **Batched gradient descent**:
  - process a batch of size \( n \) before updating the weights
  - if \( n \) is all the data, then its **gradient descent**
  - if \( n = 1 \), its **incremental gradient descent**
- Incremental can be more efficient than batch, but convergence not guaranteed

### Linear Classifier

- Assume we are doing **binary classification**, with classes \( \{0, 1\} \)
- There is no point in making a prediction of less than 0 or greater than 1.
- A **squashed linear function** is of the form:

\[
    \hat{Y}(\vec{w})(e) = f(w_0 + w_1 X_1(e) + \cdots + w_n X_n(e))
\]

\[
    = f(\sum_{i=0}^{n} w_i X_i(e))
\]

where \( f \) is an **activation function**.

- A simple activation function is the **step function**:

\[
    f(x) = \begin{cases} 
        1 & \text{if } x \geq 0 \\
        0 & \text{if } x < 0 
    \end{cases}
\]
Gradient Descent for Linear Classifiers

If the activation function is differentiable, we can use gradient descent to update the weights. The sum of squares error:

$$Error(E, \vec{w}) = \sum_{e \in E} \left( Y(e) - f \left( \sum_{i=0}^{n} w_i * X_i(e) \right) \right)^2$$

The partial derivative with respect to weight $w_i$ is:

$$\frac{\partial Error(E, \vec{w})}{\partial w_i} = -2 * \delta * f'(\sum_{i} w_i * X_i(e)) * X_i(e)$$

where $\delta = (Y(e) - f(\sum_{i=0}^{n} w_i X_i(e)))$.

Thus, each example $e$ updates each weight $w_i$ by

$$w_i \leftarrow w_i + \eta * \delta * f' \left( \sum_{i} w_i * X_i(e) \right) * X_i(e)$$

**The sigmoid or logistic activation function**

$$f(x) = \frac{1}{1 + e^{-x}}$$

$$f'(x) = \frac{e^{-x}}{(1 + e^{-x})^2} = f(x)(1 - f(x))$$

So $f'(x)$ can be computed from $f(x)$.

Discussion Board Example

Using the 18 training examples from lecture 4, this can be found in about 3000 iterations with a learning rate of $\eta = 0.05$. 

$$\hat{Reads}(e) = \text{sigmoid}(-8 + 7*\text{Short}(e) + 3*\text{New}(e) + 3*\text{Known}(e))$$
A dataset is **linearly separable** if there is a hyperplane where the classification is true on one side of the hyperplane and false on the other side.

The hyperplane is defined by where the predicted value, \( f(\mathbf{w}) = f(w_0 + w_1 x_1(e) + \cdots + w_n x_n(e)) \) is 0.5. For the sigmoid function, the hyperplane is defined by 
\[ w_0 + w_1 x_1(e) + \cdots + w_n x_n(e) = 0. \]

Some data are not linearly separable:

\[
\begin{array}{c|c}
1 & + \\
0 & - \\
\end{array}
\quad \begin{array}{c|c}
1 & - \\
0 & + \\
\end{array}
\quad \begin{array}{c|c}
1 & + \\
0 & - \\
\end{array}
\begin{array}{|c|c|c|}
\hline
& & \\
\hline
& & \\
\hline
& & \\
\hline
& & \\
\hline
\end{array}
\]

Kernel Trick

Data is not linearly separable:

\[
\begin{array}{c|c}
1 & + \\
0 & - \\
\end{array}
\quad \begin{array}{c|c}
2 & + \\
1 & - \\
\end{array}
\quad \begin{array}{c|c}
3 & + \\
2 & - \\
\end{array}
\quad \begin{array}{c|c}
4 & + \\
3 & - \\
\end{array}
\begin{array}{|c|c|c|}
\hline
& & \\
\hline
& & \\
\hline
& & \\
\hline
& & \\
\hline
\end{array}
\]

Add another dimension, data is now linearly separable:

\[
\begin{array}{c|c}
1 & + \\
0 & - \\
\end{array}
\quad \begin{array}{c|c}
2 & + \\
1 & - \\
\end{array}
\quad \begin{array}{c|c}
3 & + \\
2 & - \\
\end{array}
\quad \begin{array}{c|c}
4 & + \\
3 & - \\
\end{array}
\begin{array}{|c|c|c|}
\hline
& & \\
\hline
& & \\
\hline
& & \\
\hline
& & \\
\hline
\end{array}
\]

\( y = \text{rem}(x/2) \)
Kernel Trick: another example

\[ \phi(x_1, x_2) \rightarrow (x_1^2, \sqrt{2}x_1x_2, x_2^2) \]

\[ \left( \frac{x_1}{a} \right)^2 + \left( \frac{x_2}{b} \right)^2 = 1 \rightarrow \frac{z_1}{a^2} + \frac{z_3}{b^2} = 1 \]

Example

Support Vector Machines

Mercer’s Theorem

Key idea:
- Mercer’s Theorem
- A dot product in the new “lifted” space = function (kernel) in old space
- Means: never have to know what \( \phi \) is!!
- Only have to compute distances with the kernel.

Circle data is linearly separable if distance (dot product) is computed using \( K(x,w) \)

\[ K(x, w) = \langle \phi(x), \phi(w) \rangle \]
\[ = x_1^2 w_1^2 + 2x_1x_2 w_1 w_2 + x_2^2 w_2^2 \]
\[ = (x_1 w_1 + x_2 w_2)^2 \]
\[ = (\langle x, w \rangle)^2 \]

find the classification boundary with the widest margin

\[ o : c_i = -1 \]
\[ x : c_i = +1 \]

minimize \( ||w||^2 \) subject to \( c_i(w \cdot x_i - b) > 1 \)

Quadratic Programming problem

Also: use Kernel trick
Neural Networks

- inspired by biological networks (brain)
- connect up many simple units
- simple neuron: threshold and fire
- can help gain understanding of how biological intelligence works

Neural Networks

can learn the same things that a decision tree can

- imposes different learning bias (way of making new predictions)
- back-propagation learning: errors made are propagated backwards to change the weights
- often the linear and sigmoid layers are treated as a single layer

Neural Networks Basics

- Each node $j$ has a set of weights $w_{j0}, w_{j1}, \ldots, w_{jN}$
- Each node $j$ receives inputs $v_0, v_1, \ldots, v_N$
- number of weights = number of parents + 1 ($v_0 = 1$ constant bias term)
- output is the activation function output

$$a_j = f \left( \sum_i w_{ji} v_i \right)$$

necessarily non-linear because
A linear function of a linear function is a ...

activation functions:

- step function = integrate-and-fire (biological)
  $$f(z) = \begin{cases} 
  c & \text{if } z \geq 0 \\
  1 & \text{if } z < 0
  \end{cases}$$

- sigmoid function
  $$f(z) = \frac{1}{1 + e^{-z}}$$

- rectified linear (ReLU)
  $$g(z) = \max\{0, z\}$$

output of entire network is the classification result
Learning weights

**back-propagation** implements stochastic gradient descent

Recall:

\[ w_i \leftarrow w_i - \eta \frac{\partial \text{Error}(E, \vec{w})}{\partial w_i} \]

\( \eta \): learning rate.

Linear unit:

\[ \frac{\partial (aw + b)}{\partial w} = a \]

Sigmoid unit (chain rule):

\[ \frac{\partial f(g(w))}{\partial w} = f'(g(w)) \frac{\partial g(w)}{\partial w} \]

---

Using the *chain rule*, this can be extended throughout the network e.g. taking a derivative of the \(L^{th}\) layer w.r.t a weight in the \(R^{th}\) layer:

\[
\frac{\partial \text{output}_L}{\partial w_R} = \frac{\partial f(\text{output}_{L-1})}{\partial w_R} = f'(\text{output}_{L-1}) \sum_i w_{ji} \frac{\partial f(\text{output}_{L-2})}{\partial w_R} = f'(\text{output}_{L-1}) \sum_i w_{ji} f'(\text{output}_{L-2}) \ldots \sum_k w_{kR} \frac{\partial \text{input}_R}{\partial w_R} \]

\[
= f'(\text{output}_{L-1}) \sum_i w_{ji} f'(\text{output}_{L-2}) \ldots \text{input}_R
\]
Backpropagation

1: repeat
2: for each example $e$ in $E$ in random order do
3:   for each layer $i = 1 \ldots L$ do (forwards)
4:     $output_i = f(input_i)$
5:   for each layer $j = L \ldots 1$ do (backwards)
6:     compute back-propagated error
7:   update weights
8: until some stopping criteria is reached

Regularization

Regularized Neural nets: prevent overfitting, increased bias for reduced variance

- parameter norm penalties added to objective function
- dataset augmentation
- early stopping
- dropout
- parameter tying

- Convolutional Neural nets: used for images
- Recurrent Neural nets: used for sequences

Composite models

- Random Forests
  - Each decision tree in the forest is different
  - different features, splitting criteria, training sets
  - average or majority vote determines output
- Ensemble Learning: combination of base-level algorithms
- Boosting
  - sequence of learners
  - each learner is trained to fit the examples the previous learner did not fit well
  - learners progressively biased towards higher precision
  - early learners: lots of false positives, but reject all the clear negatives
  - later learners: problem is more difficult, but the set of examples is more focussed around the challenging boundary

Next:

- Unsupervised Learning with Uncertainty (Poole & Mackworth (2nd ed.) chapter 10.2,10.3,10.5)