ORIE 4741: Learning with Big Messy Data

Linear Models and Linear Least Squares

Professor Udell
Operations Research and Information Engineering
Cornell

August 29, 2019
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 $D = \{(x_1, y_1), \ldots, (x_n, y_n)\}$
  - $D$ is called the **data**, **examples**, **observations**, **samples** or **measurements**

- we believe there is some $f : \mathcal{X} \to \mathcal{Y}$ so that
  $$f(x_i) \approx y_i, \quad i = 1, \ldots, n$$

- we will find some $h \in \mathcal{H}$ so that (we hope!)
  $$h(x_i) \approx y_i, \quad i = 1, \ldots, n$$
Supervised learning

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**: 
  $\mathcal{Y} = \{\text{strongly disagree, …, strongly agree}\}$
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
(if $\mathcal{X} = \mathbb{R}^d$)

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

$$h(x) = w^\top x$$

- we want $h(x_i) \approx y_i$ for every $i = 1, \ldots, n$
Linear model++

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

$$h(x) = w^\top \phi(x)$$

- we want $h(x_i) \approx y_i$ for every $i = 1, \ldots, n$
Linear model++

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

$$h(x) = w^\top \phi(x)$$

- we want $h(x_i) \approx y_i$ for every $i = 1, \ldots, n$

choices:

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

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

$$h(x) = w^\top \phi(x)$$

- we want $h(x_i) \approx y_i$ for every $i = 1, \ldots, n$

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**

\[
\sum_{i=1}^{n} (r_i)^2 = \sum_{i=1}^{n} (y_i - h(x_i))^2 = \sum_{i=1}^{n} (y_i - w^\top x_i)^2
\]
Least squares fitting

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:

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

interpretation:

▶ \( Xw \) is a linear combination of the columns of \( X \)
▶ we seek the linear combination that best matches \( y \)
The gradient

the \textbf{gradient} $\nabla f(x)$ generalizes the derivative.

for $x \in \mathbb{R}^d$, $f : \mathbb{R}^d \rightarrow \mathbb{R}$ differentiable,

\begin{itemize}
  \item $\nabla f(x) = \left( \frac{\partial f}{\partial x_1}, \ldots, \frac{\partial f}{\partial x_d} \right) \in \mathbb{R}^d$
  \item allows easy computation of directional derivatives: for fixed $v \in \mathbb{R}^d$, let $y(\alpha) = x + \alpha v$. then
    \[
    \frac{d}{d\alpha} f(y(\alpha)) = \frac{\partial f}{\partial y_1} \frac{dy_1}{d\alpha} + \cdots + \frac{\partial f}{\partial y_d} \frac{dy_d}{d\alpha}
    = (\nabla f(y(\alpha)))^\top v
    \]
  \item locally approximates $f(x)$:
    \[
    f(x + \alpha v) \approx f(x) + \alpha (\nabla f(x))^\top v
    \]
\end{itemize}
The gradient

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

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

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

**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 \( (\nabla_x f(x))^\top v \)
The gradient

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

**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 \( (\nabla_x f(x))^\top v \)

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

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

**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 \((\nabla_x f(x))^\top v\)

**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 \( v = -\nabla f(x) \)
Demo: gradient descent

let’s verify these properties of gradients numerically
https://github.com/ORIE4741/demos
Some matrix calculus identities

two useful identities: let \( w, b \in \mathbb{R}^d, A \in \mathbb{R}^{d \times d} \)

1. let \( f(w) = w^\top b \). Then

\[
\nabla f(w) = b
\]

2. let \( f(w) = w^\top Aw \). Then

\[
\nabla f(w) = 2Aw
\]

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 \( \nabla f(w) \):

\[
\begin{align*}
\nabla f(w) &= \nabla (y - Xw)^\top (y - Xw) \\
&= \nabla (y^\top y - w^\top X^\top y - y^\top Xw + w^\top X^\top Xw) \\
&= -2X^\top y + 2X^\top Xw
\end{align*}
\]
Gradient descent

minimize $f(x)$

idea: go downhill to get to a (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. $\alpha^{(k)} = 1/k$
- line search. try different possibilities for $\alpha^{(k)}$ until

$$f(x^{(k-1)} - \alpha^{(k)} \nabla f(x^{(k-1)}))$$

decreases enough.
Line search

- The **Armijo rule** requires $\alpha^{(k)}$ to satisfy

\[
    f(x^{(k-1)} - \alpha^{(k)} \nabla f(x^{(k-1)})) \leq f(x^{(k-1)}) + c \alpha^{(k)} \| \nabla f(x^{(k-1)}) \|^2
\]

for some $c \in (0, 1)$, e.g., $c = .01$.

- A simple **backtracking line search** algorithm is
  - set $\alpha^{(k)} = 1$
  - while

\[
    f(x^{(k-1)} - \alpha^{(k)} \nabla f(x^{(k-1)})) \geq f(x^{(k-1)}) + c \alpha^{(k)} \| \nabla f(x^{(k-1)}) \|^2
\]

  - $\alpha^{(k)} = \alpha^{(k)}/2$

we can always satisfy the Armijo rule with small enough $\alpha^{(k)}$. Why?
Solving the least squares problem: gradient descent

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

recall $\nabla_w \|y - Xw\|^2 = -2X^\top y + 2X^\top Xw$

gradient descent for least squares:

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

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

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

to compute this quickly:

- form the **Gram matrix** \( G = X^\top X \) (2nd^2 flops)
- form \( b = X^\top y \) (2nd flops)
- for \( k = 1, \ldots \)
  - update \( w^{(k)} = w^{(k-1)} - \alpha^{(k)}(2Gw - 2b) \) (2d^2 + 3d flops)

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

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

to compute this quickly:

- form the Gram matrix \( G = X^\top X \) (2nd\(^2\) flops)
- form \( b = X^\top y \) (2nd flops)
- for \( k = 1, \ldots \)
  - update \( w^{(k)} = w^{(k-1)} - \alpha^{(k)}(2Gw - 2b) \) (2d\(^2\) + 3d flops)

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

Q: Is this a big data algorithm?
Computational considerations: gradient descent

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

to compute this quickly:

- form the **Gram matrix** \( G = X^\top X \) (2nd\(d^2\) flops)
- form \( b = X^\top y \) (2nd flops)
- for \( k = 1, \ldots \)
  - update \( w^{(k)} = w^{(k-1)} - \alpha^{(k)}(2Gw - 2b) \) (2\(d^2\) + 3\(d\) flops)

\(O(nd^2)\) 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

\[ \text{minimize} \quad \|y - Xw\|^2 \]

- solve by setting the gradient to 0: optimal \( w \) satisfies

\[
0 = \nabla_w \|y - Xw\|^2 \\
= -2X^\top y + 2X^\top Xw \\
X^\top Xw = X^\top y
\]

- \( X^\top X \) is called the **Gram matrix**
- \( X^\top Xw = X^\top y \) is called the **normal equations**
Convexity

► A function $f : \mathbb{R}^n \rightarrow \mathbb{R}$ is convex if it never lies above its chord: for all $\theta \in [0, 1]$,

$$f(\theta x + (1 - \theta)y) \leq \theta f(x) + (1 - \theta)f(y)$$

► A differentiable function $f : \mathbb{R}^n \rightarrow \mathbb{R}$ is convex if it satisfies the first order condition

$$f(y) - f(x) \geq \nabla f(x)^\top (y - x) \quad \forall x, y \in \mathbb{R}^n$$

► If a function is convex and differentiable, then

$\nabla f(x) = 0 \iff x$ minimizes $f$.

**proof:**
Convexity

- A function \( f : \mathbb{R}^n \rightarrow \mathbb{R} \) is convex if it never lies above its chord: for all \( \theta \in [0, 1] \),

\[
f(\theta x + (1 - \theta)y) \leq \theta f(x) + (1 - \theta)f(y)
\]

- A differentiable function \( f : \mathbb{R}^n \rightarrow \mathbb{R} \) is convex if it satisfies the first order condition

\[
f(y) - f(x) \geq \nabla f(x)^\top (y - x) \quad \forall x, y \in \mathbb{R}^n
\]

- If a function is convex and differentiable, then

\( \nabla f(x) = 0 \iff x \) minimizes \( f \).

**proof:** if \( \nabla f(x) \), then the first order condition says

\[
f(y) - f(x) \geq \nabla f(x)^\top (y - x) = 0 \quad \forall x, y \in \mathbb{R}^n
\]
Least squares objective is convex

we will prove the first order condition for convexity for least squares

\[ \|y - Xw'\|^2 - \|y - Xw\|^2 \geq (\nabla_w \|y - Xw\|^2)^\top (w' - w) \]

class any two models \( w \) and \( w' \). compute

\[
\begin{align*}
\|y - Xw'\|^2 - \|y - Xw\|^2 &= y^\top y - 2y^\top Xw' + w'^\top X^\top Xw' + y^\top y + 2y^\top Xw - w^\top X^\top Xw \\
&= -2y^\top X(w' - w) + w'^\top X^\top X(w' - w) + w^\top X^\top X(w' - w) \\
&= -2y^\top X(w' - w) + (w' - w)^\top X^\top X(w' - w) + 2w^\top X^\top X(w' - w) \\
&= -2y^\top X(w' - w) + \|X(w' - w)\|^2 + 2w^\top X^\top X(w' - w) \\
&\geq (-2y^\top X + 2w^\top X^\top X)(w' - w) \\
&= (\nabla_w \|y - Xw\|^2)^\top (w' - w)
\end{align*}
\]
\[ X^\top Xw = X^\top y \iff w \text{ is optimal} \]

using first order condition,

\[ \|y - Xw'\|^2 - \|y - Xw\|^2 \geq (\nabla_w \|y - Xw\|^2)^\top (w' - w) \]

- if \( \nabla_w \|y - Xw\|^2 = 0 \), then for any \( w' \),
  \[ \|y - Xw'\|^2 - \|y - Xw\|^2 \geq 0 \]

- so \( w \) minimizes \( \|y - Xw\|^2 \)!
- rewrite \( \nabla_w \|y - Xw\|^2 = 0 \) to get normal equations

\[ 0 = \nabla_w \|y - Xw\|^2 = -2X^\top y + 2X^\top Xw \]

\[ X^\top Xw = X^\top y \]
Pseudo-inverse

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

\[
X^\top Xw = X^\top y
\]

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

- $X^\dagger = (X^\top X)^{-1} X^\top$ is called the pseudo-inverse of $X$
  - $X^\dagger X = I_d$
  - but $XX^\dagger \neq I_n$
  - if $y \in \text{range}(X)$, then $y = XX^\dagger y$
The QR factorization

rewrite $X$ in terms of **QR decomposition** $X = QR$

- $Q \in \mathbb{R}^{n \times d}$ has orthogonal columns: $Q^\top Q = I_d$
- $R \in \mathbb{R}^{d \times d}$ is upper triangular: $R_{ij} = 0$ for $i > j$
- diagonal of $R \in \mathbb{R}^{d \times d}$ is positive: $R_{ii} > 0$ for $i = 1, \ldots, d$
- this factorization always exists and is unique
  (proof by Gram-Schmidt construction)

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

rewrite $X$ in terms of **QR decomposition** $X = QR$

▶ $Q \in \mathbb{R}^{n \times d}$ has orthogonal columns: $Q^\top Q = I_d$

▶ $R \in \mathbb{R}^{d \times d}$ is upper triangular: $R_{ij} = 0$ for $i > j$

▶ diagonal of $R \in \mathbb{R}^{d \times d}$ is positive: $R_{ii} > 0$ for $i = 1, \ldots, d$

▶ this factorization always exists and is unique (proof by Gram-Schmidt construction)

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

in julia (or matlab), use the QR function

$Q, R = qr(X)$
QR for least squares

use QR to solve least squares:

\[
\begin{align*}
  w &= (X^\top X)^{-1}X^\top y = (R^\top Q^\top QR)^{-1}R^\top Q^\top y \\
  &= (R^\top R)^{-1}R^\top Q^\top y \\
  &= R^{-1}(R^\top)^{-1}R^\top Q^\top y \\
  &= R^{-1}Q^\top y
\end{align*}
\]
never form the inverse explicitly: numerically unstable!

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

- compute $QR$ factorization of $X$ \hspace{1cm} (2nd^2$ flops$)
- to compute $w = X^\dagger y = R^{-1}Q^\top y$
  - form $b = Q^\top y$ \hspace{1cm} (2nd flops$)$
  - compute $w = R^{-1}b$ by back-substitution \hspace{1cm} ($d^2$ flops$)$
never form the inverse explicitly: numerically unstable!

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

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

in julia (or matlab), the \textbf{backslash operator} solves least-squares efficiently (usually, using QR)

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

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

- QR factorization: https://en.wikipedia.org/wiki/QR_decomposition