Linear Methods for Regression: Introduction

-Statistical Machine Learning-

Lecturer: Darren Homrighausen, PhD

The Setup

Suppose we have data

$$\mathcal{D} = \{(X_1, Y_1), (X_2, Y_2), \ldots, (X_n, Y_n)\},$$

where

- $X_i \in \mathbb{R}^p$ are the features

  (or explanatory variables or predictors or covariates. NOT INDEPENDENT VARIABLES!)
- $Y_i \in \mathbb{R}$ are the supervisor variables.

  (NOT DEPENDENT VARIABLE!)

Our goal for this class is to find a way to explain (at least approximately) the relationship between X and Y.

Prediction risk for regression

Given the training data $\mathcal{D}$, we want to predict some independent test data $Z = (X, Y) \sim \mathbb{P} \equiv \mathbb{P}_Z$

This means forming a $\hat{f}$, which is a function of both the range of X and the training data $\mathcal{D}$, which provides predictions $\hat{Y} = \hat{f}(X)$.

The quality of this prediction is measured via the prediction risk[1]

$$R(\hat{f}) = \mathbb{P}(Y - \hat{f}(X))^2 = \mathbb{P}\ell_{\hat{f}}$$

We know that the regression function, $f_*(X) = \mathbb{P}[Y|X]$, is the best possible predictor.

Note that f_* is *unknown*

[1] Sometimes we integrate with respect to $\mathcal{D}$ only, Z only, neither (loss), or both.

Notation recap

- X is a vector of measurements for each subject
  (Example: $X_i = [1, \text{income}_i, \text{education}_i]^\top$)
- x is a vector of subjects for each measurement
  (Example: $x_j = [\text{income}_1, \text{income}_2, \ldots, \text{income}_n]^\top$)
- X_{ij} is the j^{th} measurement on the i^{th} subject
  (Example: $X_{ij} = \text{income}_i$)

Notational landmine: representing the j^{th} entry of X

$\rightarrow$ A reasonable, but technically sloppy, solution: X_j

Imposing linearity

A linear model: Multiple regression

If we specify the model: $f_*(X) = X^\top \beta = \sum_{j=1}^p X_j \beta_j$

$$\Rightarrow Y_i = X_i^\top \beta + \epsilon_i$$

Then we recover the usual linear regression formulation

$$\mathbb{X} = \begin{bmatrix} x_1 & \cdots & x_p \end{bmatrix} = \begin{bmatrix} X_1^\top \\ X_2^\top \\ \vdots \\ X_n^\top \end{bmatrix} \in \mathbb{R}^{n \times p}$$

Commonly, a column $x_0^\top = \underbrace{(1, \ldots, 1)}_{n \text{ times}}$ is included

This encodes an intercept term, with intercept parameter β_0

We could (should?) seek to find a β such that $Y \approx \mathbb{X}\beta$

A linear model: Polynomial effects

Instead, we may believe

$$f_*(X) = \beta_0 + \sum_{j=1}^{p} X_j \beta_j + \sum_{j=1}^{p} \sum_{j'=1}^{p} X_j X_{j'} \alpha_{j,j'}$$

Then the feature matrix is

$$\mathbb{X} = \left[\begin{array}{cccccccc} x_0 & x_1 & \cdots & x_p & x_1^2 & x_1 x_2 & \cdots & x_p^2 \end{array} \right]$$

(Here, interpret vector multiplication in the entrywise sense, as in R: `x * y`)

A linear model: General form

Specify functions $\phi_k : \mathbb{R}^p \to \mathbb{R}$, $k = 1, \ldots, K$

$$\mathbb{X} = [\phi_k(X_i)] = \begin{bmatrix} \Phi(X_1)^\top \\ \Phi(X_2)^\top \\ \vdots \\ \Phi(X_n)^\top \end{bmatrix} \in \mathbb{R}^{n \times K},$$

where $\Phi(\cdot)^\top = (\phi_1(\cdot), \ldots, \phi_K(\cdot))$.

Example:

$$\phi_k(X) = X_j X_{j'}$$

is an interaction for the j^{th} and j'^{th} features

In this case $K = \binom{p}{2} + p = p(p-1)/2 + p = (p^2 + p)/2$

A linear model: Multiple regression redux

Let $K = p$ and define ϕ_k to be the coordinate projection map

That is,

$$\phi_k(X_i) \equiv X_{ik}$$

We recover the usual linear regression formulation

$$\mathbb{X} = [\phi_k(X_i)] = \begin{bmatrix} \Phi(X_1)^\top \\ \Phi(X_2)^\top \\ \vdots \\ \Phi(X_n)^\top \end{bmatrix} = \begin{bmatrix} X_{11} & X_{12} & \cdots & X_{1p} \\ X_{21} & X_{22} & \cdots & X_{2p} \\ \vdots & & & \\ X_{n1} & X_{n2} & \cdots & X_{np} \end{bmatrix} = \begin{bmatrix} X_1^\top \\ X_2^\top \\ \vdots \\ X_n^\top \end{bmatrix}$$

A linear model: General form

We don't know if f_* can actually be expressed as a linear function

Hence, write

$$\mathcal{F}_K = \{f : \exists(\beta_k)_{k=1}^K \text{ such that } f = \sum_{k=1}^K \beta_k \phi_k = \beta^\top \Phi\}$$

and

$$f_{*,K} = \operatorname*{argmin}_{f \in \mathcal{F}_K} \mathbb{P}\ell_f.$$

The function $f_{*,K}$ is known as the linear oracle

This is the object we are estimating when using a linear model

Alternatively, we can assume $f_* \in \mathcal{F}_K$

A linear model: Orthogonal basis expansion

Suppose $f_* \in \mathcal{F}$, where $\mathcal{F}$ is a Hilbert space with norm induced by the inner product $\langle \cdot, \cdot \rangle$.

Let $(\phi_k)_{k=1}^{\infty}$ be an orthonormal basis for $\mathcal{F}$

Write

$$f_* = \sum_{k=1}^{\infty} \langle f_*, \phi_k \rangle \phi_k = \sum_{k=1}^{\infty} \beta_k \phi_k = \beta^\top \Phi$$

Then we can estimate $f_{*,K}$ by finding the coefficients of the projection on $\mathcal{F}_K$.

By Parseval's theorem for Hilbert spaces this induces an approximation error of $\sum_{k=K+1}^{\infty} |\beta_k|^2$.

This is small if f_* is smooth

(for instance, if f_* has m derivatives, then $\sum_{k=1}^{\infty} k^{2m} |\beta_k|^2 < \infty$)

A linear model: Neural Nets

Let

$$\phi_k(X) = \sigma(\alpha_k^\top X + b_k),$$

where $\sigma(t) = 1/(1 + e^{-t})$ is the sigmoid activation function.

Then we can form the feature matrix

$$\mathbb{X} = \begin{bmatrix} \phi_1(X_1) & \phi_2(X_1) & \cdots \\ & \vdots & \\ \phi_1(X_n) & \phi_2(X_n) & \cdots \end{bmatrix}$$

For future reference, this is a

"single-layer feed-forward neural network model with linear output"

(It is actually a bit more complicated, as the parameters in the σ map are estimated, and hence this is actually nonlinear)

A linear model: Radial basis functions

Let

$$\phi_k(X) = e^{-||\mu_k - X||_2^2/\lambda_k}.$$

Then $f_{*,K}$ is called an[2]:

"Gaussian radial-basis function estimator".

This turns out to be a parametric form of a more general technique known as Gaussian process regression.

[2]More on this later

Detour

Notation comment

WARNING: It is common to conflate:

- the number of original covariates (p)
- the number of created features (K)

This means we will tend to write $\mathbb{X} \in \mathbb{R}^{n \times p}$, regardless of the transformation Φ that generates the matrix $\mathbb{X}$

The reasons for this are

- multiple regression comes from a particular, degenerate choice of Φ
- the mapping Φ is often not explicitly created (and $K = \infty$)

Bottom line: Think of X as the vector after transformations and $\mathbb{X} \in \mathbb{R}^{n \times p}$ regardless of the choice of Φ

End detour

Turning these ideas into procedures

Each of these methods have parameters to choose:

- p could be very large. Do we include all features?
- If we include some polynomial (or other function) terms, should be include all of them?
- For neural nets, we need to choose: the activation function σ, the directions α_k, bias terms b_k, as well as the number of units in the hidden layer

Additionally, we need to estimate the associated coefficient vector β, α, or whatever

We would like the data to inform these parameters

Training error and risk estimation

The linear oracle is defined to be

$$f_{*,K} = \underset{f \in \mathcal{F}_K}{\operatorname{argmin}} \, \mathbb{P}\ell_f$$

(Reminder: for regression, $\ell_f(Z) = (f(X) - Y)^2$)

Hence, it is intuitive to use $\hat{\mathbb{P}}$ to form the training error

$$\hat{R}(f) = \hat{\mathbb{P}}\ell_f = \frac{1}{n}\sum_{i=1}^{n} \ell_f(Z_i) = \frac{1}{n}\sum_{i=1}^{n}(f(X_i) - Y_i)^2 = \frac{1}{n}\,||Y - \mathbb{X}\beta||_2^2$$

In many statistical applications, this plug-in estimator is minimized

(Think of how many techniques rely on an unconstrained minimization of squared error, or maximum likelihood, or estimating equations, or ...)

This sometimes has disastrous results

Example

Let's suppose $\mathcal{D}$ is drawn from

```
n = 30
X = (0:n)/n*2*pi
Y = sin(X) + rnorm(n,0,.25)
```

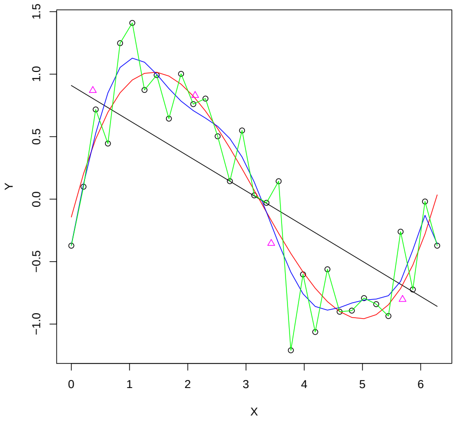
Now, let's fit some polynomials to this data.

We consider the following models:

- Model 1: $f(X_i) = \beta_0 + \beta_1 X_i$
- Model 2: $f(X_i) = \beta_0 + \beta_1 X_i + \beta_2 X_i^2 + \beta_3 X_i^3$
- Model 3: $f(X_i) = \sum_{k=0}^{10} \beta_k X_i^k$
- Model 4: $f(X_i) = \sum_{k=0}^{n-1} \beta_k X_i^k$

Let's look at what happens...

Example



The $\hat{R}$'s are:

$$\hat{R}(\text{Model 1}) = 10.98$$
$$\hat{R}(\text{Model 2}) = 2.86$$
$$\hat{R}(\text{Model 3}) = 2.28$$
$$\hat{R}(\text{Model 4}) = 0$$

What about predicting new observations (Δ)?

Bias and variance

Prediction risk for regression

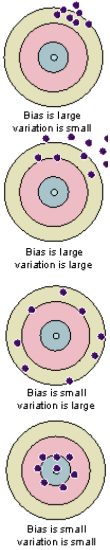
Note that $R(\hat{f})$ can be written as

$$R(\hat{f}) = \int \mathrm{bias}^2(X) d\mathbb{P}_X + \int \mathrm{var}(X) d\mathbb{P}_X + \sigma^2$$

where

$$\begin{aligned} \mathrm{bias}(X) &= \mathbb{P}\hat{f}(X) - f_*(X) \\ \mathrm{var}(X) &= \mathbb{V}\hat{f}(X) \\ \sigma^2 &= \mathbb{P}(Y - f_*(X))^2 \end{aligned}$$

(As an aside, this decomposition applies to much more general loss functions[a])



[a] *Variance and Bias for General Loss Functions*; , Machine Learning 2003♯

Bias-variance tradeoff

This can be heuristically thought of as

$$\text{Prediction risk} = \text{Bias}^2 + \text{Variance}.$$

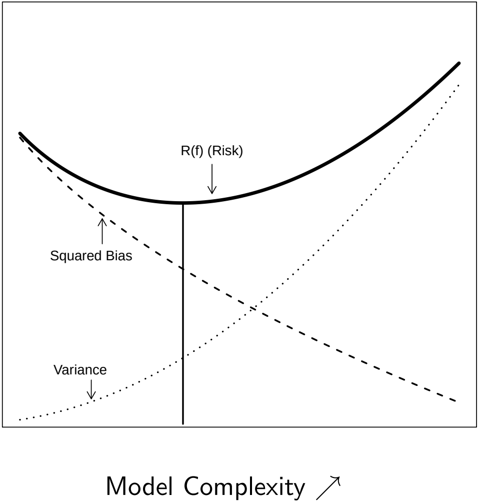
There is a natural conservation between these quantities

Low bias $\rightarrow$ complex model $\rightarrow$ many parameters $\rightarrow$ high variance

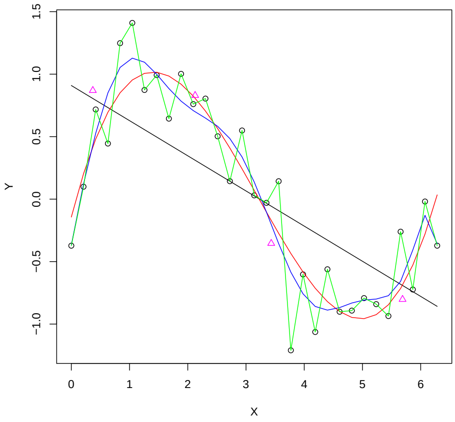
The opposite also holds
(Think: $\hat{f} \equiv 0$.)

We'd like to 'balance' these quantities to get the best possible predictions

Bias-variance tradeoff



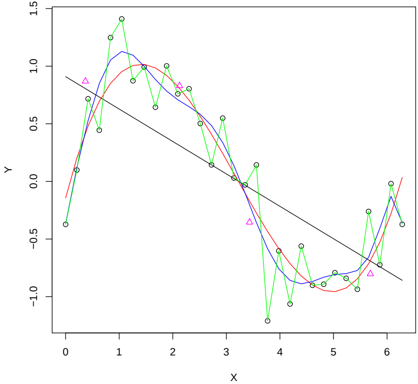
Example

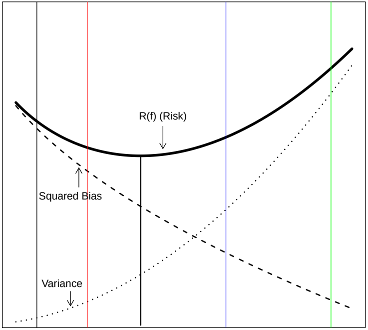


- Black model has low variance, high bias
- Green model has low bias, but high variance
- Red model and Blue model have intermediate bias and variance.

We want to balance these two quantities.

Bias vs. Variance

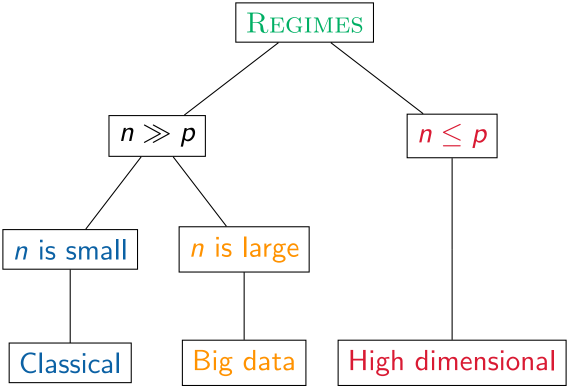




Model Complexity ↗

Turning these ideas into procedures

There are roughly three regimes of interest, assuming $\mathbb{X} \in \mathbb{R}^{n \times p}$



Classical regime

Suppose we have the matrix $\mathbb{X}$ with the features we're considering

Now, we want to estimate a parameter vector β in the model

$$Y = \mathbb{X}\beta + \epsilon$$

(E.g. we are modeling the regression function as (globally) linear in these features)

Minimize the training error $\hat{R}(f)$ over all functions $f_\beta(X) = X^\top \beta$

$$\hat{\beta}_{LS} = \underset{\beta}{\operatorname{argmin}}\, \hat{R}(f_\beta) = \underset{\beta}{\operatorname{argmin}}\, ||Y - \mathbb{X}\beta||_2^2$$

(Though we write this as equality, there is only a unique solution if $\operatorname{rank}(\mathbb{X}) = p$)

Classical regime

In this case,

$$\hat{f}(X) = X^\top \hat{\beta}_{LS} = X^\top \mathbb{X}^\dagger Y \underbrace{=}_{\mathrm{rank}(\mathbb{X})=p} X^\top (\mathbb{X}^\top \mathbb{X})^{-1} \mathbb{X}^\top Y$$

($\mathbb{X}^\dagger$ is the Moore-Penrose pseudo inverse$^\sharp$)

The fitted values are $\mathbb{X}\hat{\beta}_{LS} = HY$, where H is the orthogonal projection onto the column space of $\mathbb{X}$

(Contrary to $\hat{\beta}_{LS}$, the fitted values are always unique)

Classical regime

We can examine the first and second moment properties of $\hat{\beta}_{LS}$

$$\mathbb{E}\hat{\beta}_{LS} = \beta \qquad \text{(unbiased)} \tag{1}$$

$$\mathbb{V}\hat{\beta}_{LS} = \mathbb{X}^{\dagger}(\mathbb{V}Y)(\mathbb{X}^{\dagger})^{\top} \underbrace{=}_{\text{rank}(\mathbb{X})=p, \mathbb{V}Y \propto I_n} \mathbb{V}[Y_i](\mathbb{X}^{\top}\mathbb{X})^{-1} \tag{2}$$

NOTE: Here is where we need to be more careful

The 'true' parameter β we are estimating is a coefficient vector of the linear oracle with respect to

$$\{f : \text{ There exists } \beta \text{ where } f(X) = \beta^{\top}X\}$$

There is no reason to believe this approximation error is zero, hence 'bias' really references the linear oracle

Classical regime

The Gauss-Markov theorem assures us that this is the best linear unbiased estimator of β

(Effectively, equation (2) is minimized subject to equation (1))

Also, it is the maximum likelihood estimator under a homoskedastic, independent Gaussian model

(Hence, it is asymptotically efficient)

Does that necessarily mean it is any good?

Classical regime

Write $\mathbb{X} = UDV^\top$ for the SVD of $\mathbb{X}$

Then $\mathbb{V}\hat{\beta}_{LS} \propto (\mathbb{X}^\top\mathbb{X})^{-1} = (VD\underbrace{U^\top U}_{=I}DV^\top)^{-1} = VD^{-2}V^\top$

Reminder: Elements of D, d_j, are the axes lengths of the ellipse induced by $\mathbb{X}$

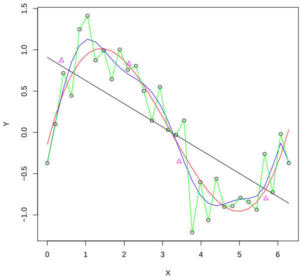
Also, suppose we are interested in estimating β,

$$\mathbb{E}||\hat{\beta}_{LS} - \beta||_2^2 = \text{trace}(\mathbb{V}\hat{\beta}) \propto \sum_{j=1}^{p} \frac{1}{d_j^2}$$

(Can you show this? Hint: add and subtract $\mathbb{E}\hat{\beta}_{LS}$)$^\sharp$

Important: Even in the classical regime, we can do arbitrarily badly if $d_p \approx 0$!

Returning to polynomial example: Bias



Using a Taylor's series, for all X

$$\sin(X) = \sum_{q=0}^{\infty} \frac{(-1)^q X^{2q+1}}{(2q+1)!} = \Phi(X)^\top \beta$$

Higher order polynomial models will reduce the bias part

Returning to polynomial example: Variance

The least squares solution is given by solving $\min ||\mathbb{X}\beta - Y||_2^2$

$$\mathbb{X} = \begin{bmatrix} 1 & X_1 & \dots & X_1^{p-1} \\ & \vdots & & \\ 1 & X_n & \dots & X_n^{p-1} \end{bmatrix},$$

is the associated Vandermonde$^\sharp$ matrix.

This matrix is well known for being numerically unstable

(Letting $\mathbb{X} = UDV^\top$, this means that $d_1/d_p \to \infty$)

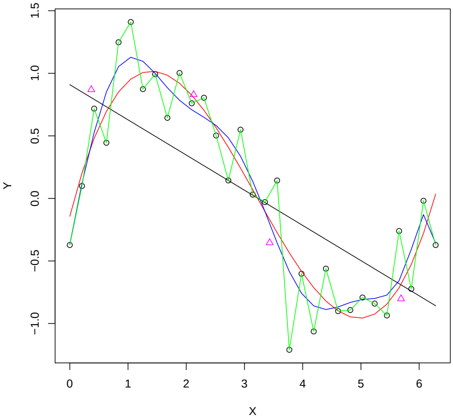
Hence[3]

$$||(\mathbb{X}^\top \mathbb{X})^{-1}||_2 = \frac{1}{d_p^2}$$

grows larger, where here $||\cdot||_2$ is the spectral (operator) norm$^\sharp$

[3] This should be compared with the variance computation in equation (2)

Returning to the polynomial example



Conclusion

Conclusion: Fitting the full least squares model, even in the classical regime, can lead to poor prediction/estimation performance

In the other regimes, we encounter even more sinister problems

Big data regime

Big data: The computational complexity scales extremely quickly. This means that procedures that are feasible classically are not for large data sets

Example: Fit $\hat{\beta}_{LS}$ with $\mathbb{X} \in \mathbb{R}^{n \times p}$. Next fit $\hat{\beta}_{LS}$ with $\mathbb{X} \in \mathbb{R}^{3n \times 4p}$

The second case will take $\approx (3 * 4^2) = 48$ times longer to compute, as well as ≈ 12 times as much memory!

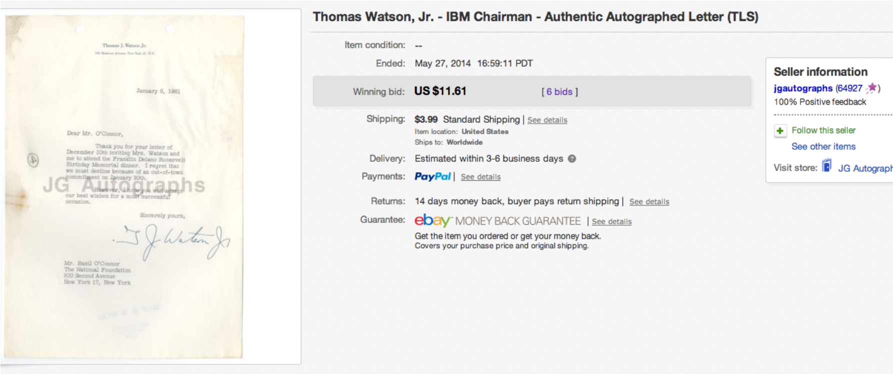
(Actually, for software such as R it might take 36 times as much memory, though there are data structures specifically engineered for this purpose that update objects 'in place')

Conclusion

```
p = 300; n = 10000
Y = rnorm(n); X = matrix(rnorm(n*p),nrow=n,ncol=p)
start = proc.time()[3]
out   = lm(Y~.,data=data.frame(X))
end   = proc.time()[3]
smallTime = end - start

n = nMultiple*n; nMultiple = 3
p = pMultiple*p; pMultiple = 4
Y = rnorm(n); X = matrix(rnorm(n*p),nrow=n,ncol=p)
start = proc.time()[3]
out   = lm(Y~.,data=data.frame(X))
end   = proc.time()[3]
bigTime = end - start
> print(bigTime/smallTime)
 elapsed
38.61458
> print(nMultiple*pMultiple**2)
[1] 48
```

Example big data problem



Example big data problem

Buyer:

Always a pleasure! Smooth & pleasant transaction!	f***a (3618 ★)	Jun-10-14 13:52
Thomas Watson, Jr. - IBM Chairman - Authentic Autographed Letter (TLS) (#390846670600)	US $11.61	View Item

Seller:

Great communication. A pleasure to do business with.	Buyer: f***a (3618 ★)	Jun-05-14 18:59
Thomas Watson, Jr. - IBM Chairman - Authentic Autographed Letter (TLS) (#390846670600)	--	View Item

The data (∼750 Gb, millions of rows, thousands of columns):

User	ID	Rating	Comment	Role	WinBid	SellerID
dorkyporky	134	1	fast delivery.....very good seller...AAA++	B	15.51	princesskitten2001

Treatment in practice

Depending on the data and the desired method, we could:

- Combine randomized projections together with in-memory procedures
- Use stochastic gradient descent (approximate)
- Leverage an iterative implementation for exact computation
  (An example is the QR decomposition for least squares)
- Break the computations down into small bits and distribute these to different cores/processors/nodes

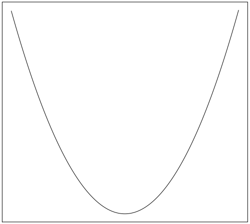
High dimensional regime

High dimensional: These problems tend to have many of the computational problems as Big data, as well as a rank problem:

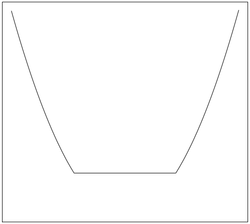
Suppose $\mathbb{X} \in \mathbb{R}^{n \times p}$ and $p > n$

Then $\mathrm{rank}(\mathbb{X}) = n$ and the equation $\mathbb{X}\hat{\beta} = Y$:

- can be solved *exactly* (that is; the training error is 0)
- has an infinite number of solutions



$n > p$



$n < p$

High dimensional regime: Examples