Supervised learning

- input space $\mathcal{X}$
  - $x \in \mathcal{X}$ is called the covariate, feature, or independent variable
- output space $\mathcal{Y}$
  - $y \in \mathcal{Y}$ is called the response, outcome, label, or dependent variable
- given $\mathcal{D} = \{(x_1, y_1), \ldots, (x_n, y_n)\}$
  - $\mathcal{D}$ is called the data, examples, observations, samples or measurements
- want to find $f : \mathcal{X} \to \mathcal{Y}$ so that

$$f(x_i) \approx y_i, \quad i = 1, \ldots, n$$

(use $f$ to predict the outcome $y$ given new example $x$)
different names for different $\mathcal{Y}$'s:

- **classification**: $\mathcal{Y} = \{-1, 1\}$
- **regression**: $\mathcal{Y} = \mathbb{R}$
- **multiclass classification**: $\mathcal{Y} = \{\text{apple, banana, pear}\}$
- **ordinal regression**:
  $\mathcal{Y} = \{\text{strongly disagree, \ldots, strongly agree}\}$
Regression

examples where $\mathcal{Y} = \mathbb{R}$:

- predict credit score of applicant
- predict temperature in Ithaca a year from today
- predict travel time at rush hour
- predict blood alcohol level
Linear model

(if $\mathcal{X} = \mathbb{R}^d$)

- predict $y$ using a linear function $h : \mathbb{R}^d \to \mathbb{R}$

$$h(x) = w^T x = \sum_{i=1}^{d} w_i x_i$$

- we want $h(x_i) \approx y_i$ for every $i = 1, \ldots, n$
- and we hope $h(x) \approx y$ with high probability when

$$(x, y) \sim P(x, y)$$
Linear model++

- pick a transformation $\phi : \mathcal{X} \to \mathbb{R}^d$
- predict $y$ using a linear function of $\phi(x)$

$$h(x) = w^T \phi(x) = \sum_{i=1}^{d} w_i(\phi(x))_i$$

- we want $h(x_i) \approx y_i$ for every $i = 1, \ldots, n$
- and we hope $h(x) \approx y$ with high probability when

$$(x, y) \sim P(x, y)$$
Linear model++

- pick a transformation $\phi : \mathcal{X} \rightarrow \mathbb{R}^d$
- predict $y$ using a linear function of $\phi(x)$

$$h(x) = w^T \phi(x) = \sum_{i=1}^{d} w_i (\phi(x))_i$$

- we want $h(x_i) \approx y_i$ for every $i = 1, \ldots, n$
- and we hope $h(x) \approx y$ with high probability when $(x, y) \sim P(x, y)$

choices:
- how to pick $d$?
- how to pick $\phi$?
- how to pick $w$?
Linear model++

- pick a transformation \( \phi : \mathcal{X} \rightarrow \mathbb{R}^d \)
- predict \( y \) using a linear function of \( \phi(x) \)

\[
h(x) = w^T \phi(x) = \sum_{i=1}^{d} w_i (\phi(x))_i
\]

- we want \( h(x_i) \approx y_i \) for every \( i = 1, \ldots, n \)
- and we hope \( h(x) \approx y \) with high probability when

\[
(x, y) \sim P(x, y)
\]

choices:

- how to pick \( d \)?
- how to pick \( \phi \)?
- how to pick \( w \)?

for now, assume \( d \) and \( \phi \) are fixed; we’ll return to these later...
Least squares fitting

- $r_i = y_i - h(x_i)$ is prediction error or residual
- choose $w$ to minimize sum of square residuals

$$E_{\text{in}}(w) = \sum_{i=1}^{n} (r_i)^2 = \sum_{i=1}^{n} (y_i - h(x_i))^2 = \sum_{i=1}^{n} (y_i - w^T x_i)^2$$

- rewrite using linear algebra:
  - form vector $y \in \mathbb{R}^n$: each outcome $y_i$ is an entry of $y$
  - form matrix $X \in \mathbb{R}^{n \times d}$: each example $x_i$ is a row of $X$
  - rewrite error:

$$\sum_{i=1}^{n} (r_i)^2 = \sum_{i=1}^{n} (y_i - w^T x_i)^2 = \|y - Xw\|^2$$

- $Xw$ is a linear combination of the columns of $X$
- we seek the linear combination that best matches $y$
The gradient

the gradient \( \nabla f(x) \) generalizes the derivative.

for \( x \in \mathbb{R}^d, f : \mathbb{R}^d \to \mathbb{R} \) differentiable,

- \( \nabla f(x) = [\frac{\partial f}{\partial x_1}, \ldots, \frac{\partial f}{\partial x_d}] \in \mathbb{R}^d \)

- allows easy computation of directional derivatives: for fixed \( v \in \mathbb{R}^d \),

\[
\frac{\partial}{\partial \alpha} f(x + \alpha v) = \frac{\partial f}{\partial x_1} \frac{\partial x_1}{\partial \alpha} + \cdots + \frac{\partial x_d}{\partial \alpha} = (\nabla f(x))^T v
\]

- locally approximates \( f(x) \):

\[
f(x + \alpha v) \approx f(x) + \alpha (\nabla f(x))^T v
\]

we sometimes write \( \nabla_x f(x) \) when the variable is ambiguous
The gradient

\[ f(x + \alpha \nu) \approx f(x) + \alpha (\nabla f(x))^T \nu \]

**Q:** From the point \( x \), which direction \( \nu \) should we travel in to make \( f(x) \) **increase** as fast as possible?
The gradient

\[ f(x + \alpha v) \approx f(x) + \alpha (\nabla f(x))^T v \]

**Q:** From the point \( x \), which direction \( v \) should we travel in to make \( f(x) \) **increase** as fast as possible?

**A:** In the direction \( v = \nabla f(x) \), to maximize \((\nabla_x f(x))^T v\)
The gradient

\[ f(x + \alpha v) \approx f(x) + \alpha(\nabla f(x))^T v \]

**Q:** From the point \( x \), which direction \( v \) should we travel in to make \( f(x) \) **increase** as fast as possible?

**A:** In the direction \( v = \nabla f(x) \), to maximize \( (\nabla_x f(x))^T v \)

**Q:** From the point \( x \), which direction \( v \) should we travel in to make \( f(x) \) **decrease** as fast as possible?
The gradient

\[ f(x + \alpha v) \approx f(x) + \alpha (\nabla f(x))^T v \]

**Q:** From the point \( x \), which direction \( v \) should we travel in to make \( f(x) \) **increase** as fast as possible?

**A:** In the direction \( v = \nabla f(x) \), to maximize \( (\nabla_x f(x))^T v \)

**Q:** From the point \( x \), which direction \( v \) should we travel in to make \( f(x) \) **decrease** as fast as possible?

**A:** In the direction \( v = -\nabla f(x) \)
Some matrix calculus identities

two useful identities:

- $\nabla_w (w^T b) = b$
- $\nabla_w (w^T A w) = (A + A^T)w$

verify:

- take partial derivatives wrt each entry of $w$
- concatenate to get the matrix calculus result
Gradient of the least squares problem

\[ f(w) = \| y - Xw \|^2 \]

compute \( \nabla f(w) = \nabla_w \| y - Xw \|^2 \):

\[
\nabla f(w) = \nabla_w (y - Xw)^T (y - Xw) \\
= \nabla_w (y^T y - w^T X^T y - y^T Xw + w^T X^T Xw) \\
= -2X^T y + 2X^T Xw
\]
Gradient descent

minimize \( f(x) \)

idea: go downhill to get to the minimum!

**gradient descent algorithm:** to minimize \( f(x) \), repeat

- start at any \( x^{(0)} \in \mathbb{R}^d \)
- for \( k = 1, \ldots \)
  - update \( x^{(k)} = x^{(k-1)} - \alpha^{(k)} \nabla f(x^{(k-1)}) \)

nomenclature

- \( x^{(k)} \in \mathbb{R}^d \) are called **iterates**
- \( \alpha^{(k)} \in \mathbb{R} \) are called **step-sizes**
Gradient descent: choosing a step-size

- **constant step-size.** \( \alpha^{(k)} = \alpha \) (constant)
- **decreasing step-size.** \( \alpha^{(k)} = \frac{1}{k} \)
- **line search.** try different possibilities for \( \alpha^{(k)} \) until 
\[
 f(x^{(k-1)} - \alpha^{(k)} \nabla f(x^{(k)})) \]

\[
 \text{decreases enough}
\]
Solving the least squares problem: gradient descent

minimize \( \|y - Xw\|^2 \)

recall \( \nabla_w \|y - Xw\|^2 = -2X^Ty + 2X^TXw \)

gradient descent for least squares:

▶ start at any \( w^{(0)} \in \mathbb{R}^d \)
▶ for \( k = 1, \ldots \)
  ▶ update \( w^{(k)} = w^{(k-1)} - \alpha^{(k)}(2X^TXw^{(k-1)} - 2X^Ty) \)
Demo: gradient descent

https://github.com/ORIE4741/demos
Computational considerations: gradient descent

\[ w^{(k)} = w^{(k-1)} - \alpha^{(k)}(2X^TXw - 2X^Ty) \]

to compute this quickly:

- form the **Gram matrix** \( G = X^TX \) (2nd^2 flops)
- form \( b = X^Ty \) (2nd flops)
- for \( k = 1, \ldots \)
  - update \( w^{(k)} = w^{(k-1)} - \alpha^{(k)}(2Gw - 2b) \) (\( d^2 + 2d \) flops)

2nd^2 flops to start, plus \( O(d^2) \) per iteration
Computational considerations: gradient descent

\[ w^{(k)} = w^{(k-1)} - \alpha^{(k)} (2X^T Xw - 2X^T y) \]

to compute this quickly:

- form the **Gram matrix** \( G = X^T X \) (\( 2nd^2 \) flops)
- form \( b = X^T y \) (\( 2nd \) flops)
- for \( k = 1, \ldots \)
  - update \( w^{(k)} = w^{(k-1)} - \alpha^{(k)} (2Gw - 2b) \) (\( d^2 + 2d \) flops)

\( 2nd^2 \) flops to start, plus \( O(d^2) \) per iteration

**Q:** Is this a big data algorithm?

Yes: it’s \( O(n) \). Just make sure \( d \) doesn’t grow with \( n \).
Computational considerations: gradient descent

\[ w^{(k)} = w^{(k-1)} - \alpha^{(k)}(2X^T Xw - 2X^T y) \]

to compute this quickly:

- form the **Gram matrix** \( G = X^T X \) (2nd\(^2\) flops)
- form \( b = X^T y \) (2nd flops)
- for \( k = 1, \ldots \)
  - update \( w^{(k)} = w^{(k-1)} - \alpha^{(k)}(2Gw - 2b) \) (\( d^2 + 2d \) flops)

2nd\(^2\) flops to start, plus \( O(d^2) \) per iteration

**Q:** Is this a big data algorithm?

**A:** Yes: it’s \( O(n) \). Just make sure \( d \) doesn’t grow with \( n \).
Solving the least squares problem: straight to the bottom

minimize $\|y - Xw\|^2$

- solve by setting the derivative to 0: optimal $w$ satisfies

$$0 = \nabla_w \|y - Xw\|^2 = -2X^Ty + 2X^TXw$$

$$X^TXw = X^Ty$$

- make one assumption: $X$ has linearly independent columns
- so $X^TX$ is invertible, and

$$w = (X^TX)^{-1}X^Ty$$
\[ X^T X w = X^T y \]

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

- \( X^T X \) is called the **Gram matrix**
- \( X^T X w = X^T y \) is called the **normal equations**
- \( X^+ = (X^T X)^{-1} X^T \) is called the **pseudo-inverse** of \( X \)
  - \( X^+ X = I_d \) but \( XX^+ \neq I_n \)
  - if \( y \in \text{range}(X) \), then \( y = XX^+ y \)
The QR factorization

rewrite $X$ in terms of easier matrices

- $X = QR$
- $Q \in \mathbb{R}^{n \times d}$ has orthogonal columns: $Q^T Q = I_d$
- $R \in \mathbb{R}^{d \times d}$ is upper triangular: $R_{ij} = 0$ for $i > j$
- diagonal of $R \in \mathbb{R}^{d \times d}$ is positive: $R_{ii} > 0$ for $i = 1, \ldots, d$

can compute $QR$ factorization of $X$ in $2nd^2$ flops
The QR factorization

rewrite $X$ in terms of easier matrices

- $X = QR$
- $Q \in \mathbb{R}^{n \times d}$ has orthogonal columns: $Q^TQ = I_d$
- $R \in \mathbb{R}^{d \times d}$ is upper triangular: $R_{ij} = 0$ for $i > j$
- diagonal of $R \in \mathbb{R}^{d \times d}$ is positive: $R_{ii} > 0$ for $i = 1, \ldots, d$

can compute $QR$ factorization of $X$ in $2nd^2$ flops

in julia (or matlab), use the QR function

$Q, R = \text{qr}(X)$
use QR to solve least squares:

\[ w = (X^T X)^{-1} X^T y = (R^T Q^T QR)^{-1} R^T Q^T y \]
\[ = (R^T R)^{-1} R^T Q^T y \]
\[ = R^{-1} (R^T)^{-1} R^T Q^T y \]
\[ = R^{-1} Q^T y \]
Computational considerations

**never** form the inverse explicitly: numerically unstable!

instead, to compute $X^\dagger y$, use QR factorization:

- compute QR factorization of $X$ (2nd$^2$ flops)
- to compute $w = X^\dagger y = R^{-1}Q^T y$
  - form $b = Q^T y$ (2nd flops)
  - compute $w = R^{-1}b$ by back-substitution ($d^2$ flops)
Computational considerations

**never** form the inverse explicitly: numerically unstable!

instead, to compute $X^\dagger y$, use QR factorization:

- compute $QR$ factorization of $X$ ($2nd^2$ flops)
- to compute $w = X^\dagger y = R^{-1}Q^Ty$
  - form $b = Q^Ty$ ($2nd$ flops)
  - compute $w = R^{-1}b$ by back-substitution ($d^2$ flops)

in julia (or matlab), the **backslash operator** solves least-squares efficiently (usually, using QR)

$$w = X \backslash y$$
Demo: QR

https://github.com/ORIE4741/demos/linear.ipynb
References