CS M148 –

Data Science Fundamentals

Lecture #7: Regularization, Classification

Baharan Mirzasoleiman

UCLA Computer Science

Announcements

HW 1

- Deadline extended: due Fri Jan 28, 11pm

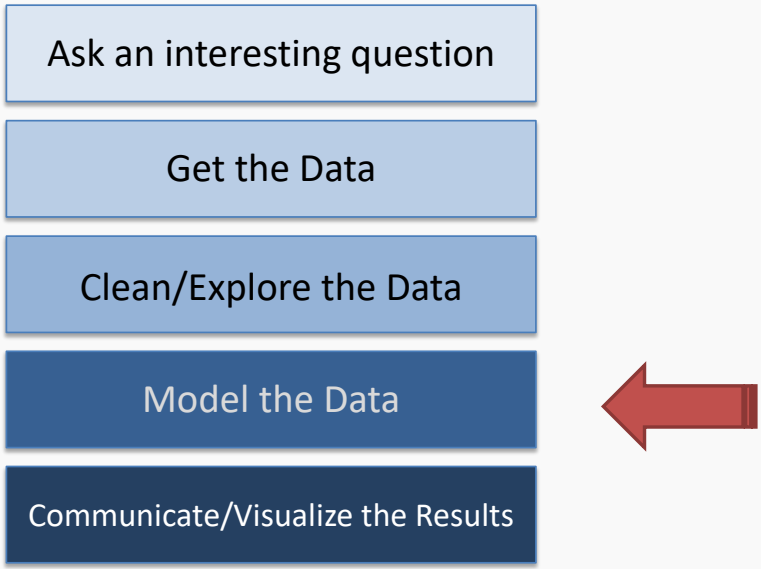
Project 2 is posted

- Due Wed Feb 9, 2pm

Let’s quickly review what we saw last time

Still here!

The Data Science Process



Inference in Linear Regression

Uncertainty in estimating the linear regression coefficients

How reliable are the model interpretation

Suppose our model for advertising is:

$$y = 1.01x + 120$$

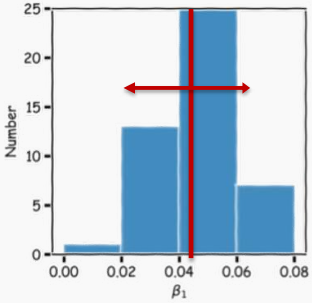
Where y is the sales in 1000$, x is the TV budget.

Interpretation: for every dollar invested in advertising, you get 1.01 back in sales, which is 1% net increase.

But how certain are we in our estimation of the coefficient 1.01?

Why aren't we certain?

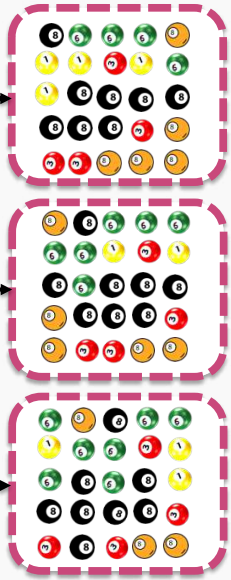
Bootstrap



Size N

Dataset

Sample with replacement



train

Model 1: $\hat{y} = \hat{\beta}_0^{(1)} + \hat{\beta}_1^{(1)} x$

train

Model 2: $\hat{y} = \hat{\beta}_0^{(2)} + \hat{\beta}_1^{(2)} x$

train

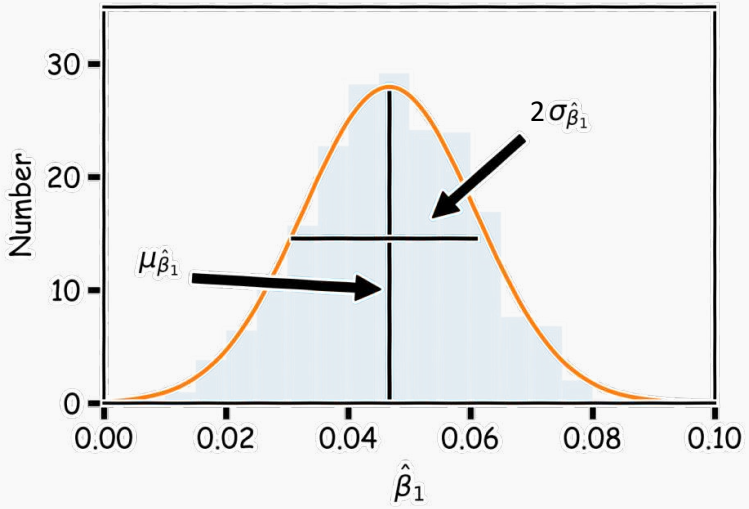
Model s: $\hat{y} = \hat{\beta}_0^{(s)} + \hat{\beta}_1^{(s)} x$

$$\mu_{\hat{\beta}} = \frac{1}{s}\sum_{i=1}^{s} \hat{\beta}^{(i)}$$

$$\sigma_{\hat{\beta}} = \sqrt{\frac{1}{s}\sum_{i=1}^{s} \left(\hat{\beta}^{(i)} - \bar{\beta}\right)^2}$$

Confidence intervals for the predictors estimates (cont)

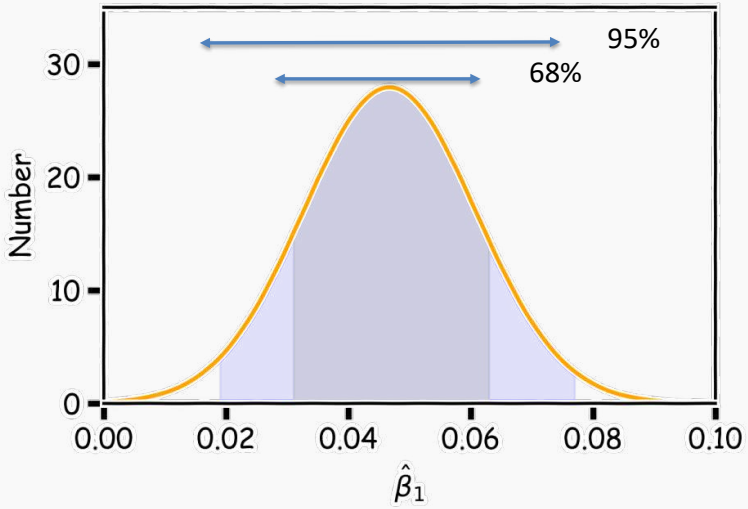
We can now estimate the mean and standard deviation of the estimates of $\hat{\beta}_0, \hat{\beta}_1$.



Confidence intervals for the predictors estimates (cont)

The standard errors give us a sense of our uncertainty over our estimates.

Typically we express this uncertainty as a 95% confidence interval, which is the range of values such that the **true** value of β_1 is contained in this interval with *95%* percent probability.



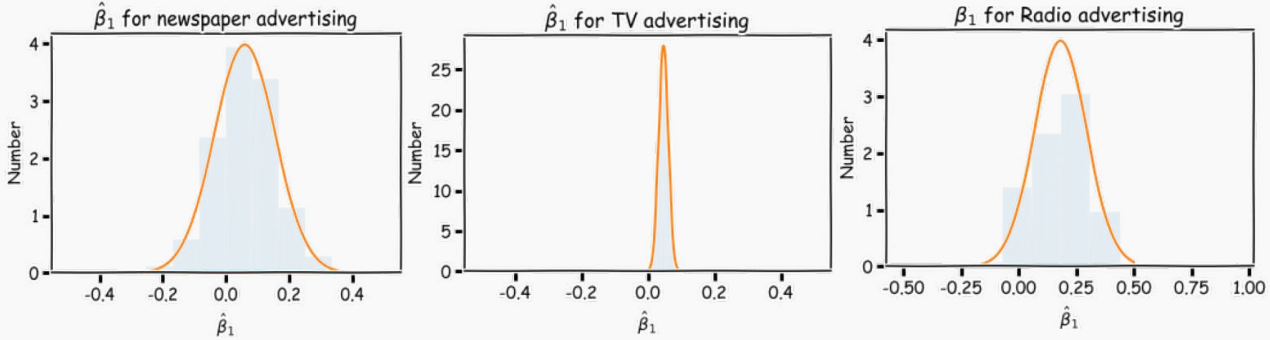
$$CI_{\hat{\beta}} = (\hat{\beta} - 2\sigma_{\hat{\beta}}, \hat{\beta} + 2\sigma_{\hat{\beta}})$$

Estimating Significance of Predictors
Hypothesis testing

Feature importance

Now we know how to generate these distributions we are ready to answer ***two important questions:***

A. Which predictors are most important?
B. And which of them really affect the outcome?



$\mu_{\beta_1} = 0.03$
σ_{β_1}=0.13

$\mu_{\beta_1} = 0.033$
σ_{β_1}=0.01

$\mu_{\beta_1} = 0.23$
σ_{β_1}=0.25

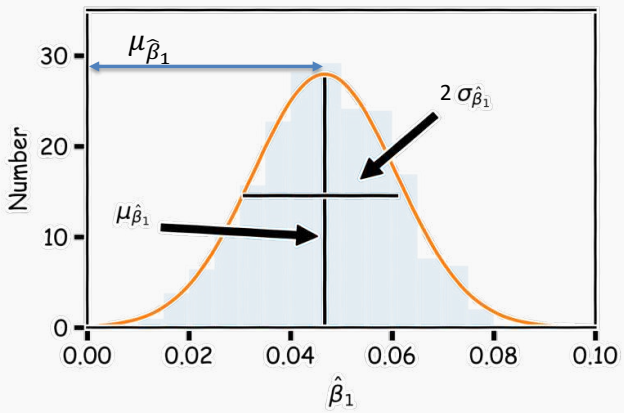
Feature Importance

To incorporate the coefficients' uncertainty, we need to determine whether the estimates of $\beta's$ are sufficiently far from zero.

To do so, we define a new metric, which we call t-test statistic:

$$t = \frac{\mu_{\hat{\beta}_1}}{\sigma_{\hat{\beta}_1}}$$

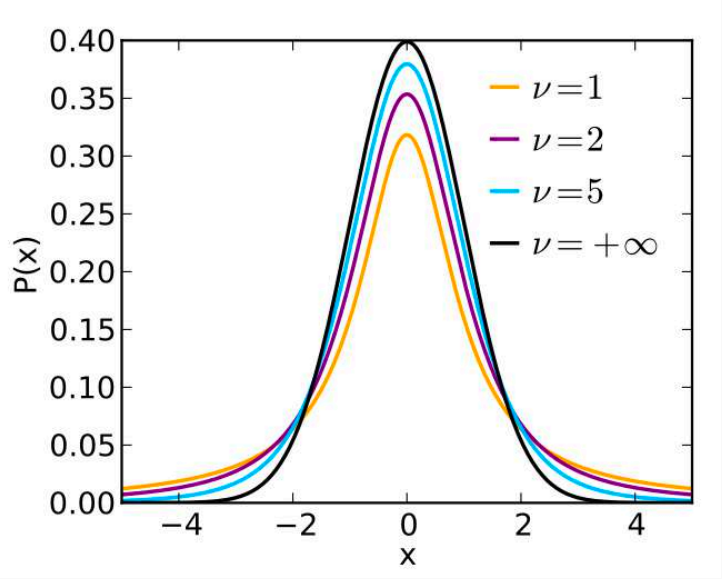
which measures the distance from zero in units of standard deviation.



1. For n random datasets fit n models.
2. Generate distributions for all predictors and calculate the means and standard errors $(\mu_{\hat{\beta}}, \sigma_{\hat{\beta}})$.
3. Calculate the t-tests.

Repeat and create a probability density function (pdf) for all the t-tests.

It turns out we do not have to do this, because this is a known distribution called student-t distribution.



Student-t distribution, where ν is the degrees of freedom (number of data points minus number of predictors).

To learn more about why student-t, what are degrees of freedom and more details see https://en.wikipedia.org/wiki/Student%27s_t-test

P-value

To compare the t-test values of the predictors from our model, $|t^*|$, with the t-tests, calculated using random data, $|t^R|$, we estimate the probability of observing $|t^R| \geq |t^*|$.

We call this probability the p-value.

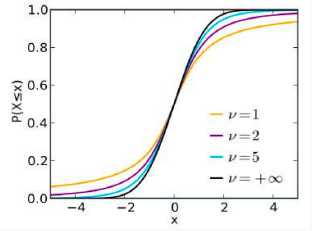
$$p - value = P(|t^R| \geq |t^*|)$$

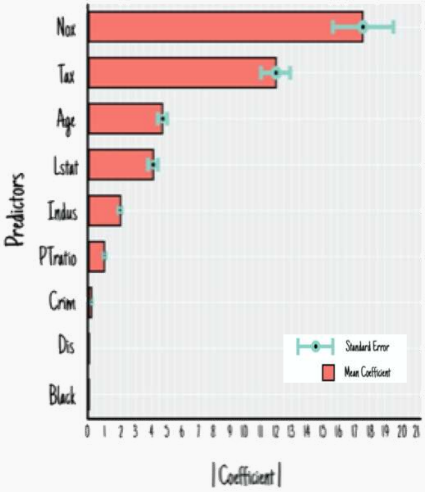
small p-value indicates that it is unlikely to observe such a substantial association between the predictor and the response due to chance.

It is common to use p-value<0.05 as the threshold for significance.

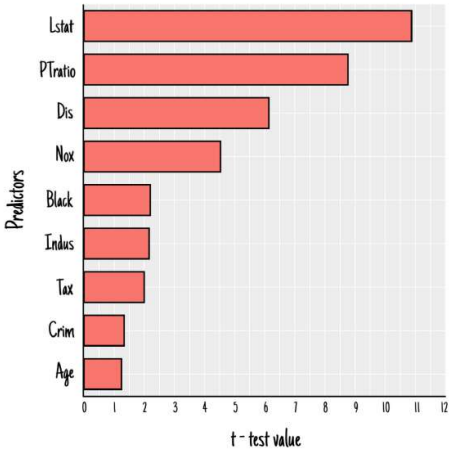
To calculate the p-value we use the cumulative distribution function (CDF) of the student-t.

`stats model` a python library has a build-in function `stats.t.cdf()` which can be used to calculate this.

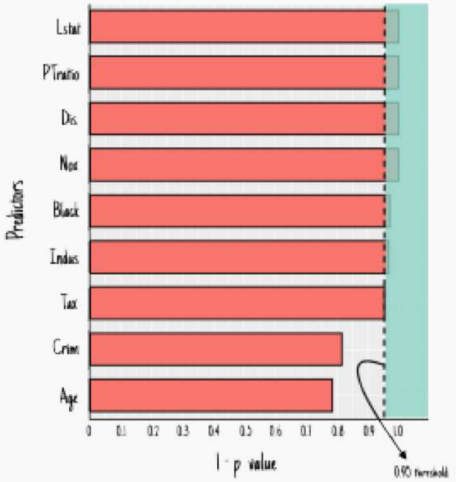




Feature importance based on the absolute value of the coefficients over multiple bootstraps and includes the coefficients' uncertainty.



Feature importance based on t-test. Notice the rank of the importance has changed.



Feature importance using p-value.

Hypothesis testing

1. State Hypothesis:

Null hypothesis:

H_0: There is no relation between *X* and *Y*

The alternative:

H_a: There is some relation between *X* and *Y*

2. **Choose test statistics**

t-test

3. Sample:

Using bootstrap we can estimate $\hat{\beta}_1's$, and $\mu_{\hat{\beta}_1}$ and $\sigma_{\hat{\beta}_1}$ and the t-test.

Hypothesis testing

4. **Reject or not reject the hypothesis:**

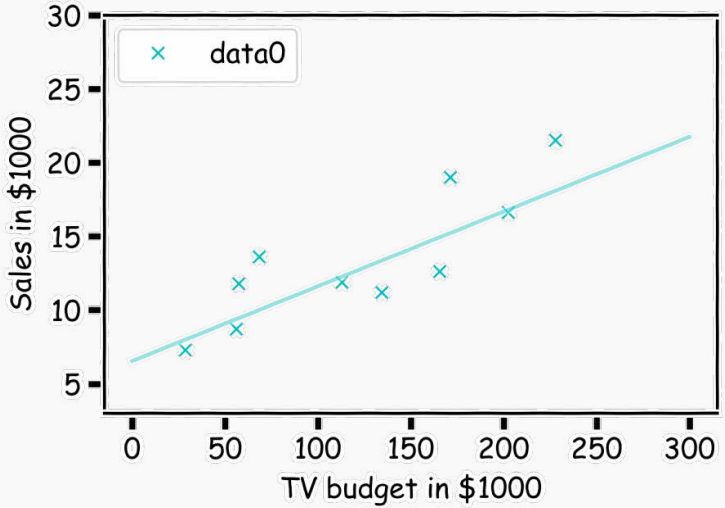
   We compute ***p-value ,*** the probability of observing any value equal to $|t|$ or larger, from random data.

   p-value < p-value-threshold we reject the null.

Prediction Intervals

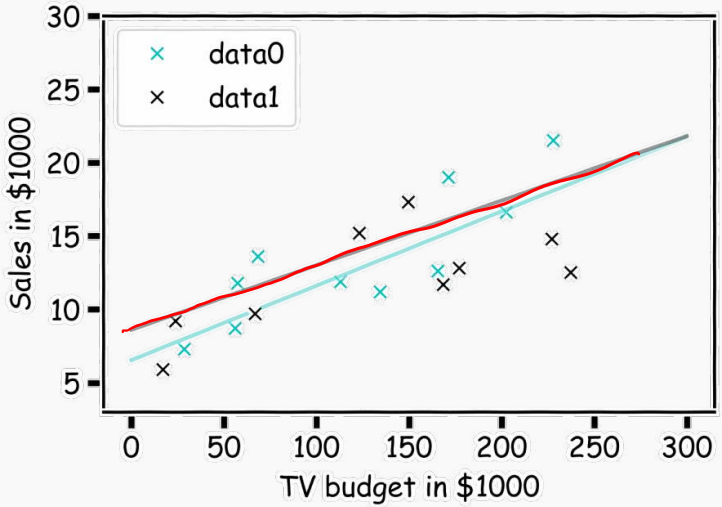
How well do we know $\hat{f}$?

Our confidence in f is directly connected with our confidence in βs. For each bootstrap sample, we have one β, which we can use to determine the model, $f(x) = X\beta$.



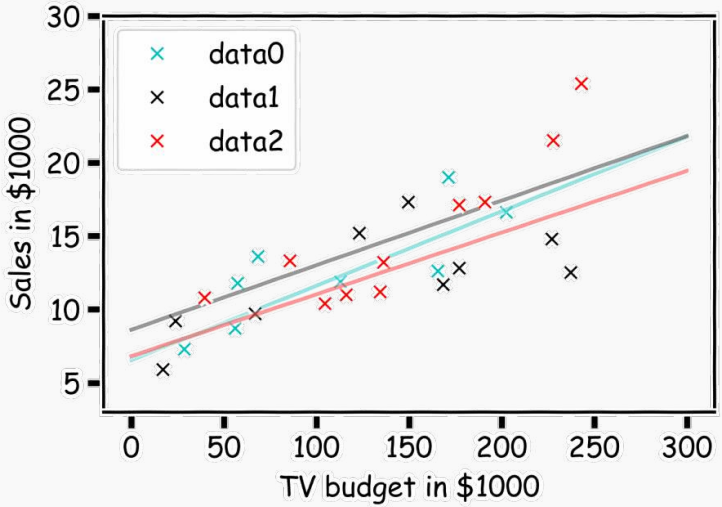
How well do we know $\hat{f}$?

Here we show two difference models predictions given the fitted coefficients.



How well do we know $\hat{f}$?

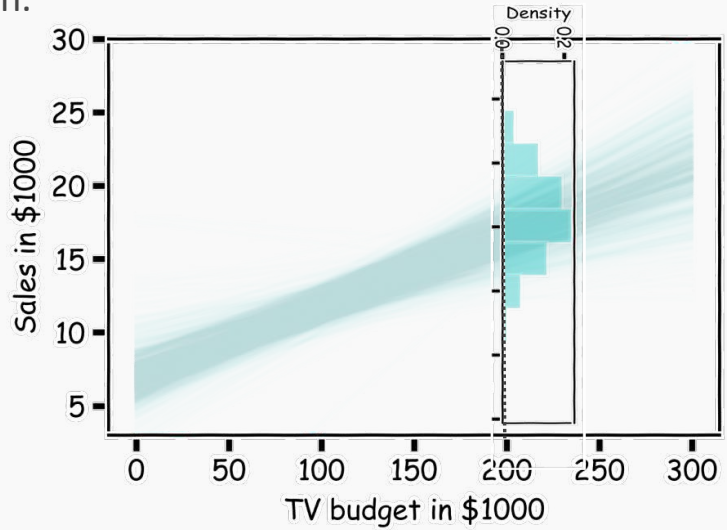
There is one such regression line for every bootstrapped sample.



How well do we know $\hat{f}$?

Below we show all regression lines for a thousand of such bootstrapped samples.

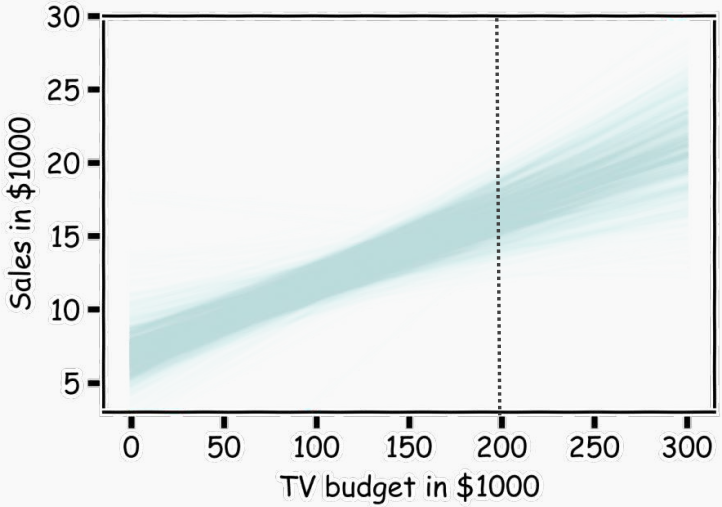
For a given x, we examine the distribution of $\hat{f}$, and determine the mean and standard deviation.

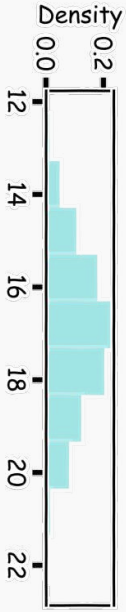


How well do we know $\hat{f}$?

Below we show all regression lines for a thousand of such bootstrapped samples.

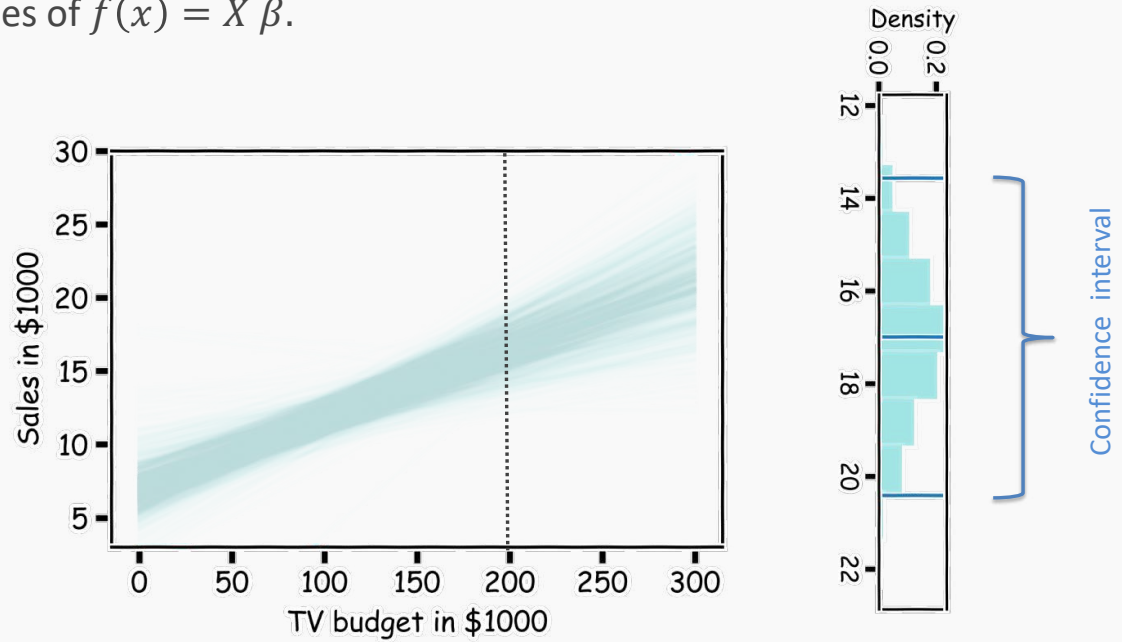
For a given x, we examine the distribution of $\hat{f}$, and determine the mean and standard deviation.





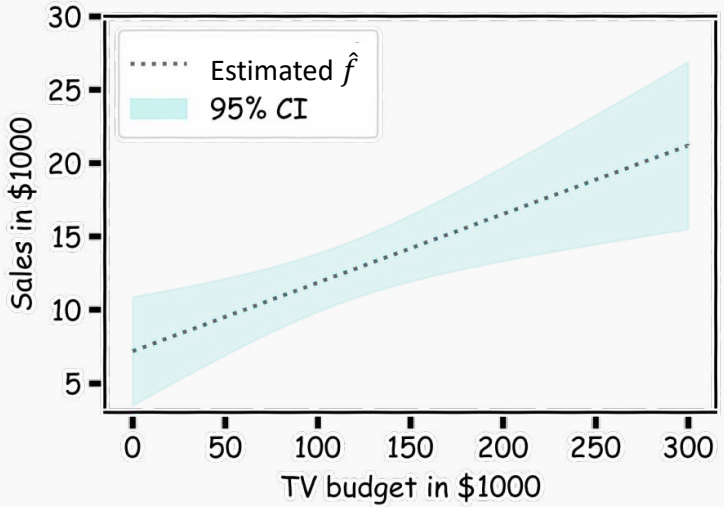
How well do we know $\hat{f}$?

We determine the confidence interval of $\hat{f}$ by selecting the region that contains 95% of the samples of $\hat{f}(x) = X\,\hat{\beta}$.



How well do we know $\hat{f}$?

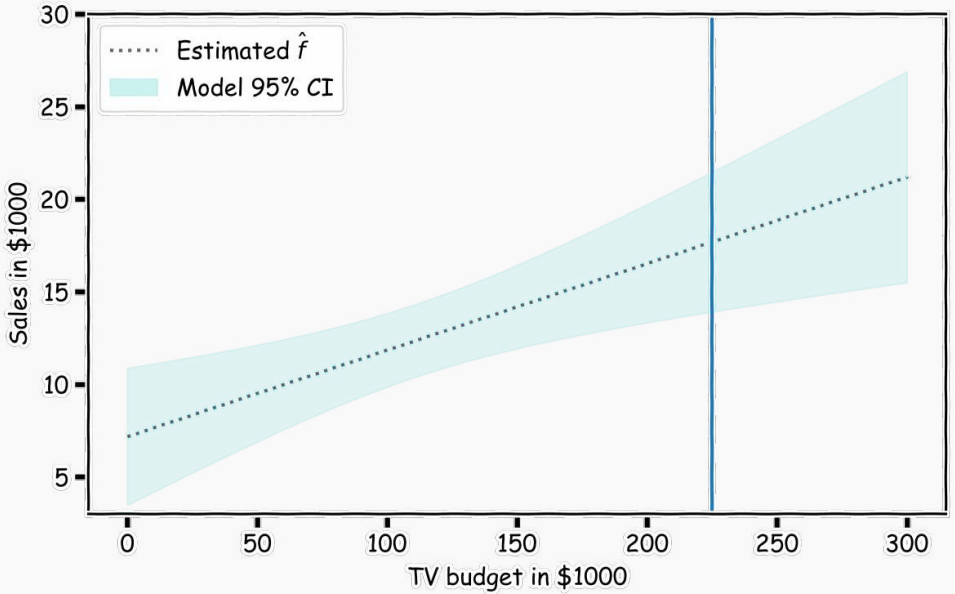
For every x, we calculate the mean of the models, $\widehat{\mu_f}$ (shown with dotted line) and the 95% CI of those models (shaded area).



Confidence in predicting $\hat{y}$

Even if we knew $f(x)$ —the response value cannot be predicted perfectly because of the random error in the model (irreducible error).

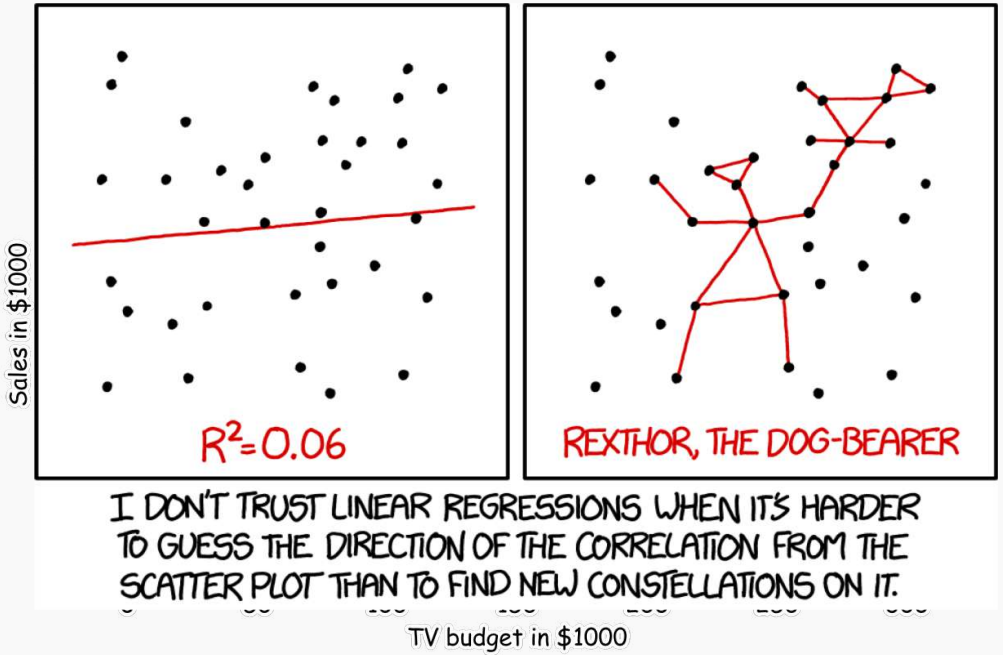
How much will Y vary from $\hat{Y}$? We use prediction intervals to answer this question.



Confidence in predicting $\hat{y}$

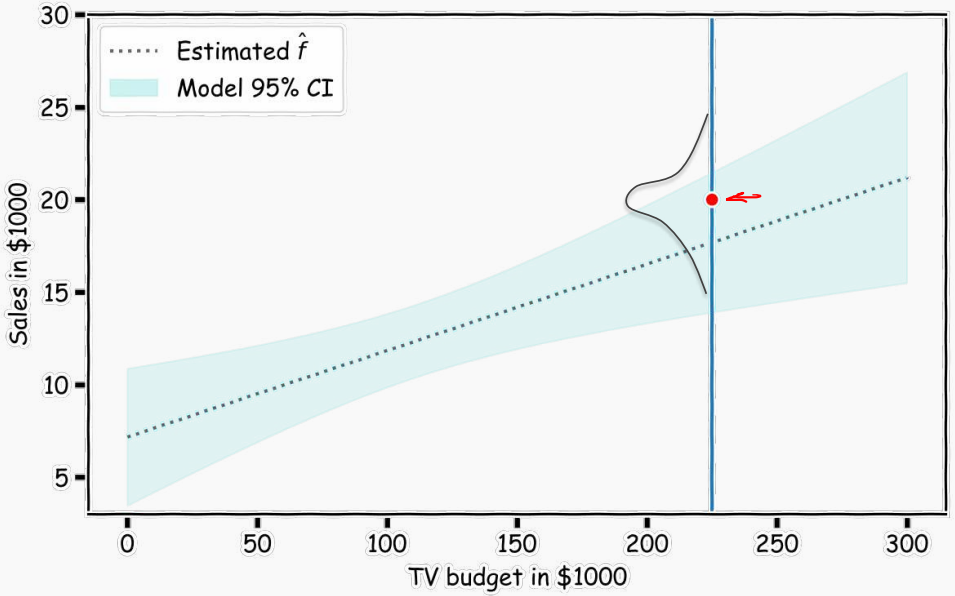
Even if we knew $f(x)$ —the response value cannot be predicted perfectly because of the random error in the model (irreducible error).

How much will Y vary from $\hat{Y}$? We use prediction intervals to answer this question.



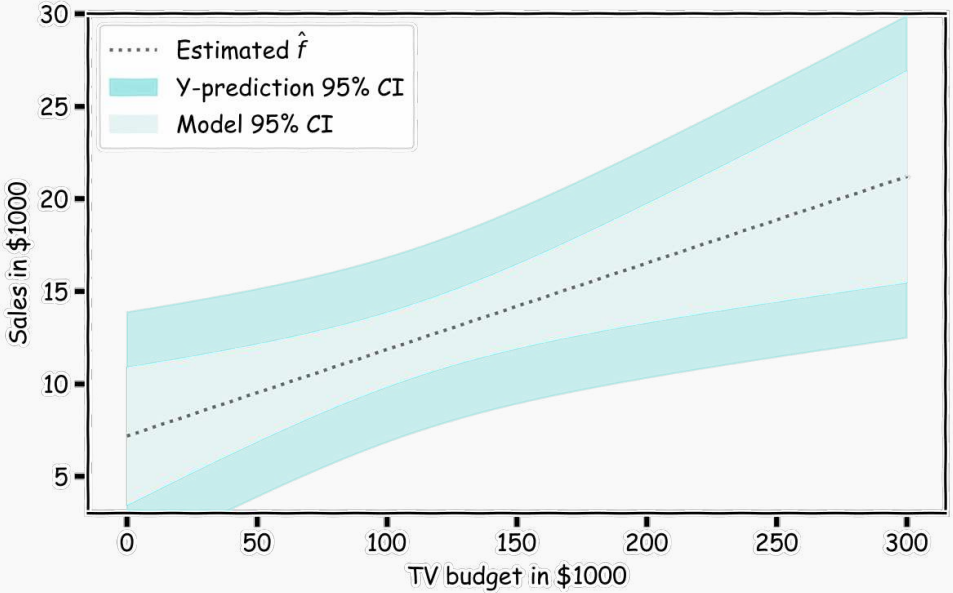
Confidence in predicting $\hat{y}$

- for a given x, we have a distribution of models $f(x)$
- for each of these $f(x)$, the prediction for $y \sim N(f(x), \sigma_\epsilon)$



Confidence in predicting $\hat{y}$

- for a given x, we have a distribution of models $f(x)$
- for each of these $f(x)$, the prediction for $y \sim N(f(x), \sigma_\epsilon)$
- The prediction confidence intervals are then …



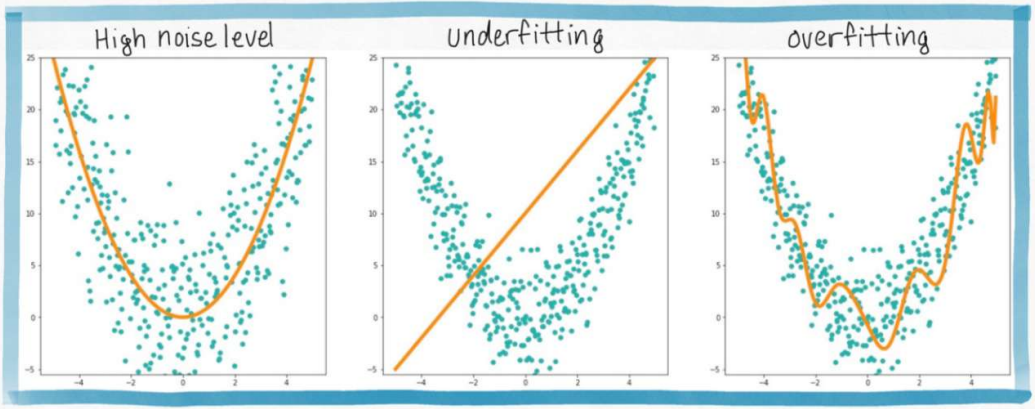
Regularization error, bias vs variance

Test Error and Generalization

We know to evaluate models on both train and test data because models can do well on training data but do poorly on new data.

When models do well on new data is called generalization.

There are at least three ways a model can have a high test error.



Irreducible and Reducible Errors

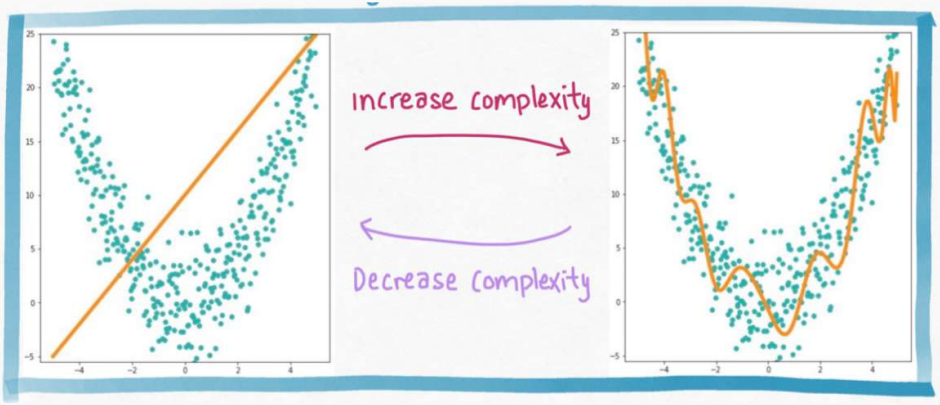
We distinguished the contributions of noise to the generalization error:

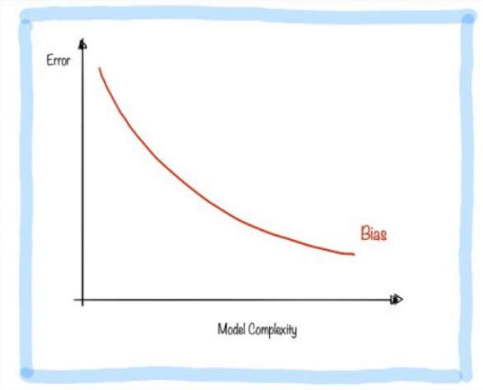
Irreducible error: we can’t do anything to decrease error due to noise.

Reducible error: we can decrease error due to overfitting and underfitting by improving the model.

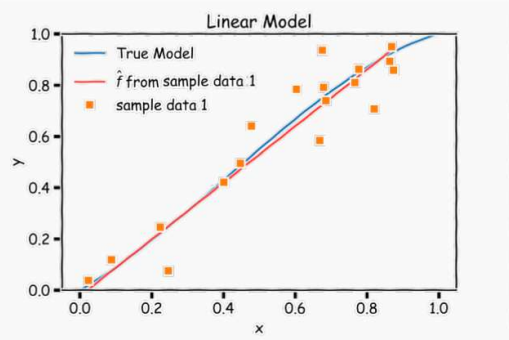
The Bias-Variance: Bias

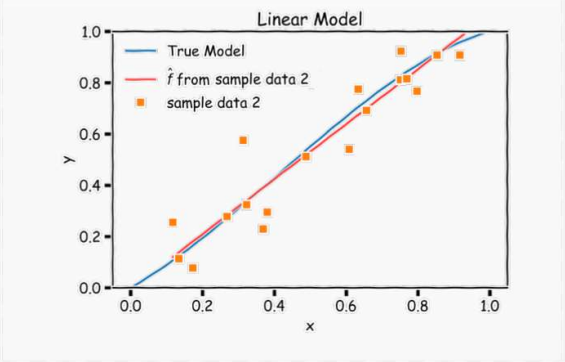
Reducible error comes from either underfitting or overfitting. There is a trade-off between the two sources of errors:

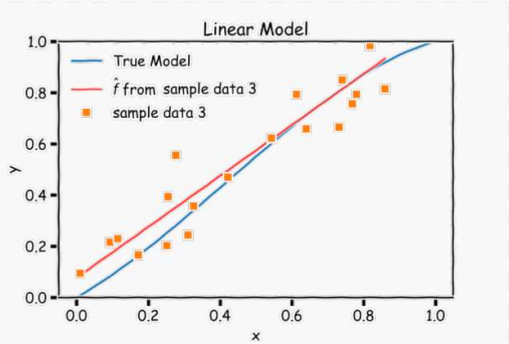


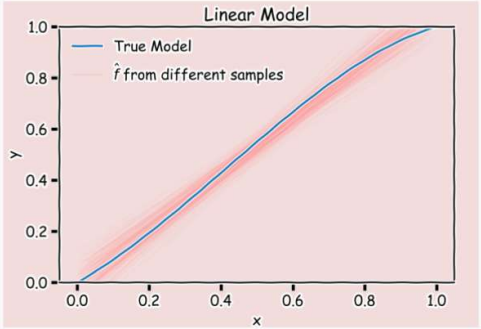


Bias vs Variance: Variance

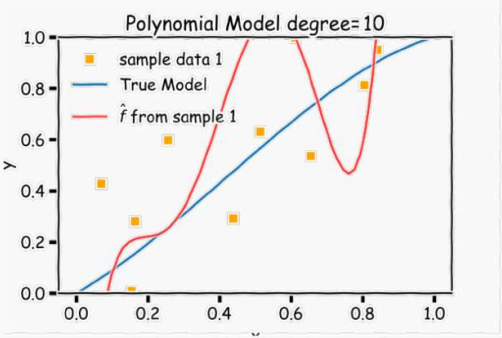


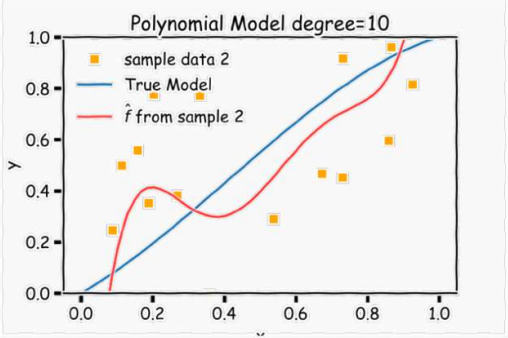


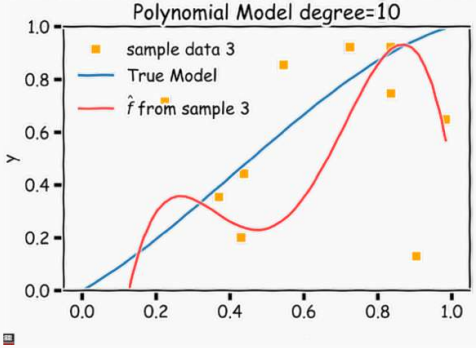


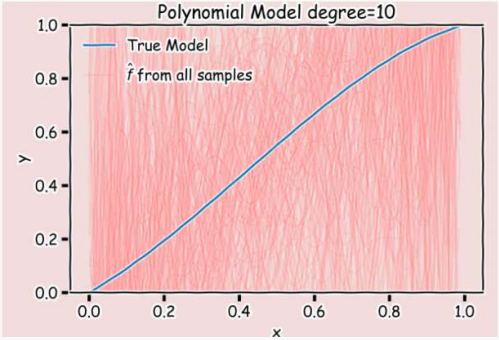


Bias vs Variance





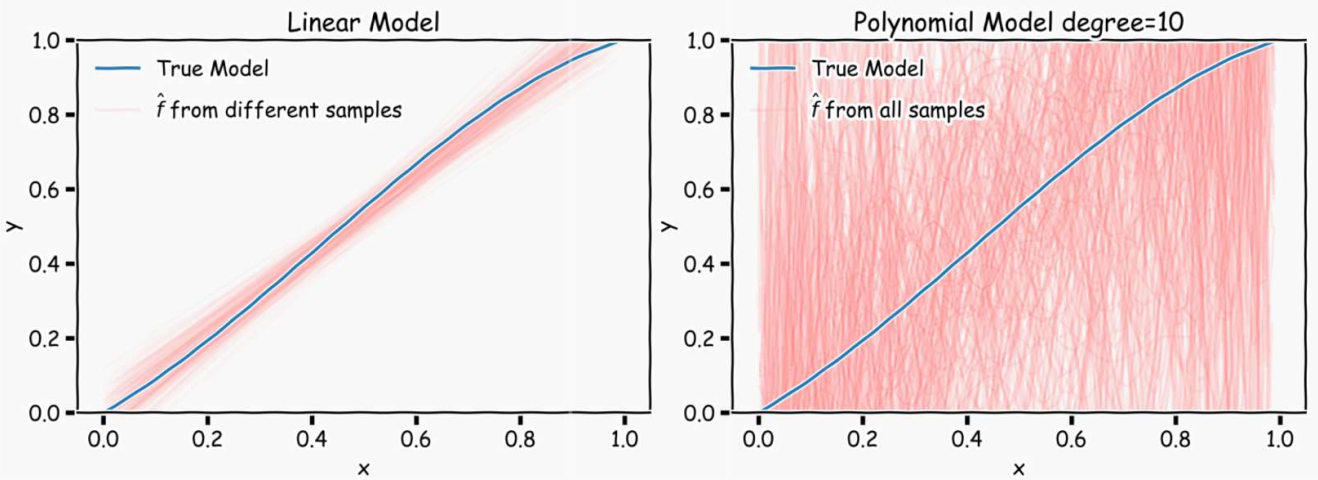




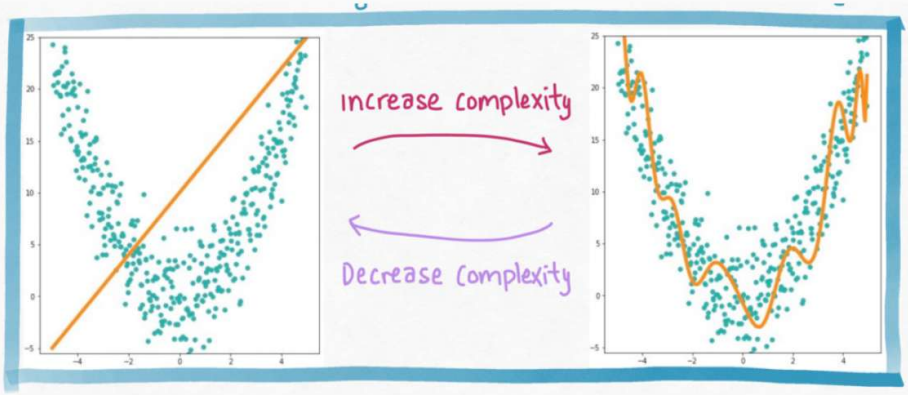
Bias vs Variance

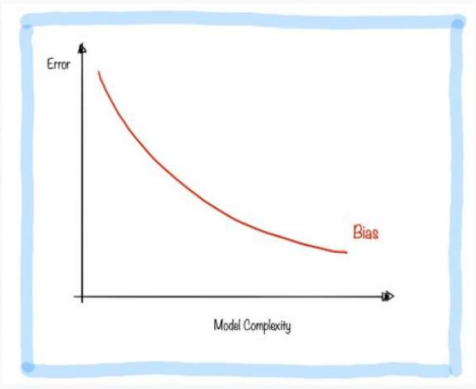
Left: 2000 best fit straight lines, each fitted on a different 20-points training set.

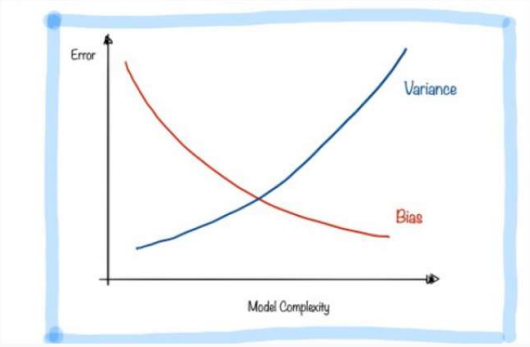
Right: Best-fit models using degree 10 polynomials.

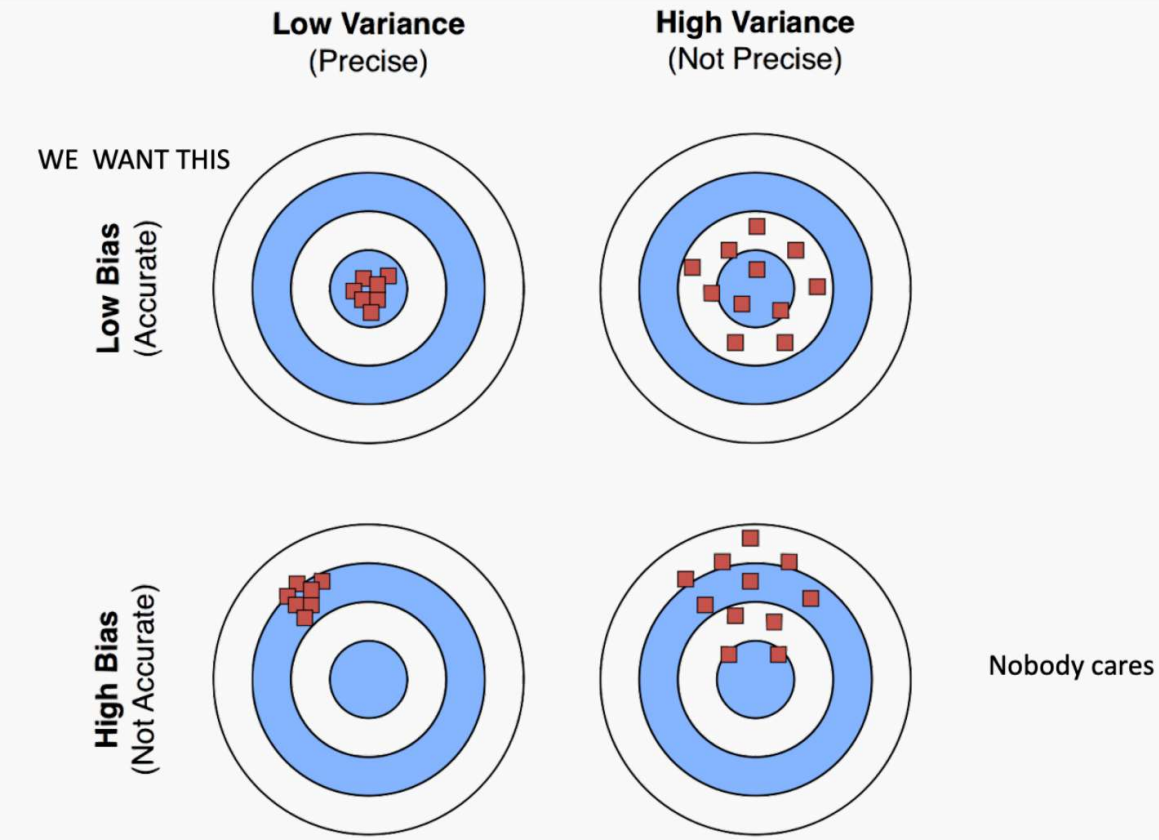


The Bias-Variance Trade Off: Bias








Low Variance
(Precise)
High Variance
(Not Precise)
WE WANT THIS
Low Bias
(Accurate)
High Bias
(Not Accurate)
Nobody cares

Overfitting

Overfitting occurs when a model corresponds too closely to the training set, and as a result, the model fails to fit additional data.

So far, we have seen that overfitting can happen when:

- Too many parameters
- Degree of the polynomial is too large
- Too many interaction terms

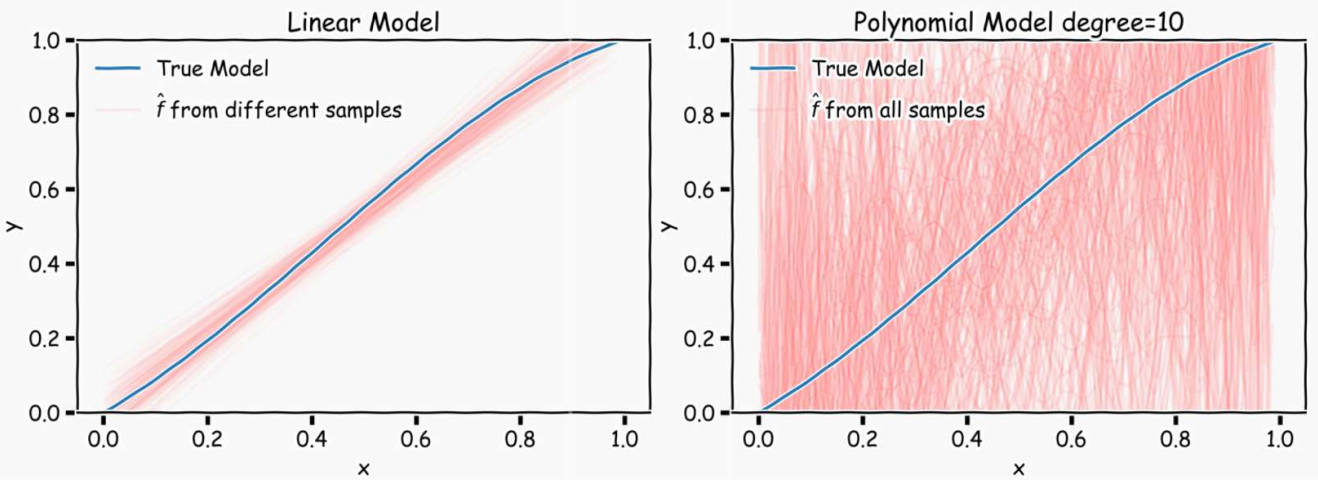
Next, we will see other evidence of overfitting, which will point to a way of avoiding overfitting: Ridge and Lasso regressions.

Ridge and Lasso

Bias vs Variance

Left: 2000 best fit straight lines, each fitted on a different 20-points training set.

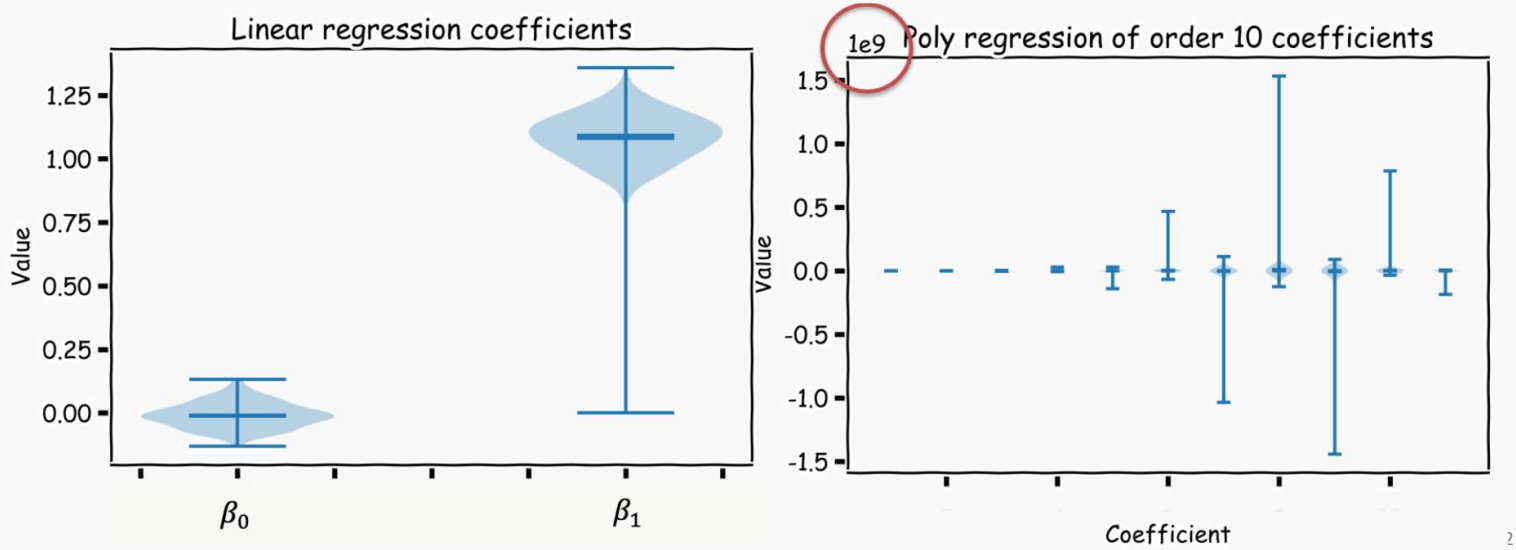
Right: Best-fit models using degree 10 polynomials.



Bias vs Variance

Left: Linear regression coefficients

Right: Poly regression of order 10 coefficients



Regularization: An Overview

The idea of regularization revolves around modifying the loss function L; in particular, we add a regularization term that penalizes some specified properties of the model parameters

e.g. increasing loss

$$L_{reg}(\beta) = L(\beta) + \lambda R(\beta)$$

where λ is a scalar that gives the weight (or importance) of the regularization term.

Fitting the model using the modified loss function L_{reg} would result in model parameters with desirable properties (specified by R).

LASSO Regression

Since we wish to discourage extreme values in model parameter, we need to choose a regularization term that penalizes parameter magnitudes. For our loss function, we will again use MSE.

Together our regularized loss function is:

$$L_{LASSO}(\beta) = \frac{1}{n}\sum_{i=1}^{n} |y_i - \boldsymbol{\beta}^\top \boldsymbol{x}_i|^2 + \lambda \sum_{j=1}^{J} |\beta_j|.$$

Note that $\sum_{j=1}^{J} |\beta_j|$ is the $\boldsymbol{l_1}$ norm of the vector $\boldsymbol{\beta}$

$$\sum_{j=1}^{J} |\beta_j| = \|\boldsymbol{\beta}\|_1$$

Ridge Regression

Alternatively, we can choose a regularization term that penalizes the squares of the parameter magnitudes. Then, our regularized loss function is:

$$L_{Ridge}(\beta) = \frac{1}{n}\sum_{i=1}^{n} |y_i - \boldsymbol{\beta}^\top \boldsymbol{x}_i|^2 + \lambda \sum_{j=1}^{J} \beta_j^2.$$

Note that $\sum_{j=1}^{J} |\beta_j|^2$ is the $\boldsymbol{l_2}$ norm of the vector $\boldsymbol{\beta}$

$$R_{ridge} = \sum_{j=1}^{J} \beta_j^2 = \|\boldsymbol{\beta}\|_2^2$$

Choosing λ

In both ridge and LASSO regression, we see that the larger our choice of the **regularization parameter** λ, the more heavily we penalize large values in β,

- If λ is close to zero, we recover the MSE, i.e. ridge and LASSO regression is just ordinary regression.
- If λ is sufficiently large, the MSE term in the regularized loss function will be insignificant and the regularization term will force β_{ridge} and β_{LASSO} to be close to zero.

To avoid ad-hoc choices, we should select λ using validation or better cross-validation.

Regularization Parameter with a Validation Set

The solution of the Ridge/Lasso regression involves three steps:

- Select λ
- Find the minimum of the ridge/Lasso regression loss function (using the formula for ridge) and record the MSE **on the validation set**.
- Find the λ that gives the **smallest MSE on the validation set**.

Ridge regularization with **validation** only: step by step

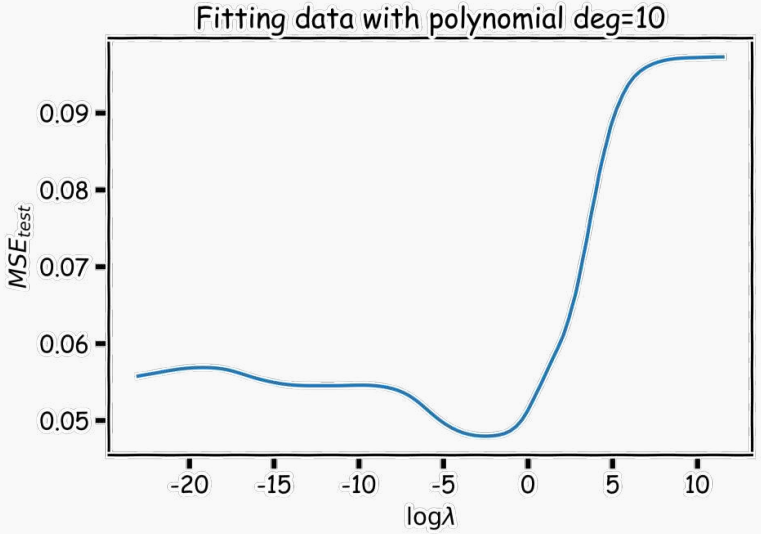
For ridge regression there exist an analytical solution for the coefficients:

$$\hat{\beta}_{Ridge}(\lambda) = \left(\mathrm{X}^{\mathrm{T}}\mathrm{X} + \lambda I\right)^{-1} X^T Y$$

1. split data into $\{\{X,Y\}_{train}, \{X,Y\}_{validation}, \{X,Y\}_{test}\}$
2. for λ in $\{\lambda_{min}, \dots \lambda_{max}\}$:
    1. determine the β that minimizes the L_{ridge}, $\hat{\beta}_{Ridge}(\lambda) = \left(\mathrm{X}^{\mathrm{T}}\mathrm{X} + \lambda I\right)^{-1} X^T Y$, using the train data.
    2. record $L_{MSE}(\lambda)$ using validation data.
3. select the λ that minimizes the loss on the validation data,
   $\lambda_{ridge} = \mathrm{argmin}_{\lambda}\, L_{MSE}(\lambda)$
4. Refit the model using both train and validation data, $\{\{X,Y\}_{train}, \{X,Y\}_{validation}\}$, resulting to $\hat{\beta}_{ridge}(\lambda_{ridge})$
5. report MSE or R^2 on $\{X,Y\}_{test}$ given the $\hat{\beta}_{ridge}(\lambda_{ridge})$

final performance

Ridge regularization with **validation** only: step by step



Lasso regularization with **validation** only: step by step

For Lasso regression there **not** an analytical solution for the coefficients so we use a **solver**.

1. split data into $\{\{X,Y\}_{train}, \{X,Y\}_{validation}, \{X,Y\}_{test}\}$
2. for λ in $\{\lambda_{min}, \dots \lambda_{max}\}$:
   A. determine the β that minimizes the L_{lasso}, $\hat{\beta}_{lasso}(\lambda)$, using the train data. **This is done using a solver.**
   B. record $L_{MSE}(\lambda)$ using validation data
3. select the λ that minimizes the loss on the validation data,
   $\lambda_{lasso} = \text{argmin}_{\lambda}\, L_{MSE}(\lambda)$
4. Refit the model using both train and validation data, $\{\{X,Y\}_{train}, \{X,Y\}_{validation}\}$, resulting to $\hat{\beta}_{lasso}(\lambda_{lasso})$
5. report MSE or R^2 on $\{X,Y\}_{test}$ given the $\hat{\beta}_{lasso}(\lambda_{lasso})$

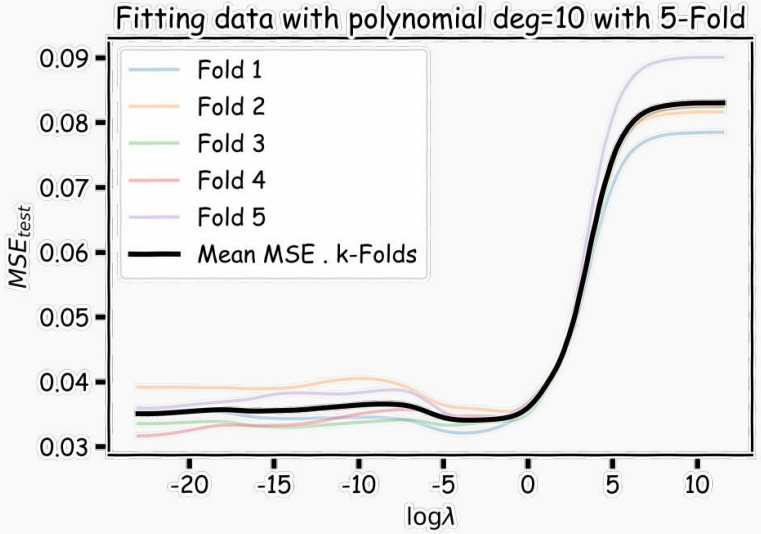
Ridge regularization with **CV**: step by step

	λ_1	λ_2	...	λ_n
k_1	L_{11}	L_{12}	..	...
k_2	L_{21}	...	..	...
...	..	...	..	...
k_n	...	...	...	...
E[]	$\bar{L}_1$	$\bar{L}_2$	...	$\bar{L}_n$

1. remove $\{X,Y\}_{test}$ from data
2. split the rest of data into K folds, $\{\{X,Y\}_{train}^{-k}, \{X,Y\}_{val}^{k}\}$
3. for k in $\{1,\dots,K\}$
    1. for λ in $\{\lambda_0, \dots, \lambda_n\}$:
        A. determine the β that minimizes the L_{ridge}, $\hat{\beta}_{ridge}(\lambda,k) = \left(\mathrm{X^T X} + \lambda I\right)^{-1} X^T Y$, using the train data of the fold, $\{X,Y\}_{train}^{-k}$.
        B. record $L_{MSE}(\lambda,k)$ using the validation data of the fold $\{X,Y\}_{val}^{k}$

        At this point we have a 2-D matrix, rows are for different k, and columns are for different λ values.
4. Average the $L_{MSE}(\lambda,k)$ for each λ, $\bar{L}_{MSE}(\lambda)$.
5. Find the λ that minimizes the $\bar{L}_{MSE}(\lambda)$, resulting to λ_{ridge}.
6. Refit the model using the full training data, $\{\{X,Y\}_{train}, \{X,Y\}_{val}\}$, resulting to $\hat{\beta}_{ridge}(\lambda_{ridge})$
7. report MSE or R^2 on $\{X,Y\}_{test}$ given the $\hat{\beta}_{ridge}(\lambda_{ridge})$

Ridge regularization with **validation** only: step by step



Ridge and Lasso Comparision

Ridge, LASSO - Computational complexity

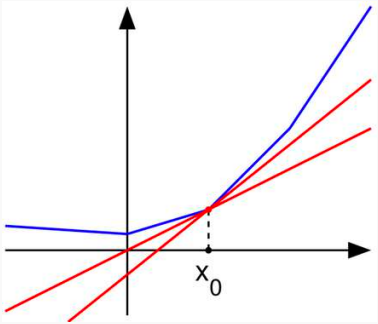
Solution to ridge regression:

lasso has no closed form solution

$$\beta = (X^T X + \lambda I)^{-1} X^T Y$$

The solution to the LASSO regression:

LASSO has no conventional analytical solution, as the L1 norm has no derivative at 0. We can, however, use the concept of subdifferential or subgradient to find a manageable expression.



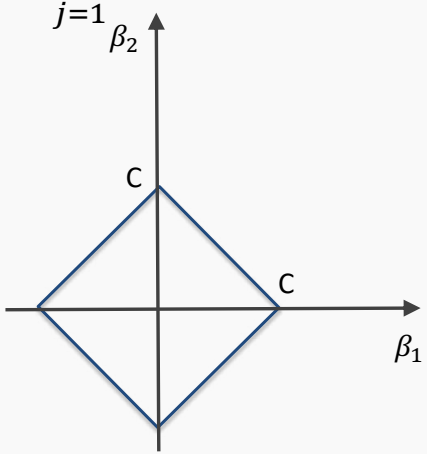
The Geometry of Regularization (LASSO)

$$L_{LASSO}(\boldsymbol{\beta}) = \frac{1}{n}\sum_{i=1}^{n} |y_i - \boldsymbol{\beta}^T \boldsymbol{x}|^2 + \lambda \sum_{j=1}^{J} |\beta_j|$$

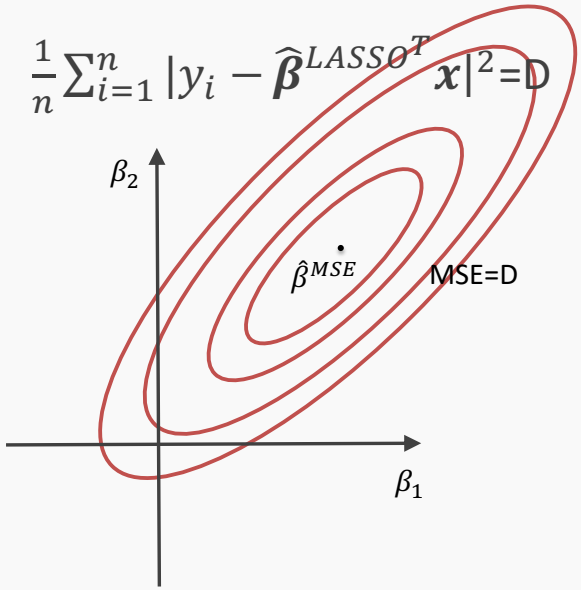
MSE

$$\widehat{\boldsymbol{\beta}}^{LASSO} = \operatorname{argmin} L_{LASSO}(\boldsymbol{\beta})$$

R: $\lambda \sum_{j=1}^{J} |\hat{\beta}_j^{LASSO}| = C$



$\frac{1}{n}\sum_{i=1}^{n} |y_i - \widehat{\boldsymbol{\beta}}^{LASSO^T} \boldsymbol{x}|^2 = D$

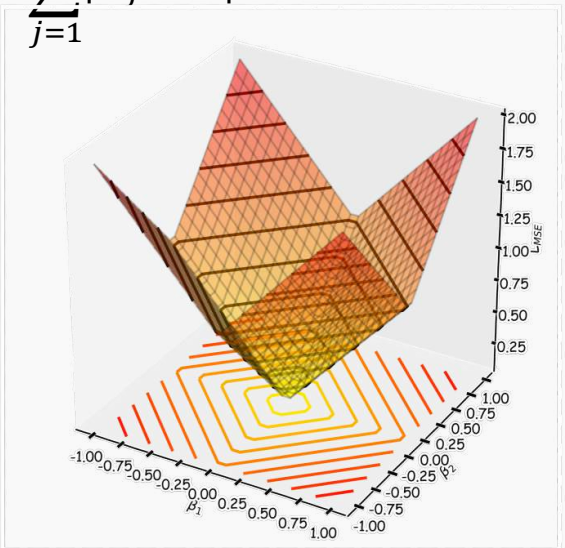


The Geometry of Regularization (LASSO)

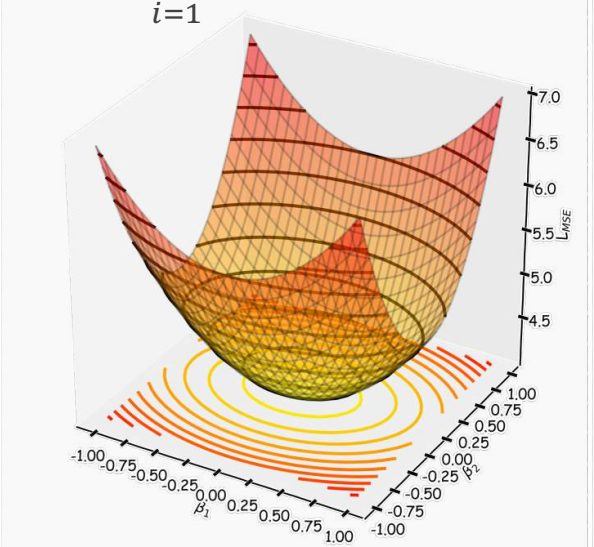
$$L_{LASSO}(\boldsymbol{\beta}) = \frac{1}{n}\sum_{i=1}^{n} |y_i - \boldsymbol{\beta}^T \boldsymbol{x}\Big|^2 + \lambda \sum_{j=1}^{J} |\beta_j|$$

$$\widehat{\boldsymbol{\beta}}^{LASSO} = \operatorname{argmin} L_{LASSO}(\boldsymbol{\beta})$$

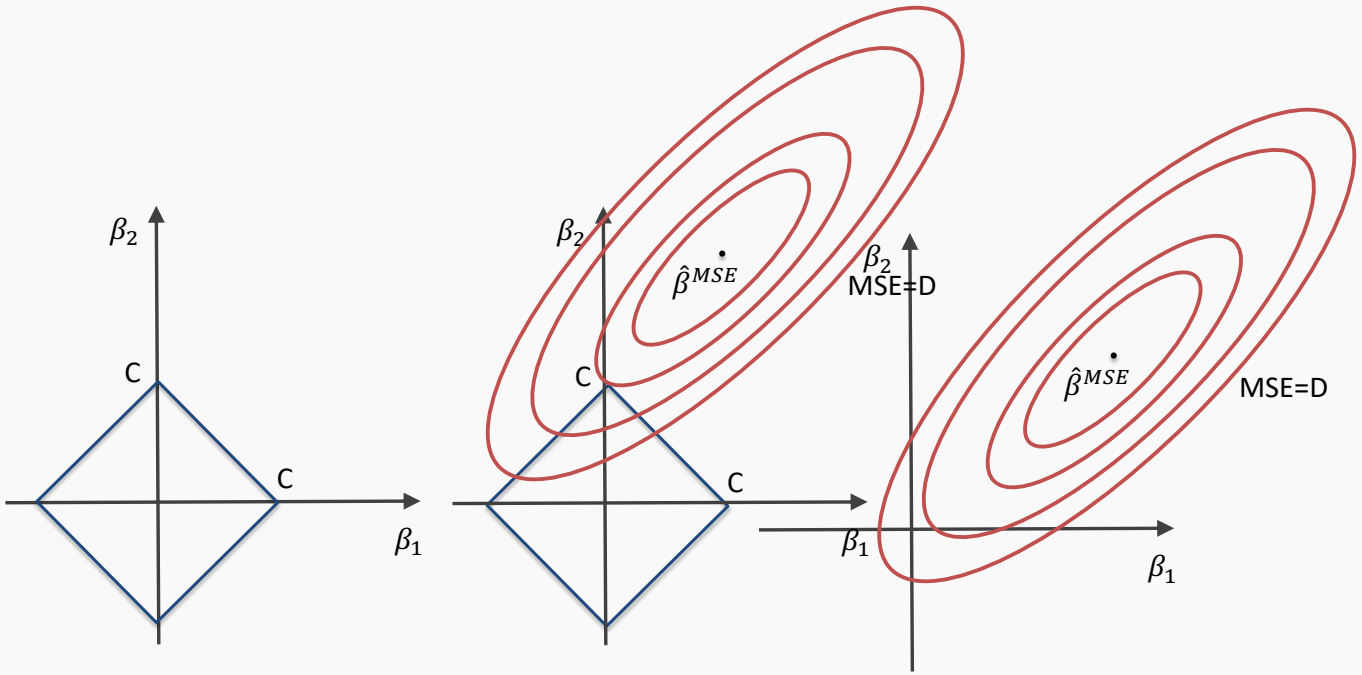
$$L_1 = \lambda \sum_{j=1}^{J} \left| \hat{\beta}_j^{LASSO} \right|$$



$$L_{MSE}(\boldsymbol{\beta}) = \frac{1}{n}\sum_{i=1}^{n} |y_i - \boldsymbol{\beta}^T \boldsymbol{x}\Big|^2$$



The Geometry of Regularization (LASSO)

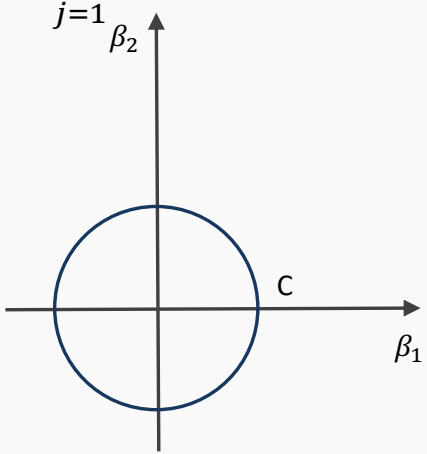


The Geometry of Regularization (Ridge)

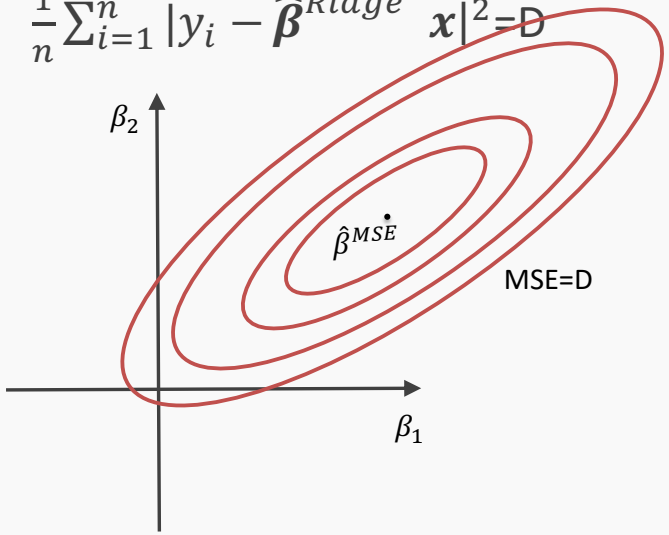
$$L_{Ridge}(\boldsymbol{\beta}) = \frac{1}{n}\sum_{i=1}^{n} |y_i - \boldsymbol{\beta}^T \boldsymbol{x}\Big|^2 + \lambda \sum_{j=1}^{J} (\beta_j)^2$$

$$\widehat{\boldsymbol{\beta}}^{Ridge} = \operatorname{argmin} L_{Ridge}(\boldsymbol{\beta})$$

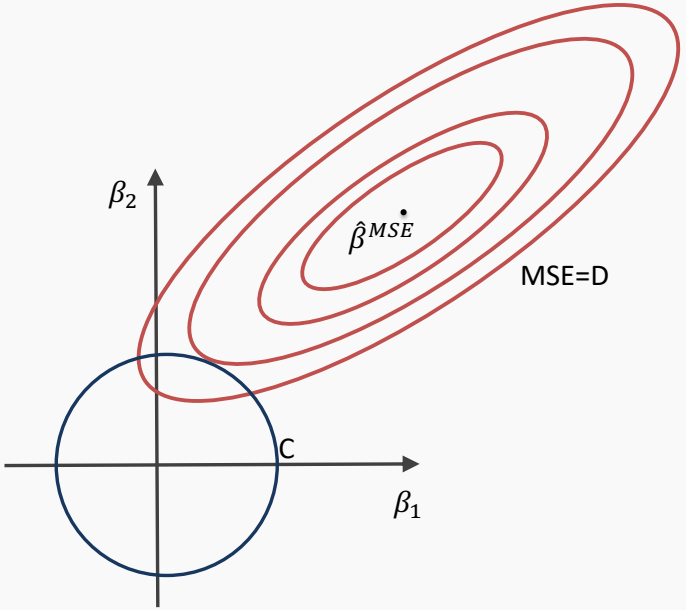
$$\lambda \sum_{j=1}^{J} \left|\hat{\beta}_j^{Ridge}\right|^2 = C$$



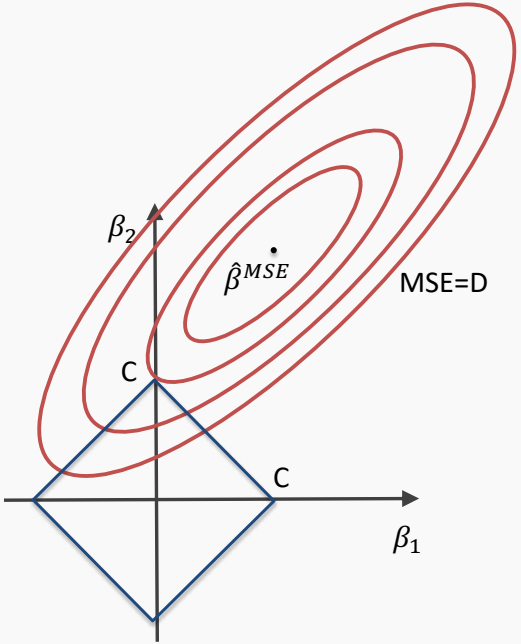
$$\frac{1}{n}\sum_{i=1}^{n} |y_i - \widehat{\boldsymbol{\beta}}^{Ridge^T} \boldsymbol{x}|^2 = D$$

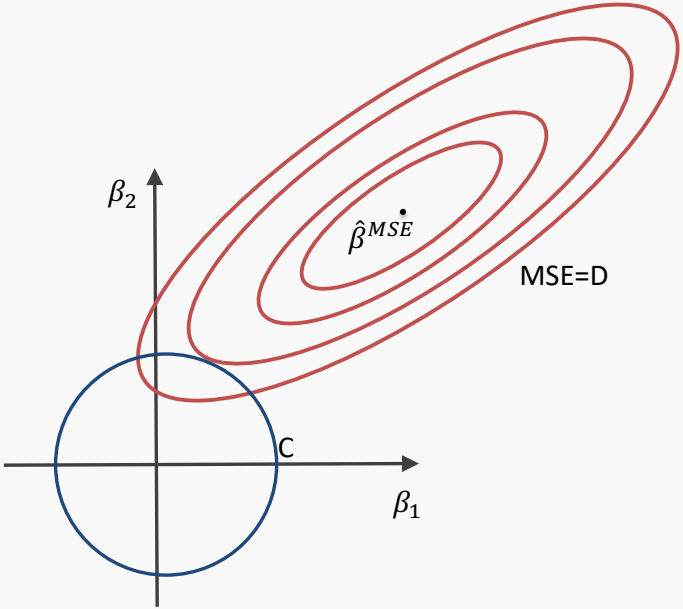


The Geometry of Regularization (Ridge)

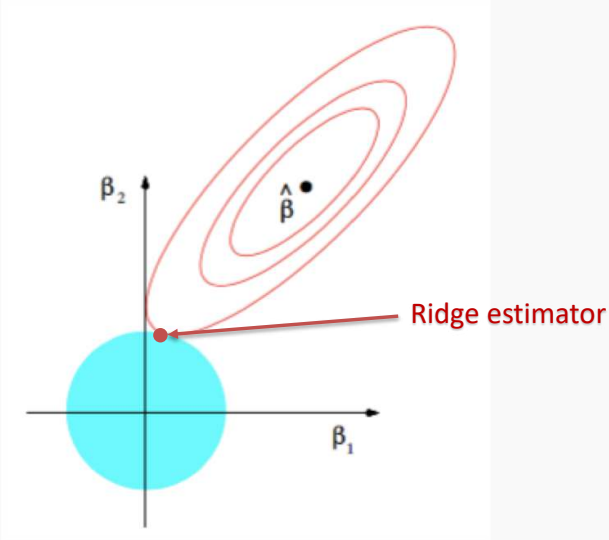


The Geometry of Regularization

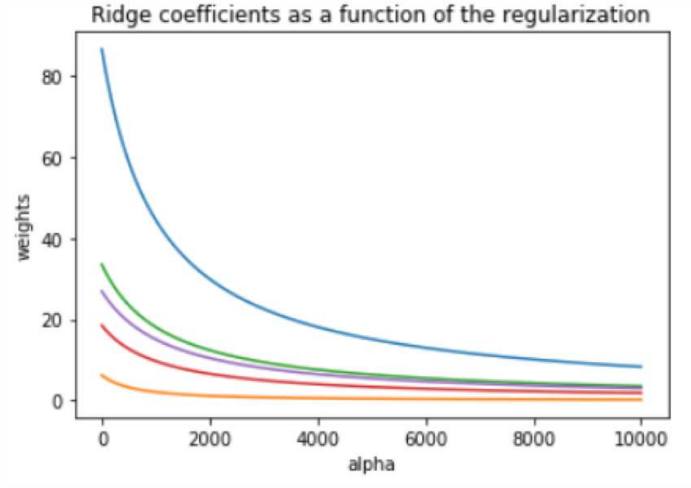




Ridge visualized

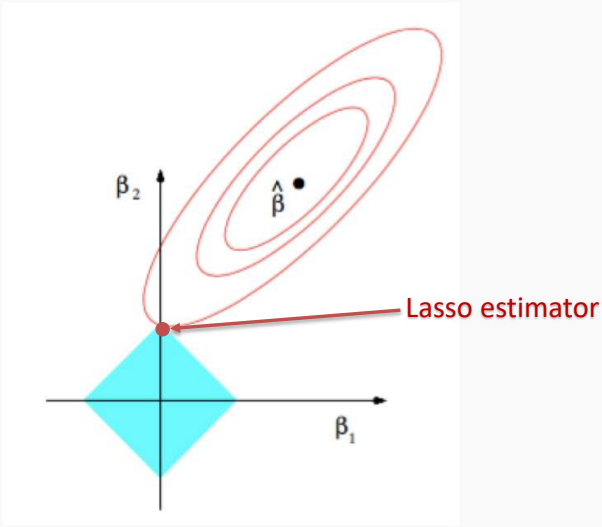


The ridge estimator is where the constraint and the loss intersect.

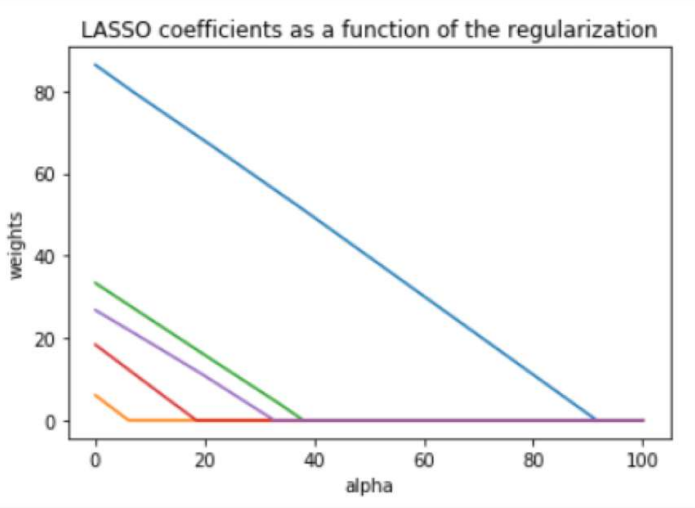


The values of the coefficients decrease as lambda increases, but they are not nullified.

LASSO visualized



The Lasso estimator tends to zero out parameters as the OLS loss can easily intersect with the constraint on one of the axis.



The values of the coefficients decrease as lambda increases, and are nullified fast.

Variable Selection as Regularization

Question: What are the pros and cons of the two approaches?

Since LASSO regression tends to produce zero estimates for a number of model parameters - we say that LASSO solutions are **sparse** - we consider LASSO to be a variable selection method.

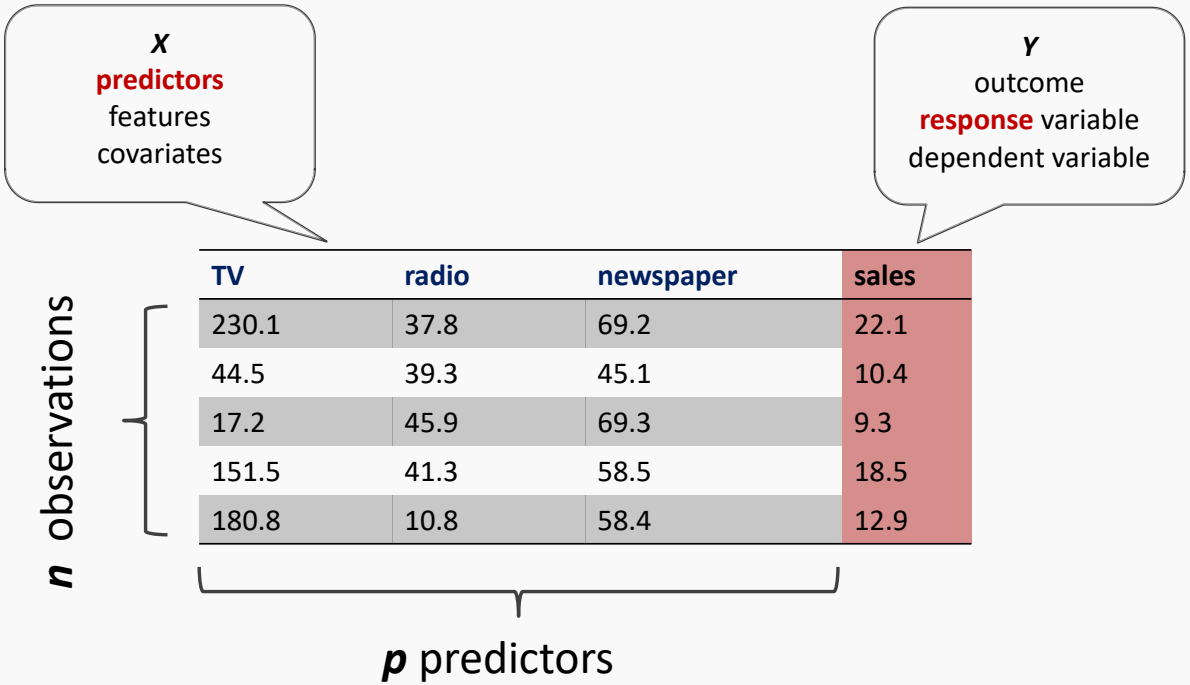
Ridge is faster to compute but many prefer using LASSO for variable selection (as well as for suppressing extreme parameter values) and therefore easier to interpret.

Classification

Lecture Outline

- Introduction to Classification
- *k*-NN for Classification
- Why not Linear Regression?
- Binary Response & Logistic Regression
  - Estimating the Simple Logistic Model
  - Classification using the Logistic Model
  - Multiple Logistic Regression

Advertising Data (from earlier lectures)



Heart Data

response variable ***Y*** is Yes/No

Age	Sex	ChestPain	RestBP	Chol	Fbs	RestECG	MaxHR	ExAng	Oldpeak	Slope	Ca	Thal	AHD
63	1	typical	145	233	1	2	150	0	2.3	3	0.0	fixed	No
67	1	asymptomatic	160	286	0	2	108	1	1.5	2	3.0	normal	Yes
67	1	asymptomatic	120	229	0	2	129	1	2.6	2	2.0	reversable	Yes
37	1	nonanginal	130	250	0	0	187	0	3.5	3	0.0	normal	No
41	0	nontypical	130	204	0	2	172	0	1.4	1	0.0	normal	No

Heart Data

These data contain a binary outcome HD for 303 patients who presented with chest pain. An outcome value of:

- ***Yes*** indicates the presence of heart disease based on an angiographic test,
- ***No*** means no heart disease.

There are 13 predictors including:

- Age
- Sex (0 for women, 1 for men)
- Chol (a cholesterol measurement),
- MaxHR
- RestBP

and other heart and lung function measurements.

Classification

Up to this point, the methods we have seen have centered around modeling and the prediction of a **quantitative** response variable (ex, number of taxi pickups, number of bike rentals, etc). Linear **regression** (and Ridge, LASSO, etc) perform well under these situations

When the response variable is **categorical**, then the problem is no longer called a regression problem but is instead labeled as a **classification problem**.

The goal is to attempt to classify each observation into a category (aka, class or cluster) defined by *Y*, based on a set of predictor variables *X*.

Typical Classification Examples

The motivating examples for the lecture(s) are based [mostly] on medical data sets. Classification problems are common in this domain:

- Trying to determine where to set the *cut-off* for some diagnostic test (pregnancy tests, prostate or breast cancer screening tests, etc...)
- Trying to determine if cancer has gone into remission based on treatment and various other indicators
- Trying to classify patients into types or classes of disease based on various genomic markers

k-NN for Classification

k-Nearest Neighbors

We've already seen the *k*-NN method for predicting a quantitative response (it was the very first method we introduced). How was *k*-NN implemented in the Regression setting (quantitative response)?

The approach was simple: to predict an observation's response, use the **other** available observations that are most similar to it.

For a specified value of *k*, each observation's outcome is predicted to be the **average** of the *k*-closest observations as measured by some distance of the predictor(s).

With one predictor, the method was easily implemented.

Review: Choice of k

How well the predictions perform is related to the choice of *k*.

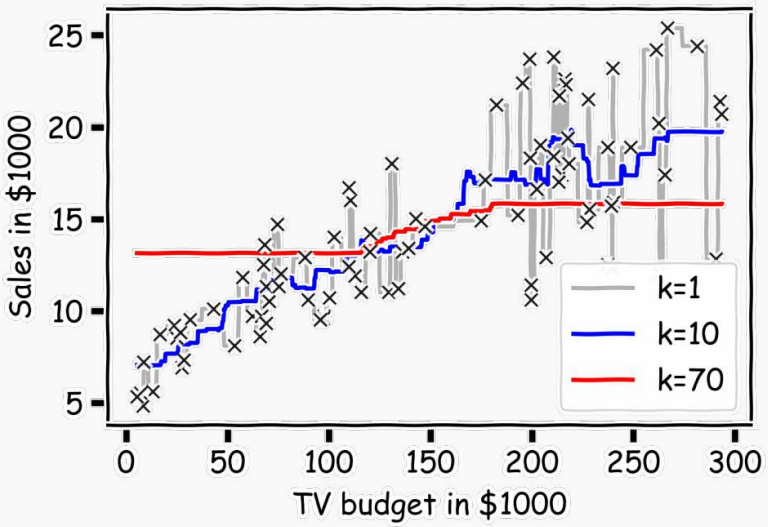
What will the predictions look like if *k* is very small? What if it is very large?

More specifically, what will the predictions be for new observations if $k = n$?

$$\bar{Y}$$

A picture is worth a thousand words...

Choice of *k* matters



k-NN for Classification

How can we modify the *k*-NN approach for classification?

The approach here is the same as for *k*-NN regression: use the other available observations that are most similar to the observation we are trying to predict (classify into a group) based on the predictors at hand.

How do we classify which category a specific observation should be in based on its nearest neighbors?

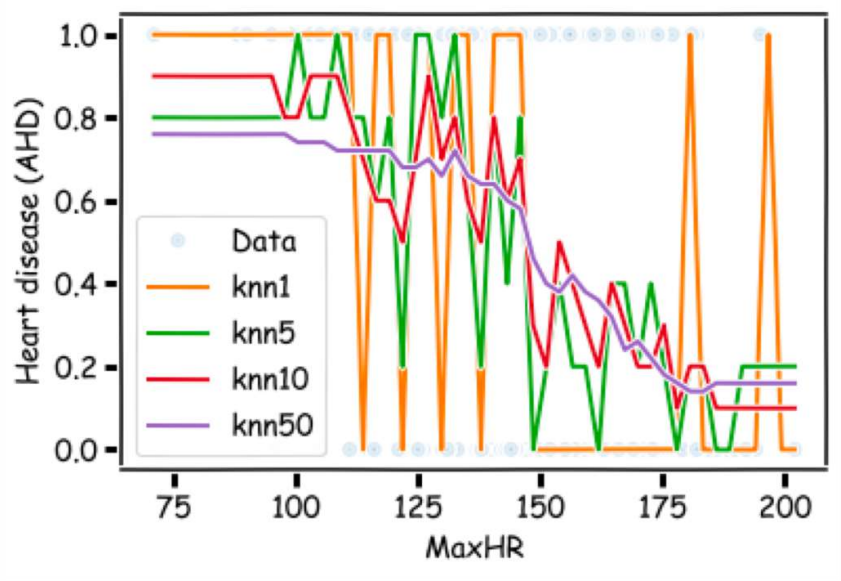
k-NN for Classification: formal definition

The *k*-NN classifier first identifies the *k* points in the training data that are closest to x_0, represented by $\mathcal{N}_0$. It then estimates the conditional probability for class *j* as the fraction of points in $\mathcal{N}_0$ whose response values equal *j*:

$$P(Y = j|X = x_0) = \frac{1}{k} \sum_{i \in \mathcal{N}_0} I(y_i = j)$$

Then, the *k*-NN classifier predicts this new observation, x_0, to be in the class with largest estimated probability.

Estimated Probabilities in *k*-NN Classification



k-NN for Classification (cont.)

There are some issues that may arise:

- How can we handle a tie?

- What could be a major problem with always classifying to the most common group amongst the neighbors?

- How can we handle this?

k-NN Classification in Python

Performing kNN classification in python is done via **`KNeighborsClassifier`** in **`sklearn.neighbors.`**

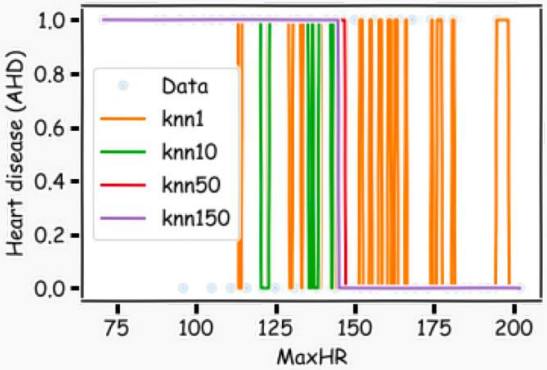
An example:

```
data_x = df_heart[['MaxHR']]
data_y = df_heart['AHD']

knn1 = KNeighborsClassifier(n_neighbors=1).fit(data_x, data_y)
knn10 = KNeighborsClassifier(n_neighbors=10).fit(data_x, data_y)
knn50 = KNeighborsClassifier(n_neighbors=50).fit(data_x, data_y)
knn150 = KNeighborsClassifier(n_neighbors=150).fit(data_x, data_y)

print(knn1.score(data_x,data_y))
print(knn10.score(data_x,data_y))
print(knn50.score(data_x,data_y))
print(knn150.score(data_x,data_y))
```

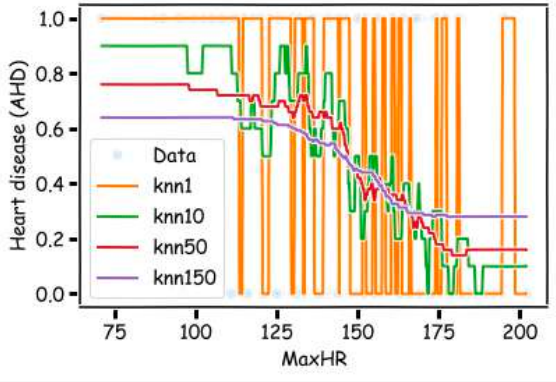
```
0.693069306930693
0.7161716171617162
0.7062706270627063
0.6996699669966997
```

Pure Classifications in *k*-NN Classification Models

Pure Classifications:



Estimated Probabilities:



What will these plots look like when there are 2 predictors? Or 3+ predictors?

This is our first glimpse at a **classification boundary** (more to come).

k-NN with Multiple Predictors

How could we extend *k*-NN (both regression and classification) when there are multiple predictors?

We would need to define a measure of distance for observations in order to which are the most similar to the observation we are trying to predict.

Euclidean distance is a good option. To measure the distance of a new observation, $\boldsymbol{x}_0$ from each observation in the data set, $\boldsymbol{x}_i$:

$$D^2(\boldsymbol{x}_i, \boldsymbol{x}_0) = \sum_{j=1}^{P} \left(x_{i,j} - x_{0,j}\right)^2$$

k-NN with Multiple Predictors (cont.)

But what must we be careful about when measuring distance?

1. Differences in variability in our predictors!
2. Having a mixture of quantitative and categorical predictors.

So what should be good practice? To determine closest neighbors when $p > 1$, you should first standardize the numeric predictors! And you can even standardize the binaries if you want to include them.

How else could we determine closeness in this multi-dimensional setting?

Parametric Modeling: Why not Linear Regression?

Simple Classification Example

Given a dataset:

$$\{(\boldsymbol{x}_1, y_1), (\boldsymbol{x}_2, y_2), \cdots, (\boldsymbol{x}_N, y_N)\}$$

where the y are categorical (sometimes referred to as ***qualitative***), we would like to be able to predict which category y takes on given x.

A categorical variable y could be encoded to be quantitative. For example, if y represents concentration of UCLA undergrads, then y could take on the values:

$$y = \begin{cases} 1 & if \text{ Computer Science (CS)} \\ 2 & if \text{ Statistics} \\ 3 & \text{otherwise} \end{cases} .$$

Linear regression **does not work well**, or is not appropriate at all, in this setting.

Simple Classification Example (cont.)

A linear regression could be used to predict *y* from ***x***. What would be wrong with such a model?

The model would imply a specific ordering of the outcome, and would treat a one-unit change in *y* equivalent. The jump from *y* = 1 to y = 2 (CS to Statistics) should not be interpreted as the same as a jump from *y* = 2 to *y* = 3 (Statistics to everyone else).

Similarly, the response variable could be reordered such that *y* = 1 represents Statistics and *y* = 2 represents CS, and then the model estimates and predictions would be fundamentally different.

If the categorical response variable was ***ordinal*** (had a natural ordering, like class year: Freshman, Sophomore, etc.), then a linear regression model would make some sense but is still not ideal.

Even Simpler Classification Problem: Binary Response

The simplest form of classification is when the response variable y has only two categories, and then an ordering of the categories is natural. For our example, a patient in the ICU could be categorized as having [atherosclerotic] heart disease (AHD) or not (note, the $y = 0$ category is a "catch-all" so it would involve those patients with lots of other diseases or diagnoses):

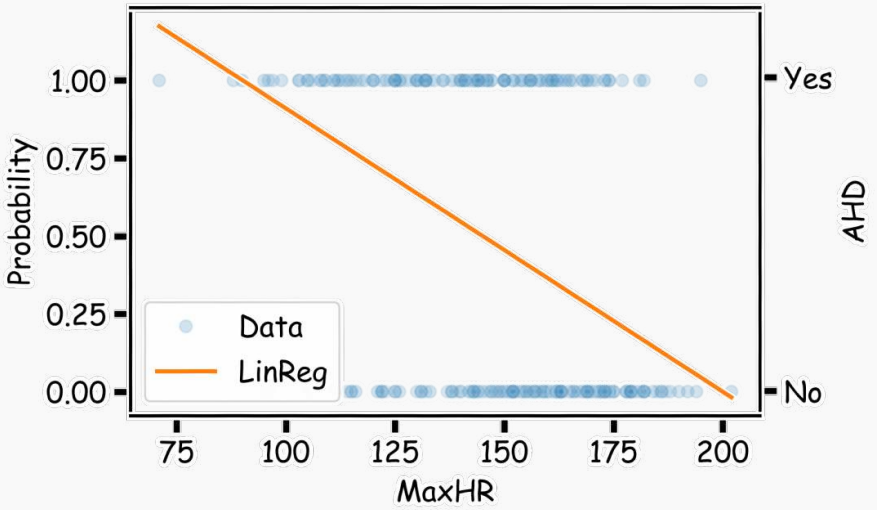
$$y = \begin{cases} 1 \text{ if patient has heart disease} \\ 0 \text{ otherwise.} \end{cases}$$

Linear regression could be used to predict y directly from a set of covariates (like sex, age, resting HR, etc.), and if $\hat{y} \geq 0.5$, we could predict the patient to have AHD and predict not to have heart disease if $\hat{y} < 0.5$.

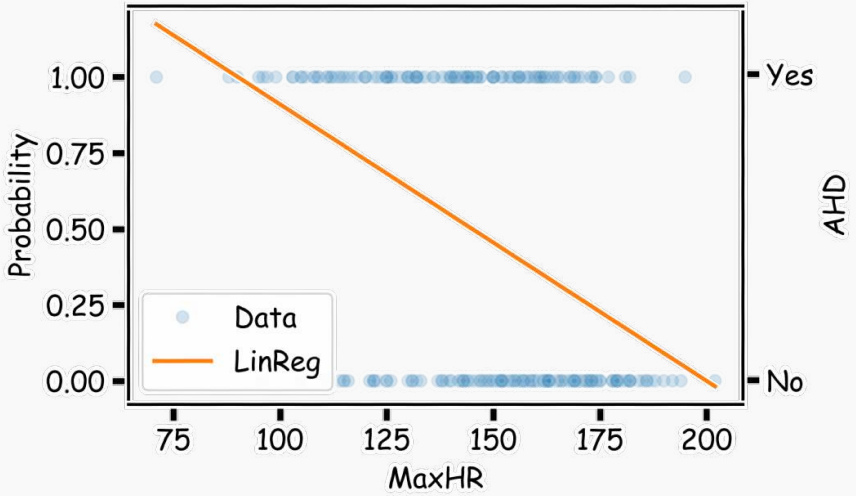
Even Simpler Classification Problem: Binary Response (cont)

What could go wrong with this linear regression model?

.



Even Simpler Classification Problem: Binary Response (cont)

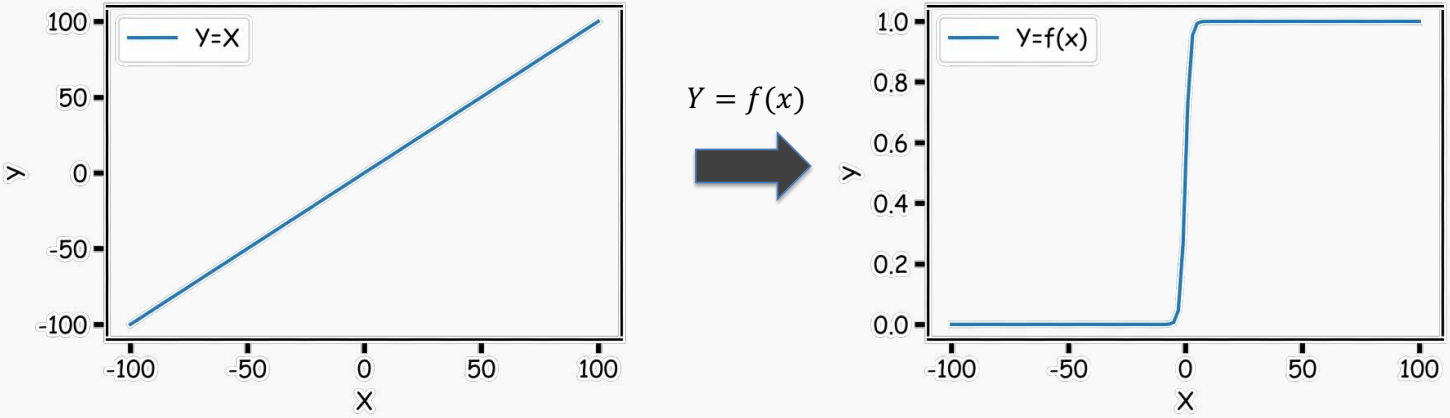


The main issue is you could get non-sensical values for y. Since this is modeling $P(y = 1)$, values for $\hat{y}$ below 0 and above 1 would be at odds with the natural measure for y. Linear regression can lead to this issue.

Binary Response & Logistic Regression

Logistic Regression

Think of a function that would do this for us



Logistic Regression

Logistic Regression addresses the problem of estimating a probability, $P(y = 1)$, to be outside the range of $[0,1]$. The logistic regression model uses a function, called the ***logistic*** function, to model $P(y = 1)$:

$$P(Y = 1) = \frac{e^{\beta_0 + \beta_1 X}}{1 + e^{\beta_0 + \beta_1 X}} = \frac{1}{1 + e^{-(\beta_0 + \beta_1 X)}}$$

Logistic Regression

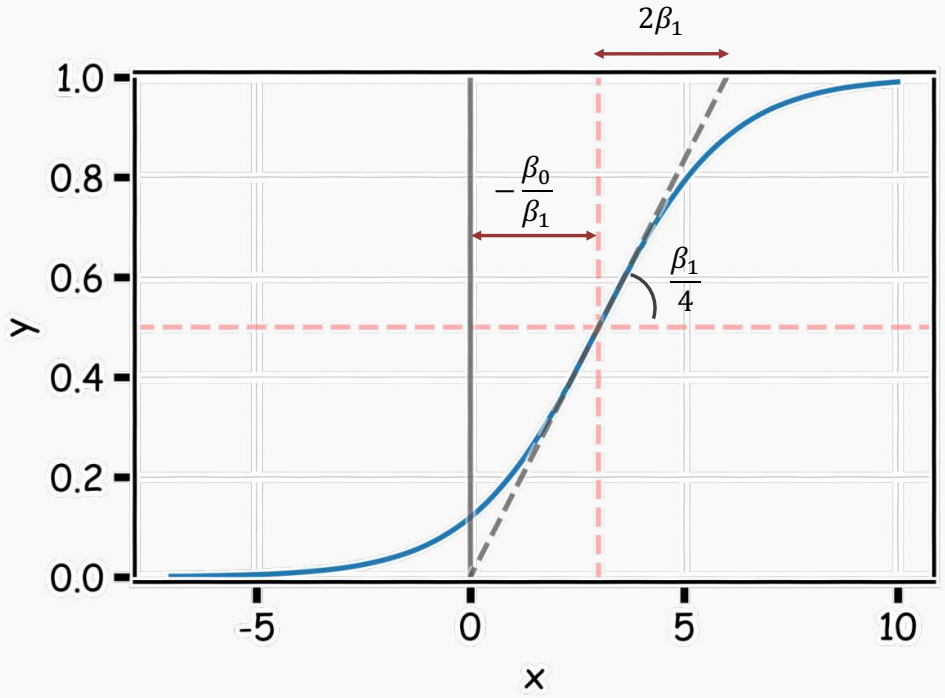
As a result the model will predict $P(y = 1)$ with an S-shaped curve, which is the general shape of the logistic function.

β_0 shifts the curve right or left by $c = -\frac{\beta_0}{\beta_1}$.

β_1 controls how steep the S-shaped curve is. Distance from ½ to almost 1 or ½ to almost 0 is $\frac{2}{\beta_1}$

Note: if β_1 is positive, then the predicted $P(y = 1)$ goes from zero for small values of X to one for large values of X and if β_1 is negative, then the $P(y = 1)$ has opposite association.

Logistic Regression



Logistic Regression: interpretation

With a little bit of algebraic work, the logistic model can be rewritten as:

$$\ln\left(\frac{P(Y=1)}{1-P(Y=1)}\right) = \beta_0 + \beta_1 X.$$

The value inside the natural log function $\frac{P(Y=1)}{1-P(Y=1)}$, is called the ***odds***, thus logistic regression is said to model the ***log-odds*** with a linear function of the predictors or features, X. This gives us the natural interpretation of the estimates similar to linear regression: a one unit change in X is associated with a β_1 change in the log-odds of *success* $(Y = 1)$; or better yet, **a one unit change in $\boldsymbol{X}$ is associated with an $\boldsymbol{e^{\beta_1}}$ multiplicate change in the odds of *success* $(\boldsymbol{Y = 1})$** .

Next lecture…

Estimating the Simple Logistic Model