Machine Learning:
Linear and logistic regression, decision trees

Milos Hauskrecht
milos@cs.pitt.edu
5329 Sennott Square
Supervised learning

**Data:** \( D = \{D_1, D_2, \ldots, D_n\} \) a set of \( n \) examples

\[ D_i = \langle x_i, y_i \rangle \]

\( x_i = (x_{i,1}, x_{i,2}, \ldots, x_{i,d}) \) is an input vector of size \( d \)

\( y_i \) is the desired output (given by a teacher)

**Objective:** learn the mapping \( f : X \rightarrow Y \)

\[ s.t. \quad y_i \approx f(x_i) \quad \text{for all} \quad i = 1, \ldots, n \]

- **Regression:**
  - \( Y \) is in \( \mathcal{R} \)

- **Classification**
  - \( Y \) is discrete
Linear regression

- **Function** \( f : X \to Y \) is a linear combination of input components

\[
f(x) = w_0 + w_1 x_1 + w_2 x_2 + \ldots w_d x_d = w_0 + \sum_{j=1}^{d} w_j x_j
\]

\( w_0, w_1, \ldots w_k \) - parameters (weights)

Bias term \[\rightarrow\] 1

Input vector \[\left\{ \begin{array}{c} x_1 \\ x_2 \\ \vdots \\ x_d \end{array} \right\}

\[\sum f(x, w)\]
Linear regression

- **Shorter (vector) definition of the model**
  - Include bias constant in the input vector
    \[ x = (1, x_1, x_2, \cdots x_d) \]
    \[ f(x) = w_0 x_0 + w_1 x_1 + w_2 x_2 + \cdots w_d x_d = w^T x \]
    \[ w_0, w_1, \ldots w_k \text{ - parameters (weights)} \]
Linear regression. Error.

- **Data:** \( D_i = \langle x_i, y_i \rangle \)
- **Function:** \( x_i \rightarrow f(x_i) \)
- We would like to have \( y_i \approx f(x_i) \) for all \( i = 1, \ldots, n \)

- **Error function** measures how much our predictions deviate from the desired answers

\[
J_n = \frac{1}{n} \sum_{i=1}^{n} (y_i - f(x_i))^2
\]

- **Learning:**
  We want to find the weights minimizing the error!
Linear regression. Example

- 1 dimensional input $x = (x_1)$
Linear regression. Example.

- 2 dimensional input \( \mathbf{x} = (x_1, x_2) \)
Linear regression. Optimization.

- We want the weights minimizing the error
  \[ J_n = \frac{1}{n} \sum_{i=1}^{n} (y_i - f(x_i))^2 = \frac{1}{n} \sum_{i=1}^{n} (y_i - w^T x_i)^2 \]
- For the optimal set of parameters, derivatives of the error with respect to each parameter must be 0
  \[ \frac{\partial}{\partial w_j} J_n(w) = -\frac{2}{n} \sum_{i=1}^{n} (y_i - w_0 x_{i,0} - w_1 x_{i,1} - \ldots - w_d x_{i,d}) x_{i,j} = 0 \]
- Vector of derivatives:
  \[ \text{grad}_w (J_n(w)) = \nabla_w (J_n(w)) = -\frac{2}{n} \sum_{i=1}^{n} (y_i - w^T x_i) x_i = \overline{0} \]
Linear regression. Optimization.

For the optimal set of parameters, derivatives of the error with respect to each parameter must be 0

\[ J_n = \frac{1}{n} \sum_{i=1}^{n} (y_i - f(x_i))^2 = \frac{1}{n} \sum_{i=1}^{n} (y_i - [w_0 + w_1 x^{(1)} + w_2 x^{(2)} + \ldots w_k x^{(k)}])^2 \]

- \[ \text{grad}_w (J_n (w)) = \overline{0} \] defines a set of equations in \( w \)

\[
\frac{\partial}{\partial w_0} J_n (w) = -\frac{2}{n} \sum_{i=1}^{n} [y_i - (w_0 + w_1 x^{(1)} + w_2 x^{(2)} + \ldots w_k x^{(k)})] = 0
\]

\[
\frac{\partial}{\partial w_1} J_n (w) = -\frac{2}{n} \sum_{i=1}^{n} [y_i - (w_0 + w_1 x^{(1)} + w_2 x^{(2)} + \ldots w_k x^{(k)})]x^{(1)} = 0
\]

\[
\frac{\partial}{\partial w_j} J_n (w) = -\frac{2}{n} \sum_{i=1}^{n} [y_i - (w_0 + w_1 x^{(1)} + w_2 x^{(2)} + \ldots w_k x^{(k)})]x^{(j)} = 0
\]

\[
\text{...}
\]
Solving linear regression

\[
\frac{\partial}{\partial w_j} J_n(w) = -\frac{2}{n} \sum_{i=1}^{n} (y_i - w_0 x_{i,0} - w_1 x_{i,1} - \ldots - w_d x_{i,d}) x_{i,j} = 0
\]

By rearranging the terms we get a **system of linear equations** with \( d+1 \) unknowns

\[
Aw = b
\]

\[
w_0 \sum_{i=1}^{n} x_{i,0} 1 + w_1 \sum_{i=1}^{n} x_{i,1} 1 + \ldots + w_j \sum_{i=1}^{n} x_{i,j} 1 + \ldots + w_d \sum_{i=1}^{n} x_{i,d} 1 = \sum_{i=1}^{n} y_i 1
\]

\[
w_0 \sum_{i=1}^{n} x_{i,0} x_{i,1} + w_1 \sum_{i=1}^{n} x_{i,1} x_{i,1} + \ldots + w_j \sum_{i=1}^{n} x_{i,j} x_{i,1} + \ldots + w_d \sum_{i=1}^{n} x_{i,d} x_{i,1} = \sum_{i=1}^{n} y_i x_{i,1}
\]

\[
\ldots
\]

\[
w_0 \sum_{i=1}^{n} x_{i,0} x_{i,j} + w_1 \sum_{i=1}^{n} x_{i,1} x_{i,j} + \ldots + w_j \sum_{i=1}^{n} x_{i,j} x_{i,j} + \ldots + w_d \sum_{i=1}^{n} x_{i,d} x_{i,j} = \sum_{i=1}^{n} y_i x_{i,j}
\]

\[
\ldots
\]
Solving linear regression

- The optimal set of weights satisfies:

\[ \nabla_w (J_n(w)) = -2 \frac{1}{n} \sum_{i=1}^{n} (y_i - w^T x_i)x_i = \mathbf{0} \]

Leads to a **system of linear equations (SLE)** with \( d+1 \) unknowns of the form

\[ A\mathbf{w} = \mathbf{b} \]

\[ w_0 \sum_{i=1}^{n} x_{i,0}x_{i,j} + w_1 \sum_{i=1}^{n} x_{i,1}x_{i,j} + \ldots + w_j \sum_{i=1}^{n} x_{i,j}x_{i,j} + \ldots + w_d \sum_{i=1}^{n} x_{i,d}x_{i,j} = \sum_{i=1}^{n} y_i x_{i,j} \]

**Solutions to SLE:**
- e.g. matrix inversion (if the matrix is singular)

\[ \mathbf{w} = A^{-1}\mathbf{b} \]
Gradient descent solution

• There are other ways to solve the weight optimization problem in the linear regression model

\[ J_n = Error(w) = \frac{1}{n} \sum_{i=1}^{n} (y_i - f(x_i, w))^2 \]

• A simple technique:
  – **Gradient descent**

  **Idea:**
  • Adjust weights in the direction that improves the Error
  • The gradient tells us what is the right direction

  \[ w \leftarrow w - \alpha \nabla_w Error_i(w) \]

  \[ \alpha > 0 \ - \ a \ learning \ rate \ (scales \ the \ gradient \ changes) \]
Gradient descent method

- Descend using the gradient information

- Change the value of $w$ according to the gradient

$$w \leftarrow w - \alpha \nabla_w Error_w(w)$$

Direction of the descent
**Gradient descent method**

- New value of the parameter
  \[ w_j \leftarrow w_j^* - \alpha \frac{\partial}{\partial w_j} \] Error(w) \bigg|_{w^*} \quad \text{For all } j \]

\[ \alpha > 0 \] - a learning rate (scales the gradient changes)
Gradient descent method

• Iteratively converge to the optimum of the Error function
On-line learning. Example
Extensions of simple linear model

Replace inputs to linear units with feature (basis) functions to model nonlinearities

\[ f(\mathbf{x}) = w_0 + \sum_{j=1}^{m} w_j \phi_j(\mathbf{x}) \]

\( \phi_j(\mathbf{x}) \) - an arbitrary function of \( \mathbf{x} \)

The same techniques as before to learn the weights
Extensions of the linear model

• Models linear in the parameters we want to fit

\[ f(x) = w_0 + \sum_{k=1}^{m} w_k \phi_k(x) \]

\( w_0, w_1 \ldots w_m \) - parameters
\( \phi_1(x), \phi_2(x) \ldots \phi_m(x) \) - feature or basis functions

• Basis functions examples:
  – a higher order polynomial, one-dimensional input \( x = (x_1) \)
    \[ \phi_1(x) = x \quad \phi_2(x) = x^2 \quad \phi_3(x) = x^3 \]
  – Multidimensional quadratic \( x = (x_1, x_2) \)
    \[ \phi_1(x) = x_1 \quad \phi_2(x) = x_1^2 \quad \phi_3(x) = x_2 \quad \phi_4(x) = x_2^2 \quad \phi_5(x) = x_1 x_2 \]
  – Other types of basis functions
    \[ \phi_1(x) = \sin x \quad \phi_2(x) = \cos x \]
Example. Regression with polynomials.

Regression with polynomials of degree $m$

- **Data points**: pairs of $<x, y>$
- **Feature functions**: $m$ feature functions
  \[ \phi_i(x) = x^i \quad i = 1, 2, \ldots, m \]
- **Function to learn**:
  \[ f(x, w) = w_0 + \sum_{i=1}^{m} w_i \phi_i(x) = w_0 + \sum_{i=1}^{m} w_i x^i \]
Multidimensional additive model example
Multidimensional additive model example
Binary classification

- **Two classes** \( Y = \{0, 1\} \)
- Our goal is to learn to classify correctly two types of examples
  - Class 0 – labeled as 0,
  - Class 1 – labeled as 1
- We would like to learn \( f : X \to \{0, 1\} \)
- **Zero-one error (loss) function**

\[
Error_1(x_i, y_i) = \begin{cases} 
1 & f(x_i, w) \neq y_i \\
0 & f(x_i, w) = y_i 
\end{cases}
\]

- Error we would like to minimize: \( E_{(x, y)}(Error_1(x, y)) \)
- **First step**: we need to devise a model of the function
Discriminant functions

- One way to represent a classifier is by using
  - **Discriminant functions**

- Works for binary and multi-way classification

- **Idea:**
  - For every class $i = 0, 1, \ldots, k$ define a function $g_i(x)$ mapping $X \rightarrow \mathbb{R}$
  - When the decision on input $x$ should be made choose the class with the highest value of $g_i(x)$

- So what happens with the input space? Assume a binary case.
Discriminant functions

- Example: Two classes in 2-D
Discriminant functions

- Discriminant functions $g_0(x)$ and $g_1(x)$ define the decision boundary

\[ g_1(x) \geq g_0(x) \]

\[ g_1(x) = g_0(x) \]

\[ g_1(x) \leq g_0(x) \]
Quadratic decision boundary

\[ g_1(x) \geq g_0(x) \]

\[ g_1(x) \leq g_0(x) \]

\[ g_1(x) = g_0(x) \]
Logistic regression model

- Defines a linear decision boundary
- Discriminant functions:
  \[ g_1(x) = g(w^T x) \quad g_0(x) = 1 - g(w^T x) \]
- where \( g(z) = 1/(1 + e^{-z}) \) - is a logistic function

\[ f(x, w) = g_1(w^T x) = g(w^T x) \]
Logistic function

function \( g(z) = \frac{1}{1 + e^{-z}} \)

- Is also referred to as a **sigmoid function**
- Replaces the threshold function with smooth switching
- Takes a real number and outputs the number in the interval \([0,1]\)
Logistic regression model

• **Discriminant functions:**
  
  $$g_1(x) = g(w^T x) \quad g_0(x) = 1 - g(w^T x)$$

• Values of discriminant functions vary in $[0,1]$
  
  – **Probabilistic interpretation**
  
  $$f(x, w) = p(y = 1 \mid w, x) = g_1(x) = g(w^T x)$$
Logistic regression

- We learn a probabilistic function
  
  \[ f : X \rightarrow [0,1] \]
  
  - where \( f \) describes the probability of class 1 given \( x \)

  \[ f(x, w) = g_1(w^T x) = p(y = 1 | x, w) \]

  Note that:
  
  \[ p(y = 0 | x, w) = 1 - p(y = 1 | x, w) \]

- Transformation to binary class values:

  If \( p(y = 1 | x) \geq 1/2 \) then choose 1
  Else choose 0
Logistic regression model. Decision boundary

- Logistic Regression defines a linear decision boundary

**Example:** 2 classes (blue and red points)
Logistic regression: parameter learning

Likelihood of outputs

• Let
  \[ D_i = \langle x_i, y_i \rangle \quad \mu_i = p(y_i = 1 | x_i, w) = g(z_i) = g(w^T x) \]

• Then
  \[ L(D, w) = \prod_{i=1}^{n} P(y = y_i | x_i, w) = \prod_{i=1}^{n} \mu_i^{y_i} (1 - \mu_i)^{1-y_i} \]

• Find weights \( w \) that maximize the likelihood of outputs
  – Apply the log-likelihood trick The optimal weights are the same for both the likelihood and the log-likelihood
  
  \[
l(D, w) = \log \prod_{i=1}^{n} \mu_i^{y_i} (1 - \mu_i)^{1-y_i} = \sum_{i=1}^{n} \log \mu_i^{y_i} (1 - \mu_i)^{1-y_i} = \]
  
  \[= \sum_{i=1}^{n} y_i \log \mu_i + (1 - y_i) \log(1 - \mu_i) \]
Logistic regression: parameter learning

- **Log likelihood**
  \[ l(D, w) = \sum_{i=1}^{n} y_i \log \mu_i + (1 - y_i) \log(1 - \mu_i) \]

- **Derivatives of the loglikelihood**
  \[ -\frac{\partial}{\partial w_j} l(D, w) = \sum_{i=1}^{n} -x_{i,j} (y_i - g(z_i)) \]
  \[ \nabla_w -l(D, w) = \sum_{i=1}^{n} -x_i (y_i - g(w^T x_i)) = \sum_{i=1}^{n} -x_i (y_i - f(w, x_i)) \]

- **Gradient descent:**
  \[ w^{(k)} \leftarrow w^{(k-1)} - \alpha(k) \nabla_w [-l(D, w)] \bigg|_{w^{(k-1)}} \]

**k-th update of the weights**

\[ w^{(k)} \leftarrow w^{(k-1)} + \alpha(k) \sum_{i=1}^{n} [y_i - f(w^{(k-1)}, x_i)] x_i \]
Gradient algorithm. Example.

\[ w_1 = 0.91773 \quad w_2 = 1.6297 \quad \text{bias} = -0.91898 \]
Gradient algorithm. Example.

\[ w_1 = 3.5934 \quad w_2 = 6.9126 \quad \text{bias} = -3.6709 \]
Gradient algorithm. Example.

\[ w_1=19.9144 \quad w_2=39.7033 \quad \text{bias}= -20.8644 \]
Decision trees

• An alternative approach to classification:
  – **Partition the input space to regions**
  – **Regress or classify independently in every region**
Decision trees

- An alternative approach to classification:
  - Partition the input space to regions
  - Regress or classify independently in every region
Decision trees

- **Decision tree model:**
  - Split the space recursively according to inputs in \( x \)
  - Classify at the bottom of the tree

**Example:**
Binary classification \{0,1\}
Binary attributes \( x_1, x_2, x_3 \)

```
Example:
Binary classification \{0,1\}
Binary attributes \( x_1, x_2, x_3 \)
```

```
<table>
<thead>
<tr>
<th>( x_1 = 0 )</th>
<th>( x_2 = 0 )</th>
<th>( x_3 = 0 )</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>1</td>
</tr>
</tbody>
</table>
```
Decision trees

- **Decision tree model:**
  - Split the space recursively according to inputs in $x$
  - Classify at the bottom of the tree

**Example:**
Binary classification $\{0,1\}$
Binary attributes $x_1, x_2, x_3$

$x = (x_1, x_2, x_3) = (1,0,0)$
Decision trees

- **Decision tree model:**
  - Split the space recursively according to inputs in $x$
  - Classify at the bottom of the tree

**Example:**
Binary classification $\{0,1\}$
Binary attributes $x_1, x_2, x_3$

$x = (x_1, x_2, x_3) = (1,0,0)$

```
class 1 0 0 1 1 0
```
Decision trees

- **Decision tree model:**
  - Split the space recursively according to inputs in \( x \)
  - Classify at the bottom of the tree

**Example:**
Binary classification \{0,1\}
Binary attributes \( x_1, x_2, x_3 \)

\[
x = (x_1, x_2, x_3) = (1,0,0)
\]
Decision trees

- **Decision tree model:**
  - Split the space recursively according to inputs in $x$
  - Classify at the bottom of the tree

**Example:**

Binary classification $\{0,1\}$
Binary attributes $x_1, x_2, x_3$

$x = (x_1, x_2, x_3) = (1,0,0)$

```
x_3 = 0
  t
  f
x_2 = 0
  t
  f
x_1 = 0
  t
  f
classify
1 0 0 1 1 0
```
Learning decision trees

How to construct /learn the decision tree?

• **Top-bottom algorithm:**
  – Find the best split condition (quantified based on the impurity measure)
  – Stops when no improvement possible

• **Impurity measure I:**
  – measures how well are the two classes in the training data D separated …. \( I(D) \)
  – Ideally we would like to separate all 0s and 1

• Splits: **finite or continuous value attributes**

Continuous value attributes conditions: \( x_3 \leq 0.5 \)
Impurity measure

Let $|D|$ - Total number of data entries in the training dataset

$|D_i|$ - Number of data entries classified as $i$

$$p_i = \frac{|D_i|}{|D|}$$ - ratio of instances classified as $i$

**Impurity measure** $I(D)$

- defines how well the classes are separated
- in general the impurity measure should satisfy:
  - Largest when data are split evenly for attribute values
    $$p_i = \frac{1}{\text{number of classes}}$$
  - Should be 0 when all data belong to the same class
Impurity measures

• There are various impurity measures used in the literature
  – **Entropy based measure** *(Quinlan, C4.5)*
    \[ I(D) = \text{Entropy}(D) = - \sum_{i=1}^{k} p_i \log p_i \]
    
    Example for k=2
    
  – **Gini measure** *(Breiman, CART)*
    \[ I(D) = \text{Gini}(D) = 1 - \sum_{i=1}^{k} p_i^2 \]
Impurity measures

- **Gain due to split** – expected reduction in the impurity measure (entropy example)

\[
\text{Gain}(D, A) = \text{Entropy}(D) - \sum_{v \in \text{Values}(A)} \frac{|D^v|}{|D|} \text{Entropy}(D^v)
\]

\(|D^v|\) - a partition of \(D\) with the value of attribute \(A = v\)
Decision tree learning

- **Greedy learning algorithm:**
  
  Repeat until no or small improvement in the purity
  - Find the attribute with the highest gain
  - Add the attribute to the tree and split the set accordingly

- Builds the tree in the top-down fashion
  - Gradually expands the leaves of the partially built tree

- The method is greedy
  - It looks at a single attribute and gain in each step
  - May fail when the combination of attributes is needed to improve the purity (parity functions)
Decision tree learning

- **Limitations of greedy methods**

  Cases in which a combination of two or more attributes improves the impurity
Decision tree learning

By reducing the impurity measure we can grow very large trees

Problem: Overfitting

• We may split and classify very well the training set, but we may do worse in terms of the generalization error

Solutions to the overfitting problem:

• Solution 1.
  – Prune branches of the tree built in the first phase
  – Use validation set to test for the overfit

• Solution 2.
  – Test for the overfit in the tree building phase
  – Stop building the tree when performance on the validation set deteriorates
Appendix: Derivation of the gradient

- Log likelihood
  \[ l(D, w) = \sum_{i=1}^{n} y_i \log \mu_i + (1 - y_i) \log(1 - \mu_i) \]

- Derivatives of the log likelihood
  \[
  \frac{\partial}{\partial w_j} l(D, w) = \sum_{i=1}^{n} \frac{\partial}{\partial z_i} \left[ y_i \log \mu_i + (1 - y_i) \log(1 - \mu_i) \right] \frac{\partial z_i}{\partial w_j}
  \]
  
  Derivative of a logistic function
  \[
  \frac{\partial g(z_i)}{\partial z_i} = g(z_i)(1 - g(z_i))
  \]

  \[
  \frac{\partial}{\partial z_i} \left[ y_i \log \mu_i + (1 - y_i) \log(1 - \mu_i) \right] = y_i \frac{1}{g(z_i)} \frac{\partial g(z_i)}{\partial z_i} + (1 - y_i) \frac{-1}{1 - g(z_i)} \frac{\partial g(z_i)}{\partial z_i}
  \]

  \[
  = y_i (1 - g(z_i)) + (1 - y_i)(-g(z_i)) = y_i - g(z_i)
  \]

  \[
  \nabla_w l(D, w) = \sum_{i=1}^{n} -x_i (y_i - g(w^T x_i)) = \sum_{i=1}^{n} -x_i (y_i - f(w, x_i))
  \]