Data Science and Big Data Analytics: Making Data-Driven Decisions

Module 2 Regression and Prediction

Lecture Notes

Overview

The tools and techniques developed in the previous module allowed us to take a peek at large and unstructured data. Once we get a better handle on what this data looks like and what information it contains, a natural question is **"Can we use it to make predictions?"**. The need to formulate predictions from data can arise in a variety of contexts, whether we are trying to predict future data or simply data that is hard or expensive to collect in the future but was available in the past. The general idea behind prediction is to analyze data to identify an input/output function. "Module 2: Regression and Prediction" covers many techniques for this task. Specifically, we will discuss:

1. Linear and nonlinear regression together with their extensions, including the important case of logistic regression for binary classification and causal inference where the goal is to understand the effects of actively manipulating a variable as opposed to passively measuring it.

2. Modern regression with high-dimensional data or finding a needle in a haystack: for large datasets, it becomes necessary to sort out which variables are relevant for prediction and which are not. Recent years have witnessed the development of new statistical techniques, such as Lasso or Random Forests, that are computationally scalable to large datasets and that automatically select relevant data.

3. Regression and causal inference to explain why "correlation does not imply causation" and how we can overcome this intrinsic limitation of regression by resorting to randomized control studies or controlling for confounding.

This module is taught by Victor Chernozhukov from the MIT Economics Department and MIT Institute for Data Systems and Society. Professor Chernozhukov's expertise covers econometrics, machine learning, and statistics in which he has developed fundamental methodology, particularly to deal with high dimensional data. This module not only covers a description of the methodology involved but also a mathematical introduction to its underlying principles.

Goals

1. Understanding of basic regression for prediction and inferential purposes.

2. Understanding of modern linear and nonlinear regression for prediction and inferential purposes.

3. Getting practical experience of using classical and modern regression methods for prediction and inferential purposes.

Objectives

Students should be able to:

1. Recognize a regression problem

2. Propose an adequate model for inference purposes

3. Recognize the difference between correlation and causation

4. Recognize high-dimensional setups

5. Select appropriate method for high-dimensional learning

6. Understand the limitations of observational studies for causal inference with particular focus on confounding

7. Recognize and implement randomized controlled studies and perform causal inference in these

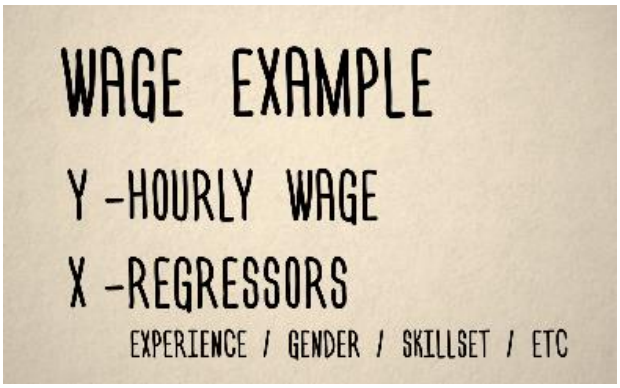
2.1.1 Introduction

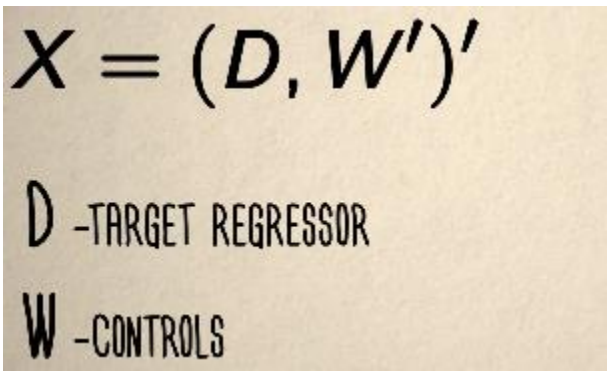
Regression analysis is about discovering correlations between the outcome, y and the set of regressors x, which sometimes are also called features.

Outcome Y: real random variable

Regressors X or features: vector of random variables: $X = (X_1, X_2, \dots, X_p), assume\ X_1 = 1$

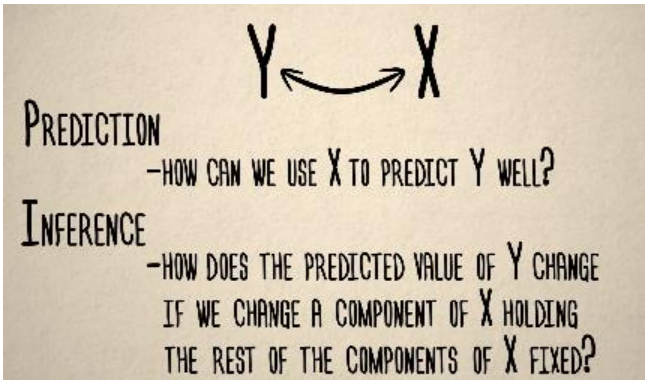
Example:



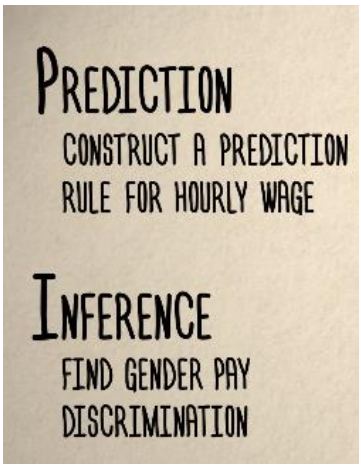


The purpose of the regression analysis is to characterize the statistical relation y to x.

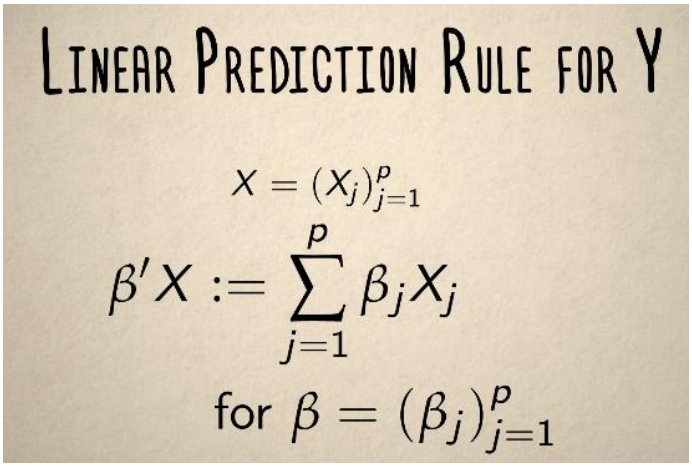
Two questions:



Example:

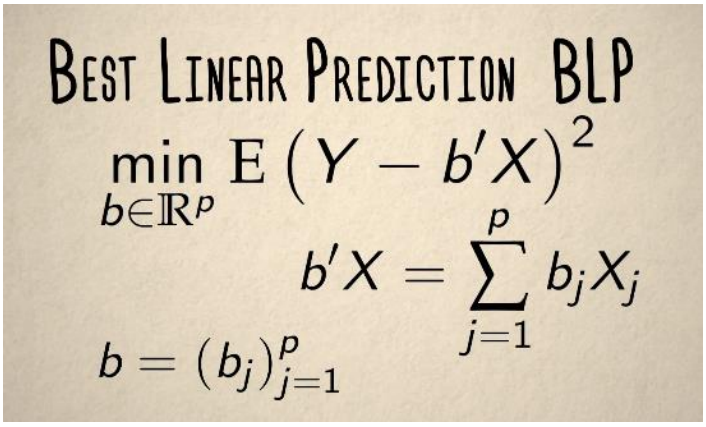


2.1.2 Linear Regression for Prediction

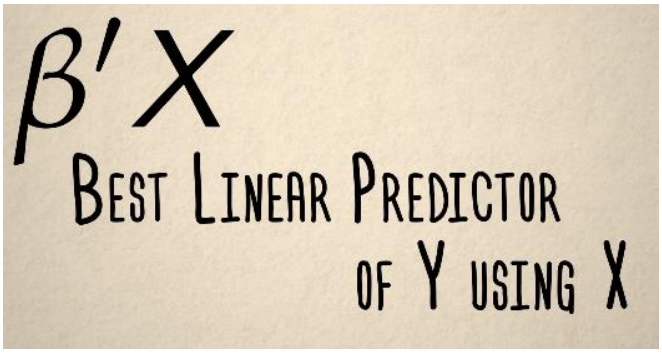


β: *regression parameter*

We minimize the expected or mean squared error that results from predicting Y, using linear rule given by $b'X$, Where b denotes a potential value for the parameter vector β.

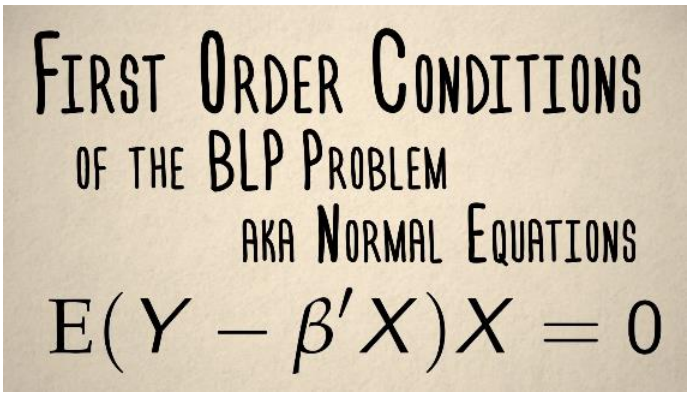


The solution, $\beta'X$, is called the Best Linear Predictor of Y using X.

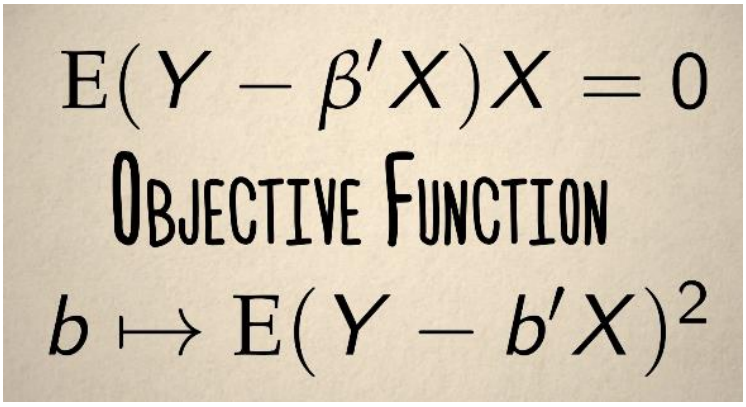


In words, this solution is the best linear prediction rule among all the possible linear prediction rules.

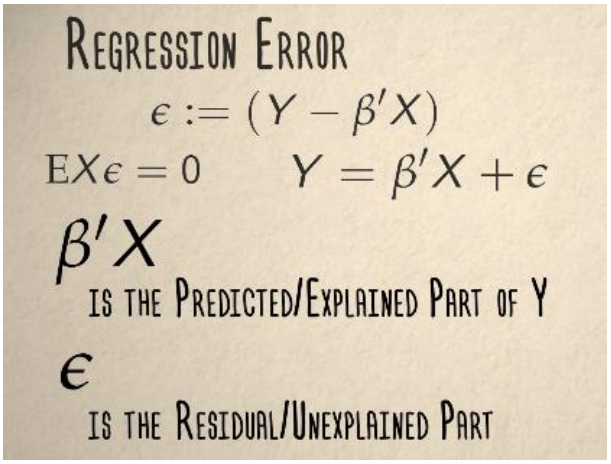
Next, we can compute an optimal beta by solving the first order conditions of the BLP problem, called the Normal Equations, where we have expected value of $(Y - \beta'X) = 0$



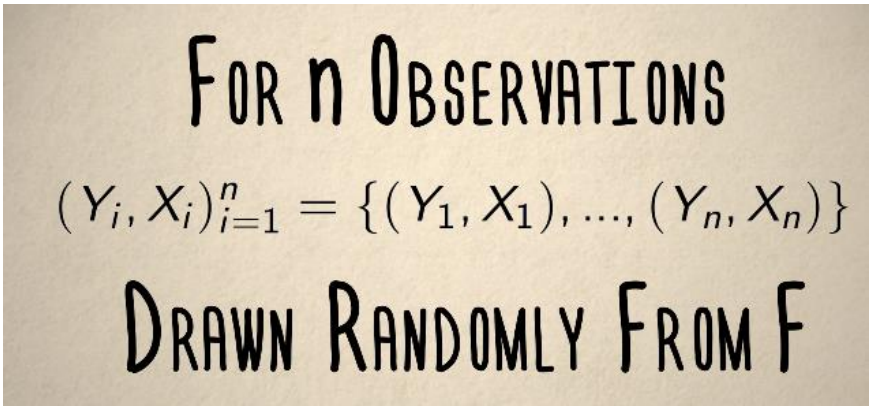
Minimize the objective function by setting the derivative to 0.



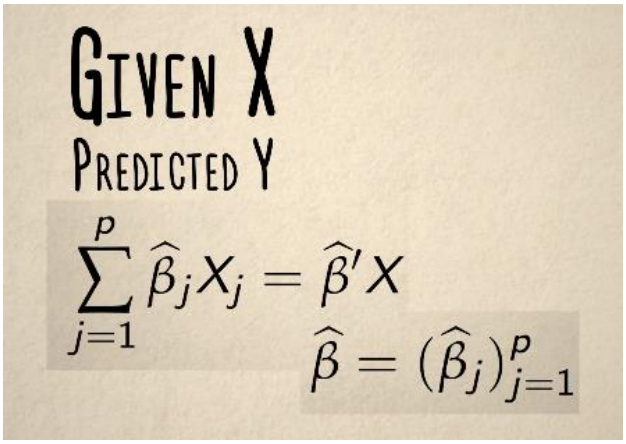
Regression error: ϵ is uncorrelated to X.



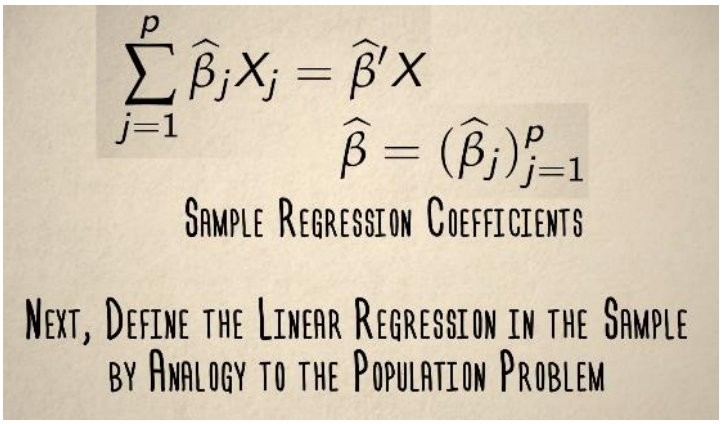
In practice, we don't have access to the population data. Instead we observe a sample of observations of size n,



We assume that observations are generated as a random sample, from a distribution F, which is the population distribution of R, Y, and X. Formally this means that the observations are obtained as realizations of independently and identically distributed copies of the random vector Y, X.
We next construct the best linear prediction rule in sample for Y using X.



$\hat{\beta}$: sample regression coefficients



Next, Define the Linear Regression in the Sample by Analogy to the Population Problem

$\widehat{\beta}$ is Any Solution to the BLP or Ordinary Least Squares Problem in the Sample

$$\min_{b \in \mathbb{R}^p} \mathbb{E}_n \left(Y_i - b' X_i \right)^2$$

Optimal $\widehat{\beta}$

Sample Normal Equations

$$\mathbb{E}_n X_i (Y_i - \widehat{\beta}' X_i) = 0$$

First Order Conditions for the BLP

In-Sample Regression Error

$$\widehat{\epsilon}_i := (Y_i - \widehat{\beta}' X_i)$$

First Order Conditions for the Sample BLP

Decomposition

$$Y_i = \widehat{\beta}' X_i + \widehat{\epsilon}_i$$

Residual/Unexplained Part of Y_i

THE BEST LINEAR PREDICTION RULE (OUT-OF-SAMPLE) IS $\beta' X$.
DOES $\widehat{\beta}' X$ APPROXIMATE $\beta' X$?

WE ARE TRYING TO ESTIMATE P PARAMETERS $\beta_1, \ldots, \beta_p$,
WITHOUT IMPOSING ANY ASSUMPTIONS ON THESE PARAMETERS.

INTUITIVELY, TO ESTIMATE EACH PARAMETER WELL, WE NEED MANY OBSERVATIONS PER PARAMETER.

THIS MEANS THAT N/P MUST BE LARGE, OR,
EQUIVALENTLY P/N MUST BE SMALL.

Theorem
Under regularity conditions

$$\sqrt{\mathrm{E}_X(\beta' X - \widehat{\beta}' X)^2} \leqslant const \cdot \sqrt{\mathrm{E}\epsilon^2} \sqrt{\frac{p}{n}},$$

where E_X *is expectation with respect to* X *and the inequality holds with probability approaching 1 as* $n \to \infty$.

If n is large and p is much smaller than n, for nearly all realizations of the sample, the sample linear regression comes close to the population linear regression.

WE DEFINE LINEAR REGRESSION IN THE POPULATION AND IN THE SAMPLE THROUGH THE BEST LINEAR PREDICTION PROBLEMS SOLVED IN THE POPULATION AND IN THE SAMPLE.

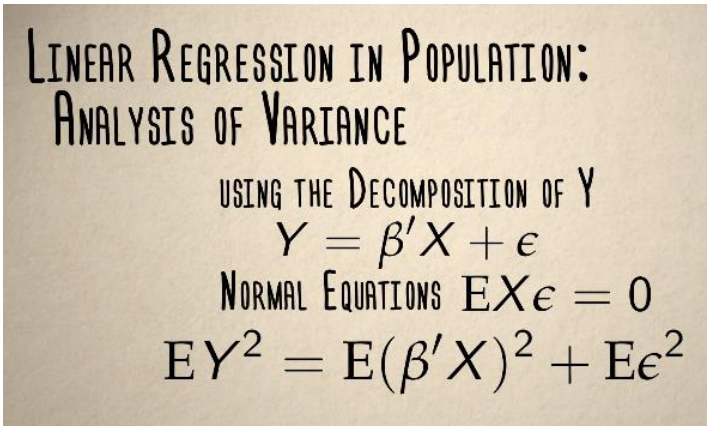
THE SAMPLE LINEAR REGRESSION (BEST LINEAR PREDICTOR) APPROXIMATES THE POPULATION LINEAR REGRESSION (BEST LINEAR PREDICTOR) VERY WELL WHEN THE RATIO P/N IS SMALL.

NEXT, WE WILL DISCUSS THE ASSESSMENT OF PREDICTION PERFORMANCE IN PRACTICE.

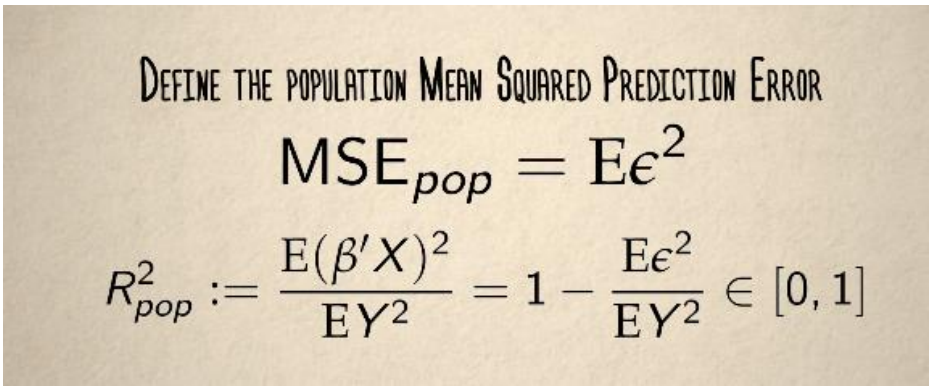
2.1.3 Assessment of Prediction Quality

In this segment, we have two learning goals.
Goal 1 is to understand the analysis of variance, or ANOVA, in the population and in the sample.
Goal 2 is to assess the out of sample predictive performance of the sample linear regression.

We begin with the population case. Using the decomposition of Y into the explain and the residual part. And the normal equations that we derived in the previous segment, we can decompose variation of Y into the sum of explain variation and residual variation, as shown in the formula.

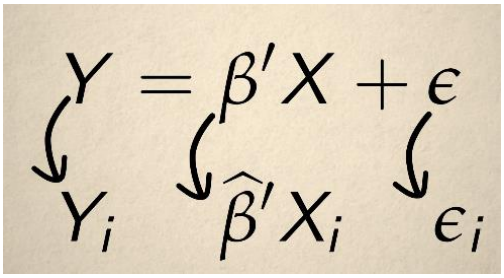


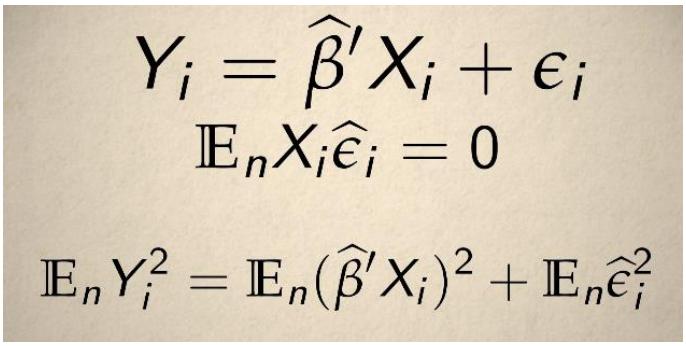
We next define the population mean squared prediction error, or MSE, as the variance of the population residual. The expectation of epsilon squared. We also define the population R squared as the ratio of explained variation to the total variation. In other words, the population R squared is the variation of Y explained by the BLP. And as such, it is bounded below by 0 and above by 1.



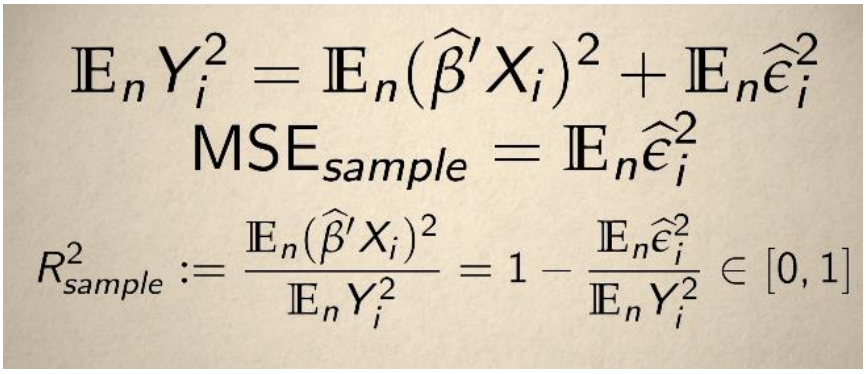
The analysis of variants in the sample process analogous. We simply replace population expectations by the empirical expectations. Using the decomposition of

Yi to explain and the residual part, and the normal equations for the sample. We can decompose the sample variation of Yi into the sum of explained variation and residual variation. The former is given by the sample variation of this sample best in a predictor. And the latter is given by the sample variation of the residual.



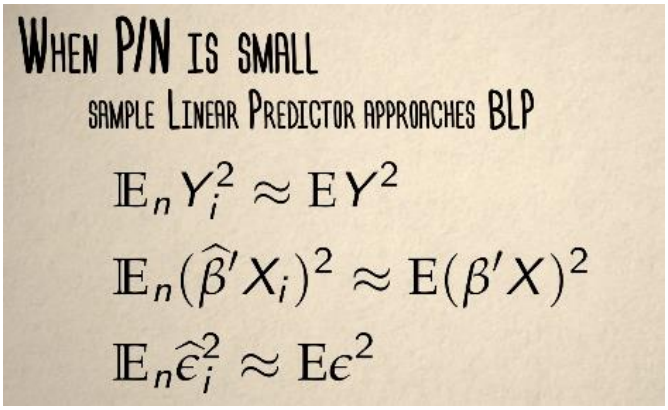


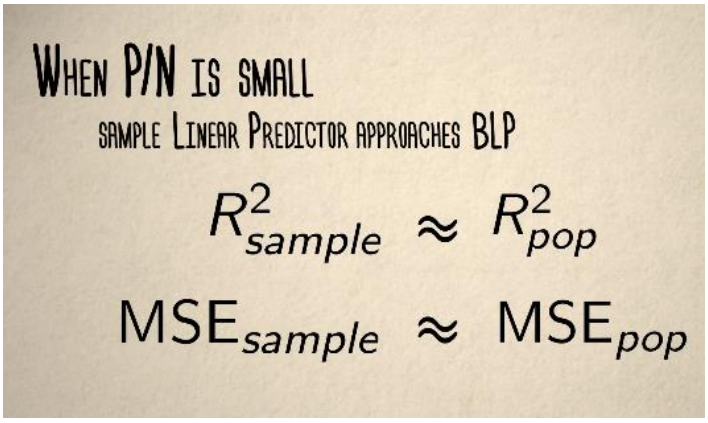
We can define the sample MSE as the sample variance of the residuals and we can define the sample R squared as the ratio of the X-plane to the total variation in the sample.



We know that when P/N is small, the sample linear predictor gets really close to the best linear predictor. Thus, when P/N is small, we expect that sample averages of y squared beta hat xi squared epsilon hat i squared to be close to the population averages of Y squared, beta X squared, and epsilon squared. So in this case, the

sample R squared and the sample MSE will be close to the true quantities, the population R squared and MSE.





When P / N is not small, the discrepancy between the two sets of measures can be substantial. And the sample R squared and the sample MSE can be very poor measures of predictability.

The following simulation example will support our reason. In this example, Y and X are statistically independent and generated from the normal distributions with mean need 0 and variance 1. The means that the true linear predictor of Y given X is simply 0 and the true R squared is also 0. Suppose the number of observations is N, the number of regressors is P.
If p = n, then the typical sample R squared will be 1. Which is very far away from the true number of 0. This is an example of extreme over fitting. If p= n / 2 then the typical sample r squared is about half which is still far off from the truth. If p = n / 20, then the typical sample r squared is about 0.05 which is no longer far off from the truth.

$Y \sim N(0,1)$ BLP OF Y IS
$X \sim N(0, I_p)$ $\beta' X = 0$

true $R^2_{pop} = 0$

n - OBSERVATIONS p - NUMBER OF REGRESSOR

n - OBSERVATIONS p - NUMBER OF REGRESSORS

IF $p = n$ THEN R^2_{sample} IS $1 \gg 0$
IF $p = n/2$ THEN R^2_{sample} IS $.5 \gg 0$
IF $p = n/20$ THEN R^2_{sample} IS $.05$

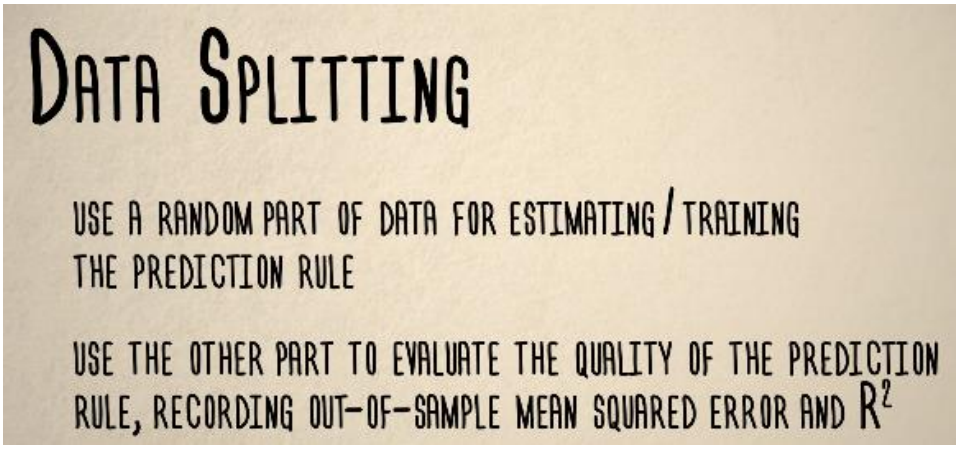
We can now see that using sample R squared and MSE to judge predicted performance could be misleading when P over N is not small. So the question is, can we design better metrics for predicted performance, and the answer is yes.The first proposal is to use the adjusted R squared and MSE.

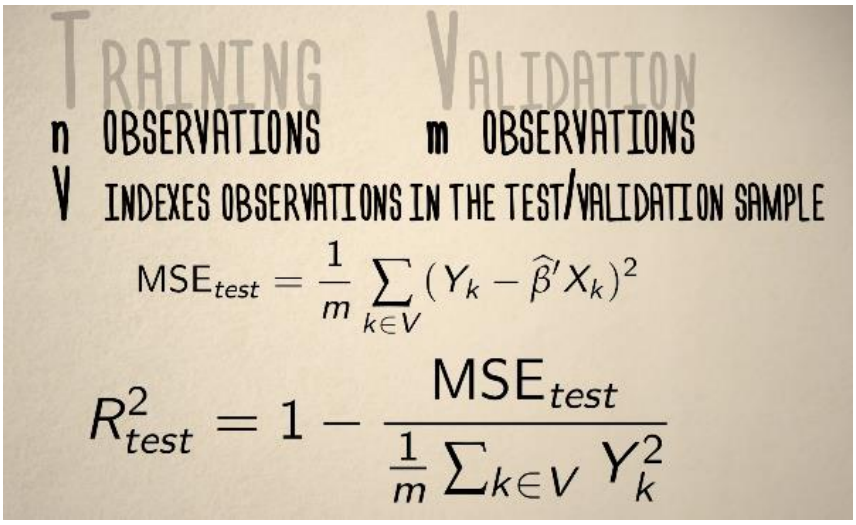
$$R^2_{adjusted} := 1 - \frac{n}{n-p} \frac{\mathbb{E}_n \widehat{\epsilon}_i^2}{\mathbb{E}_n Y_i^2}$$

$$\text{MSE}_{adjusted} = \frac{n}{n-p} \mathbb{E}_n \widehat{\epsilon}_i^2$$

We first define the adjusted MSE by rescaling the sample MSE by a factor of n over n- p.
And then making the corresponding adjustment to the sample R squared. The re-scaling will correct for over-fitting but under rather strong assumptions.

A more universal way to measure predictive performance is to perform data splitting. First, we use a random part of data, say half of data, for estimating or training the prediction rules. Second, we use the other part of data to evaluate the predictive performance or the rule, recording the out-of-sample MSE or R squared. Accordingly, we call the first part of data, the training sample. And the second part, the testing or validation sample.

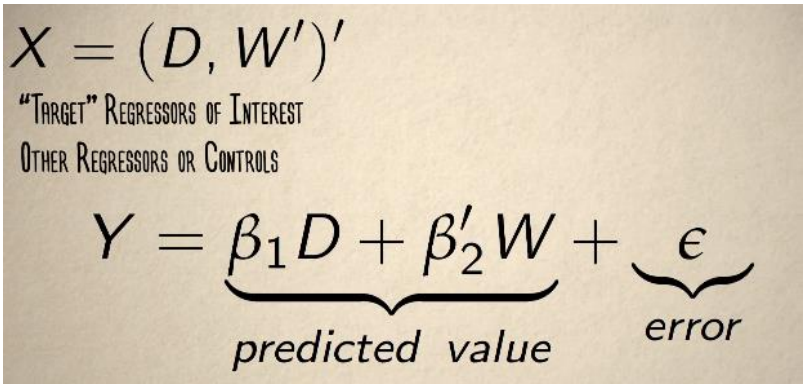




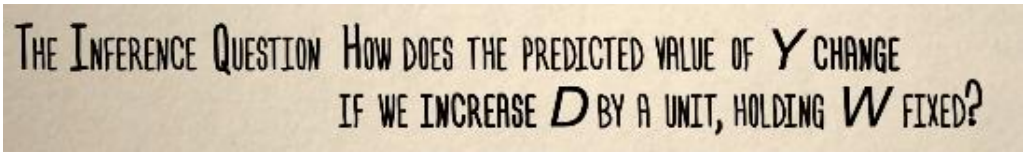
2.1.4 Inference for Linear Regression

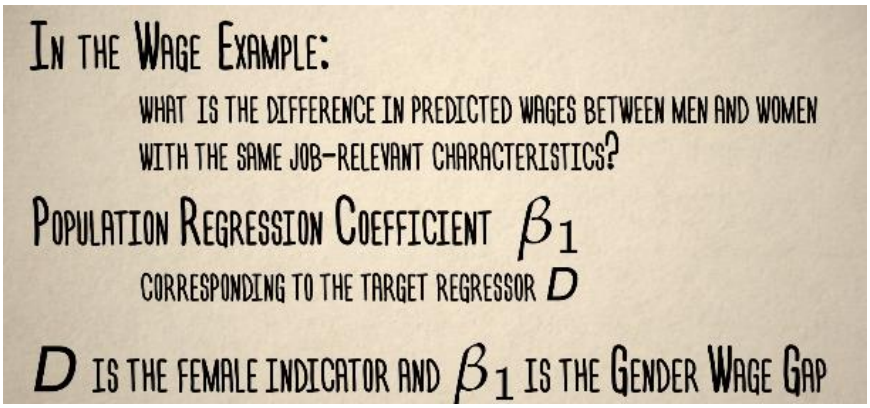
In this segment, we provide an answer to the inference question. To recall the question, we partition vector of regressors X into D and W, where D represents the target regressors of interest, and W represents other regressors, which we sometimes

call the controls. We write Y = the predicted value which is beta 1D + beta 2W plus noise.

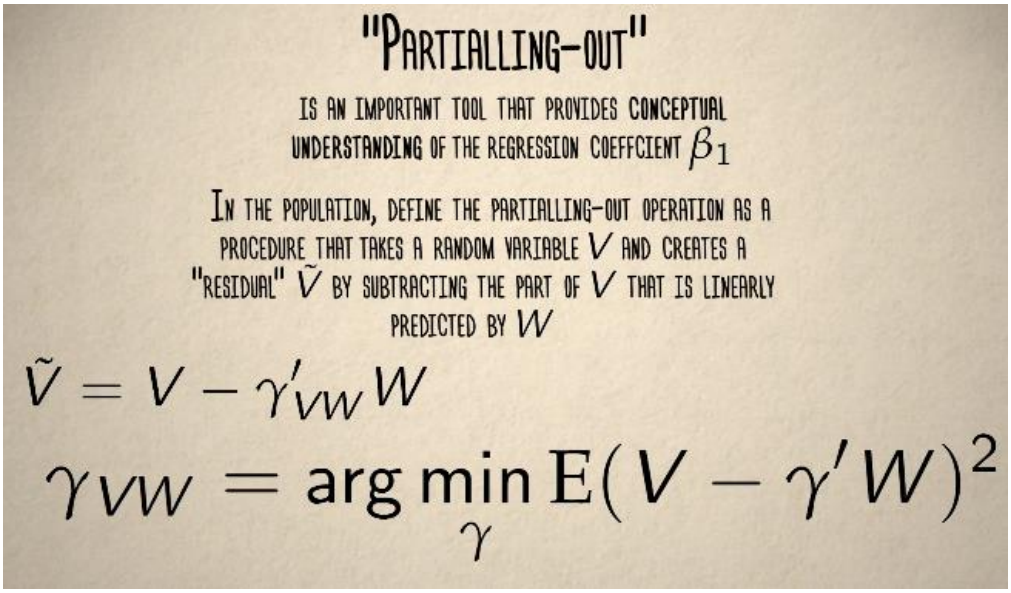


And now we recall the inference question. Which is how does the predicted value of Y change if we increase D by a unit holding W fixed?

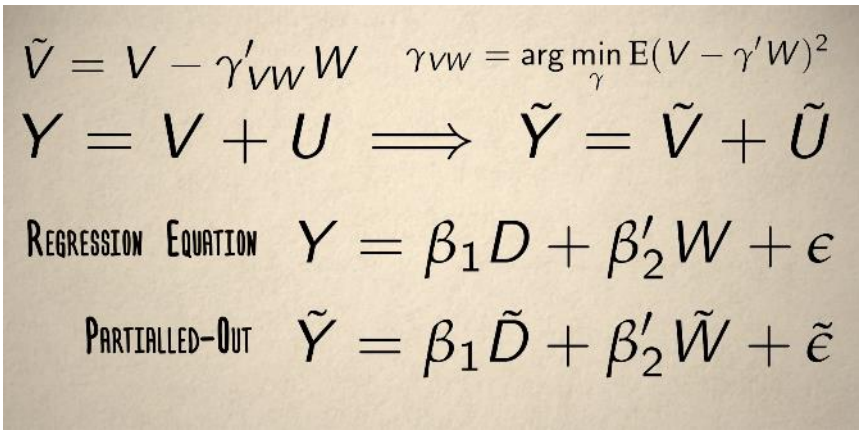




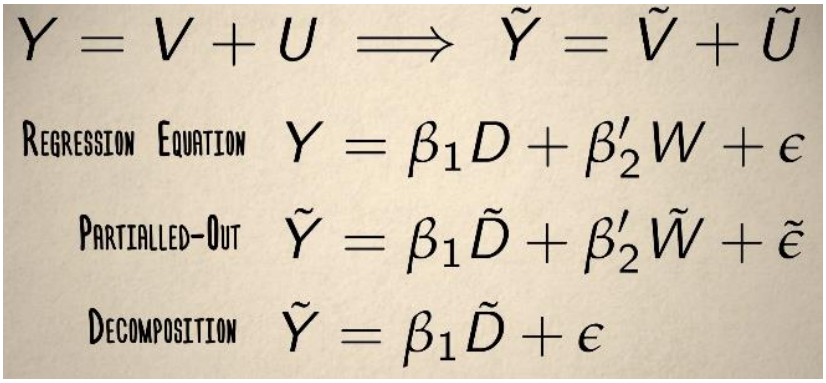
Next， we attempt to understand better the meaning of beta 1 using a tool called ’partialling-out’.



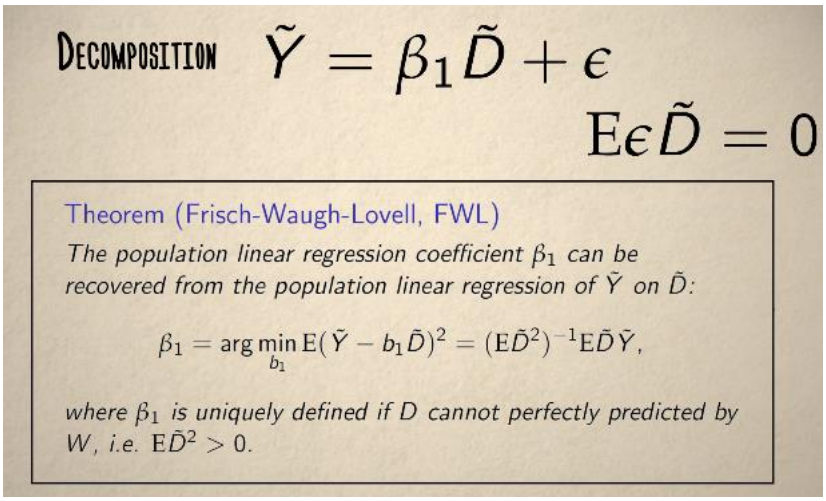
It can be shown that the partialling-out operation is linear.



We now apply partialling-out to both sides of our regression equation to get the partialled-out version which implies that we obtain the decomposition which reads tilde Y = beta 1 tilde D + epsilon, where epsilon is uncorrelated with tilde D.

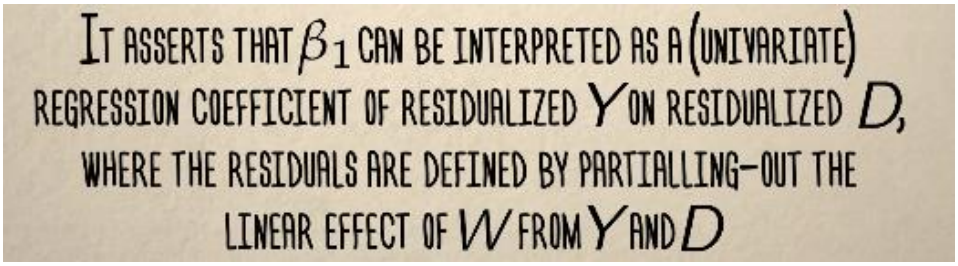


This decomposition follows because partialling out eliminates all the Ws, and also leaves epsilon untouched, since epsilon is linearly unpredictable by X, and therefore by W by definition. Moreover, since tilde D is a linear function of X, epsilon and tilde D are not correlated in our decomposition. Now recognize our decomposition as the normal equations for the population regression of tilde Y and tilde D.



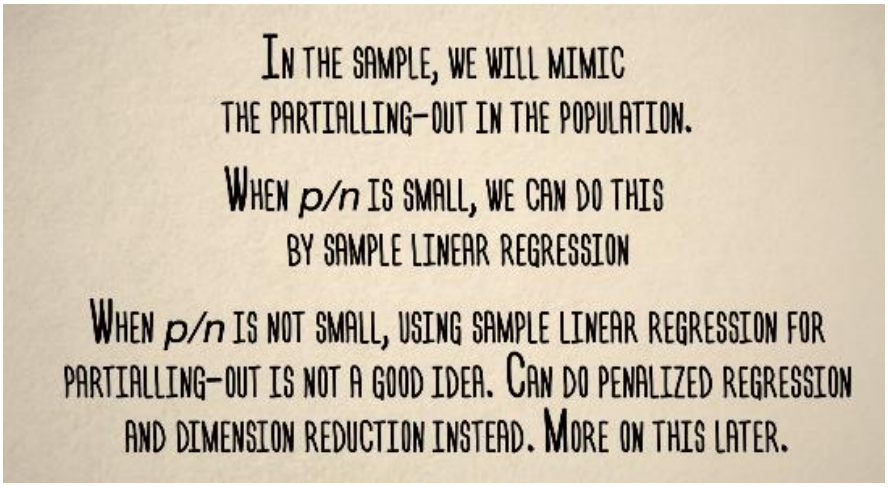
That is, we've just obtained the Frisch-Waugh-Lovell theorem, which states that the population linear regression coefficient beta 1 can be recovered from the population linear regression of tilde Y on tilde D. Beta 1 is a solution to the best linear prediction problem, where we predict tilde Y by a linear function of tilde D. We can also give an explicit formula for beta 1, which is given by the ratio of two averages that you see in this formula. We also see that beta 1 is uniquely defined if D is not perfectly predicted by the Ws. So the tilde D has a non-zero variance.

The Frisch-Waugh-Lovell theorem is a remarkable fact which is useful for both interpretation and estimation.

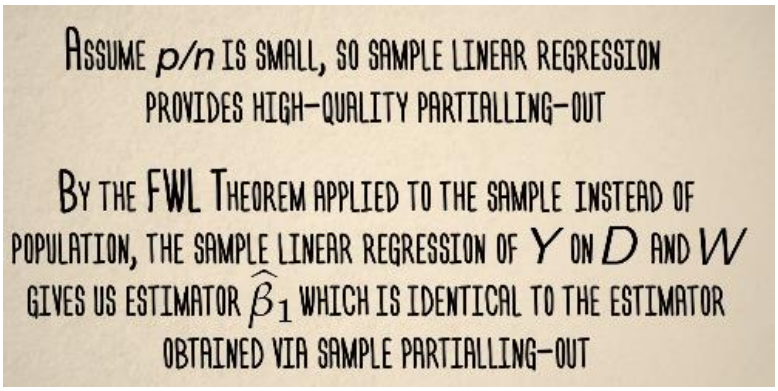


It asserts that beta 1 can be interpreted as a regression coefficient of residualized Y on residualized D, where the residuals are defined by taking out or partialling-out the linear effect of W from Y and D.

We next proceed to discuss estimation of beta 1.



When p over n is small we can rely on the sample linear regression, that is on ordinary in these squares. When p over n is not small using sample linear regression for partialling-out is not a good idea. What we can do instead is do penalized regression or variables selection, which we will discuss later in this module.



It is still useful to give the formula for hat beta 1 in terms of sample partialling-out where we use checked quantities to denote the residuals that are left after predicting the variables in the sample with the controls.

It is still useful to give the formula for $\widehat{\beta}_1$ in terms of sample partialling-out

$$\widehat{\beta}_1 = \arg\min_b \mathbb{E}_n(\check{Y}_i - b\check{D}_i)^2 = (\mathbb{E}_n\check{D}_i^2)^{-1}\mathbb{E}_n\check{D}_i\check{Y}_i$$

where $\check{V}_i$ denote the residual left after predicting V_i with controls W_i. Here we use the sample linear regression

$$\check{V}_i = V_i - \widehat{\gamma}_{VW}'W_i$$
$$\widehat{\gamma}_{VW} = \arg\min_\gamma \mathbb{E}_n(V_i - \gamma' W_i)^2$$

The population partialling-out is replaced here by the sample partialling-out where V replace the population expectation by the empirical expectation.

Using the formula, it can be shown that the following results calls.

Theorem (Inference)

If p/n is small, then the estimation error in $\check{D}_i$ and $\check{Y}_i$ has no first order effect on $\widehat{\beta}_1$, and

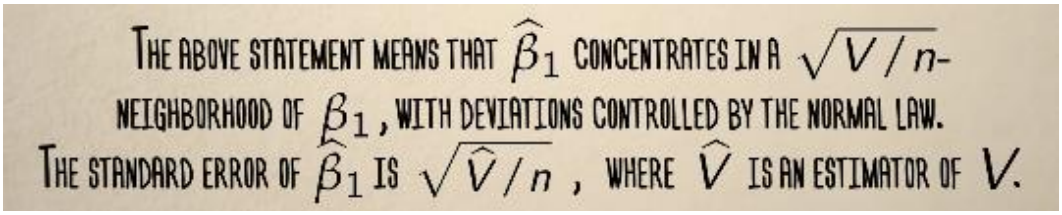
$$\widehat{\beta}_1 \overset{a}{\sim} N(\beta_1, V/n)$$

where

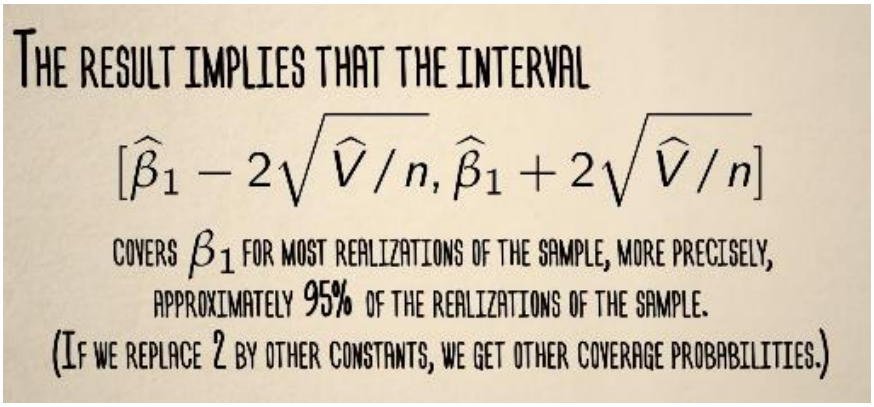
$$V = (\mathrm{E}\tilde{D}^2)^{-1}\mathrm{E}(\tilde{D}^2\epsilon^2)(\mathrm{E}\tilde{D}^2)^{-1}.$$

If p over n is small, then the estimation error in the estimated residualized quantities has a negligible effect on hat beta 1, and hat beta 1 is approximately distributed as normal variable with mean beta 1 and variance V over n where the expression of the variance appears over here. In words, we can say that the estimator hat beta 1 concentrates in a square root of V over n neighborhood of beta 1 with deviations controlled by the normal law.

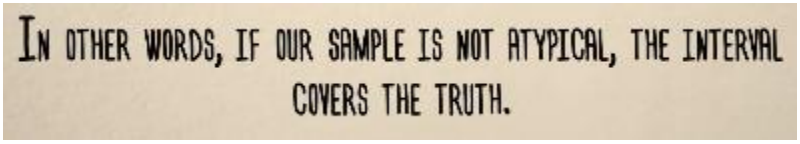
We can now define the standard error for hat beta 1 as square root of hat V over n, where hat V is an estimator of V.

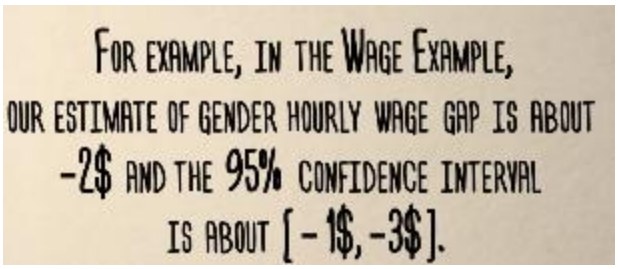


This result implies that the interval given by the estimate +- 2 standard errors covers the true value of beta 1, for most realizations of the data sample. Or more precisely approximately 95% of realizations of the data sample. If we replace 2 by other constants, we get other coverage probabilities. In other words, if our data sample is not extremely unusual, the interval covers the truth.

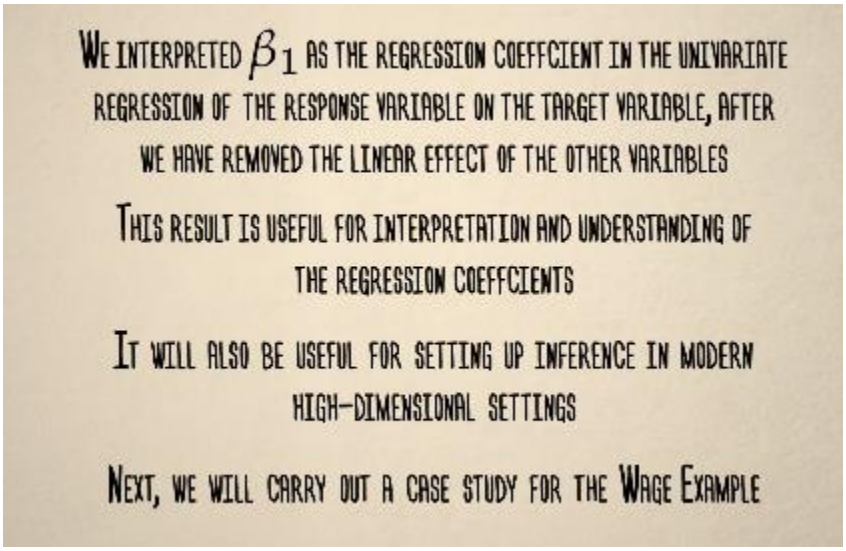


For this reason, this interval is called the confidence interval.





Let us summarize, first we interpreted beta 1 as the regression coefficient in the univariate regression of the response variable and the target variable, after we have removed the linear effect of other variables. Second, we noted that this result is useful for interpretation and
understanding of the regression coefficient. This result will also be super useful for setting up inference in modern high-dimensional settings which we will discussed later in this module. And next, we will carry out a case study for our wage example.



2.1.5 Other Types of Regression

Regressions induced by using
nonlinear functional forms,
for example, logistic regression

Regressions induced by using different types of
loss functions instead of the squared loss function,
for example, median and quantile regression

In nonlinear regression, we use a **nonlinear function** of X, $p(X, \beta)$, to predict Y. For instance, the logistic regression employs

$$p(X, \beta) = \exp(\beta' X) / (1 + \exp(\beta' X))$$

This functional form is particularly attractive when the response variable Y is binary, taking values of 0 and 1, so that the predicted values are naturally constrained between 0 and 1

The problem of predicting binary Y is a basic example of a classification problem. We can interpret the predicted values in this case as approximating probability of $Y=1$

The parameters are estimated either by solving the sample nonlinear least squares problem

$$\min_{b \in \mathbb{R}^p} \mathbb{E}_n (Y_i - p(X_i, b))^2$$

or, in the case of binary outcomes, by maximizing the logistic log-likelihood function

$$\max_{b \in \mathbb{R}^p} \mathbb{E}_n \{ Y_i \ln p(X_i, b) + (1 - Y_i) \ln(1 - p(X_i, b)) \}$$

We previously used the squared error loss to set up the best linear predictor. We can call this approach the linear mean regression, because the best predictor of Y under squared loss is the conditional expectation of Y given X

What if we use the absolute deviation loss instead? Then we will obtain the median regression, because we will implicitly try to estimate the median value of Y conditional on X

$$\min_{b \in \mathbb{R}^p} \mathrm{E} |Y - b'X|$$

In the population we use the population expectation E and in the sample we use the empirical expectation $\mathbb{E}_n$ instead

Just like mean regression, median regression can also be made nonlinear, so we can replace $b'X$ by $p(X, b)$

Median regressions are great for cases when the outcomes have heavy tails or outliers. They are much better behaved in this case than the mean regression

If we use an asymmetric absolute deviation loss, then we end up with the asymmetric absolute deviation regression or quantile regression

$$\min_{b \in \mathbb{R}^p} \mathrm{E}\left(\tau |Y - b'X| 1(Y > b'X) + (1-\tau)|Y - b'X| 1(Y < b'X)\right)$$

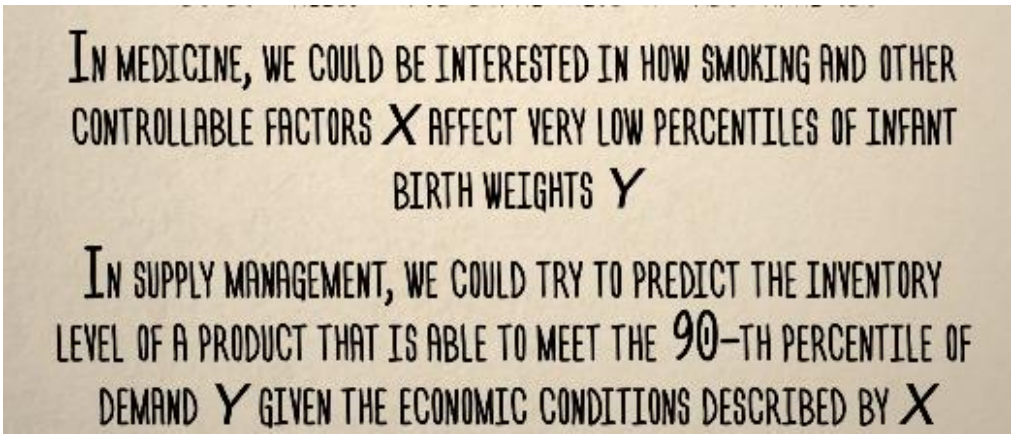
In the population we use the population expectation and in the sample we use the empirical expectation $\mathbb{E}_n$ instead

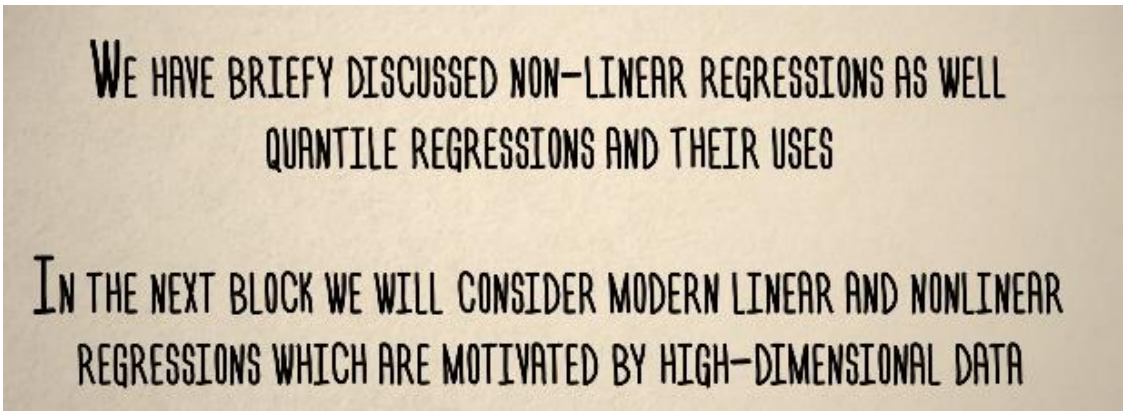
The weight τ, that appears in the definition of the absolute deviation loss, specifies the $\tau \times 100$-th percentile of Y that we are trying to model

Quantile regressions are great for determining the factors that affect the outcome in the trails

For Example

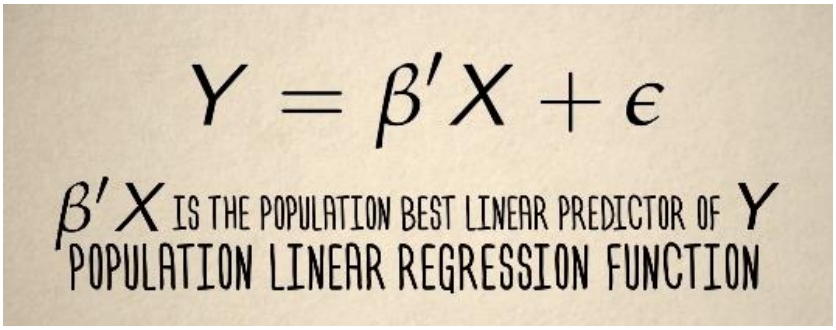
In risk management, we might forecast the extremal conditional percentiles of Y using the information X. This is called conditional value-at-risk analysis



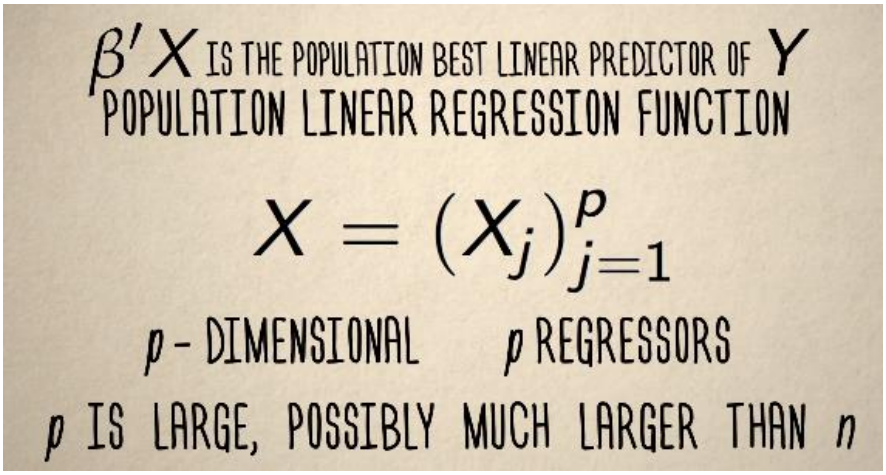


2.2.1 Modern Linear Regression for High-Dimensional Data

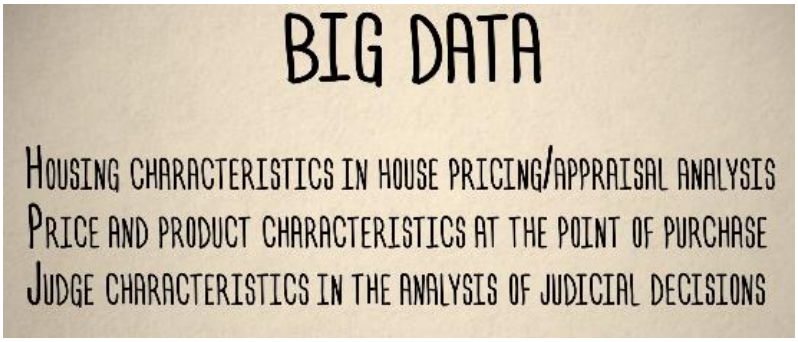
We begin a new block of segments where we consider modern linear regression for high dimensional data. We consider the linear regression model Y equals Beta X plus Epsilon where Beta X is the population best linear predictor, or BLP, of Y using X. Or simply the population linear regression function, and epsilon is uncorrelated with X.



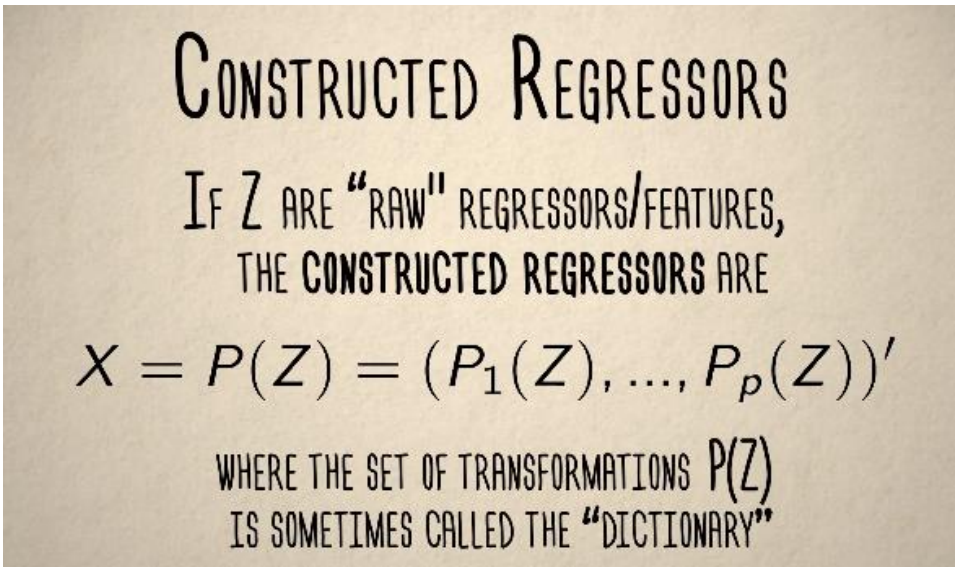
Here, the regressor X is p-dimsensional, that is, there are P regressors and the sample size is N, and P is large, possibly larger than N.

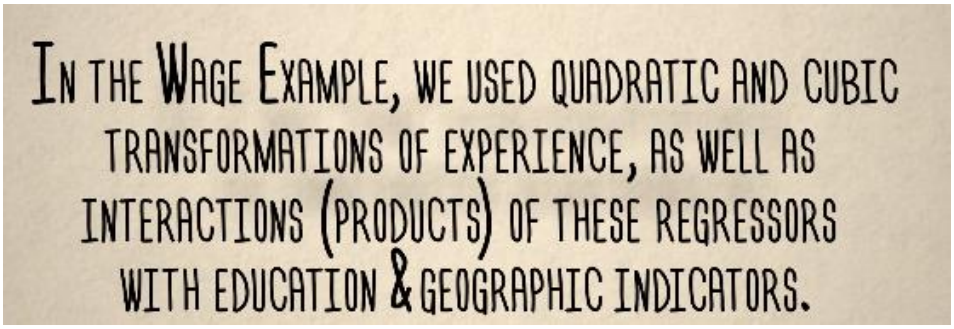


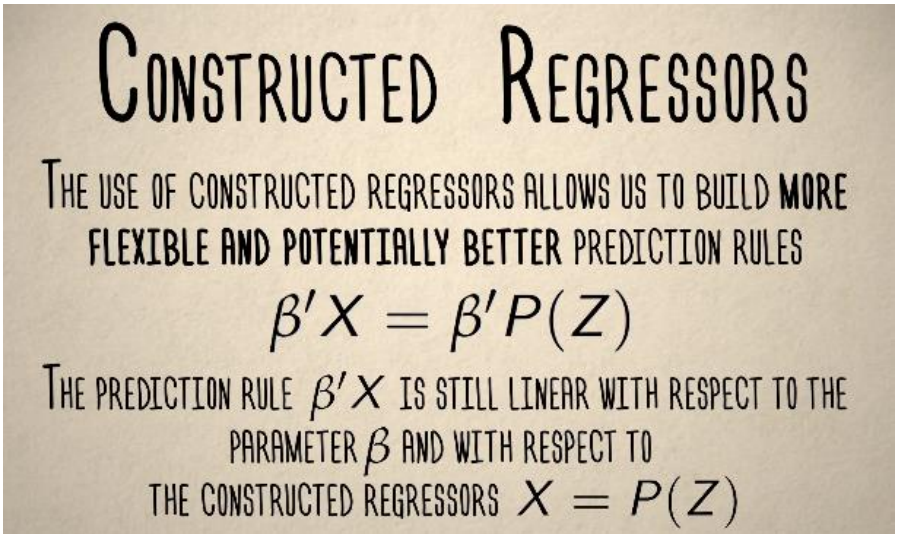
One reason for having high dimensional regessors in our model is the increasing availability of modern rich data, or quote unquote, big data. Many modern data sets are rich in that they have many recorded features or regressors. For example, in house pricing and appraisal analysis, we can make use of numerous housing characteristics. In demand analysis, we can rely on price and product characteristics, and in the analysis of court decisions, we can employ many of the judges' characteristics.



Another reason for having high dimensional regressors in our model is the use of constructed regressors.







What is the best prediction rule for Y using Z? It turns out that the best prediction rule is the conditional expectation of Y given Z. We denote this prediction rule by g of Z and we call it the regression function of Y on Z. This is the best prediction rule among all rules, and it is generally better than the best linear prediction rule.

The Best Prediction Rule

$$g(Z) = \mathrm{E}(Y \mid Z)$$

the conditional expectation of Y given Z

Regression Function of Y on Z

It can be shown that $g(Z)$ solves the best prediction problem:

$$\min_{m(Z)} \mathrm{E}(Y - m(Z))^2$$

where we minimize the mean squared prediction error *(MSE)* among all prediction rules $m(Z)$ (linear or nonlinear in Z)

By using $\beta' P(Z)$ we are implicitly approximating the best predictor $g(Z)$

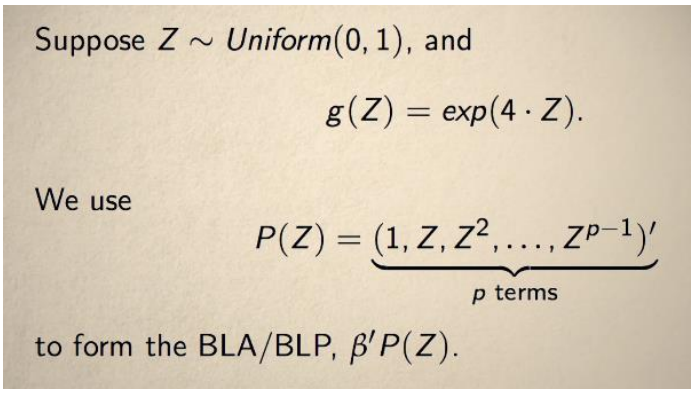
Indeed, it can be shown that for any parameter b

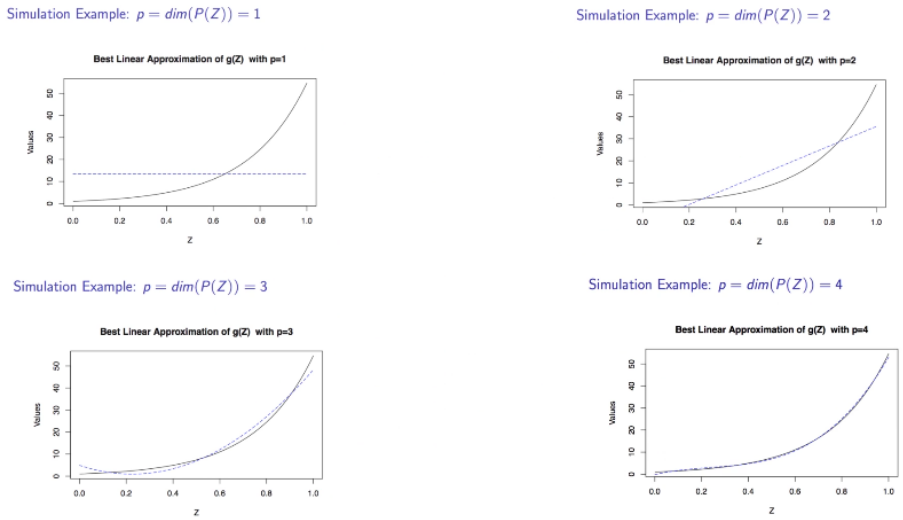
$$\mathrm{E}(Y - b'P(Z))^2 = \mathrm{E}(g(Z) - b'P(Z))^2 + \text{const.}$$

that is, the mean squared prediction error is equal to the mean squared approximation error of $b'P(Z)$ to $g(Z)$, plus a constant that does not depend on B, so that minimizing the former is the same as minimizing the latter

We now conclude that the BLP beta P(Z) is the best linear approximation, or BLA, to the regression function g(Z).

By using a richer and richer dictionary of transformations, P(Z), we can make the BLP beta P(Z) approximate g(Z) better and better. We can illustrate this point by the following simulations.

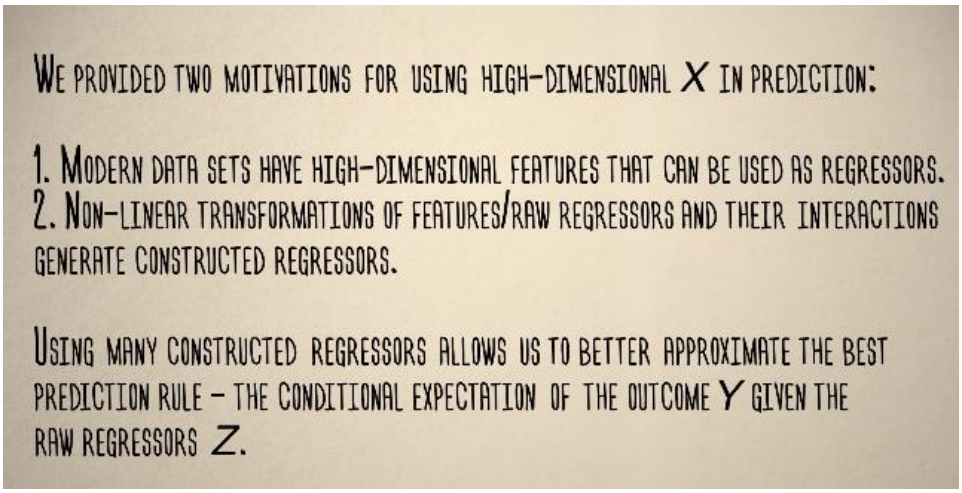




Suppose that is uniformly distributed on the unit interval, and the true regression function g(Z) is the exponential function of 4 times Z. Suppose we don't know this and we use the linear form beta P(Z) to provide approximations for g(Z)? Suppose we use P(Z) that consists of polynomial transformations of Z, consisting of the first P terms of the polynomial series 1, Z, Z squared, Z cubed, and so on. We use this dictionary to form the BLA or BLP beta P(Z). We now provide a sequence of figures that illustrate the approximation properties of the BLA or BLP corresponding to P(Z) equal to 1, 2, 3 and 4.

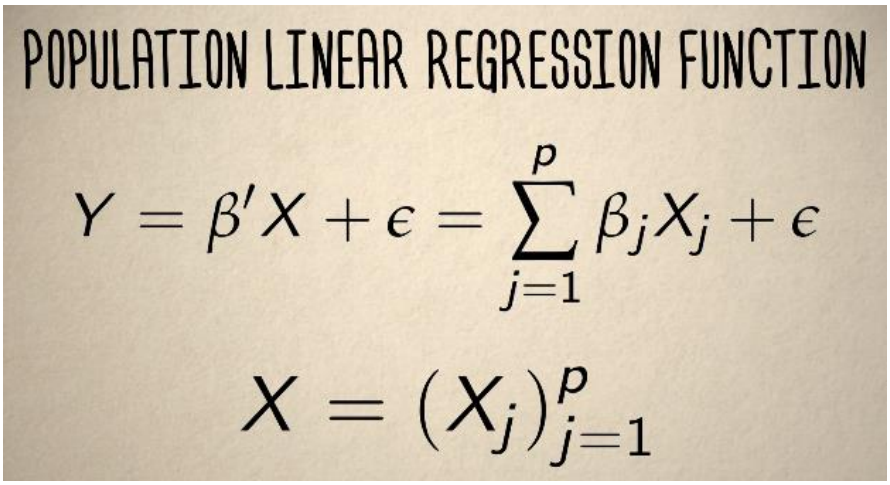
With P equal 1, we get a constant approximation to the regression function g(Z), and as we can see, the quality of this approximation is very poor. With P = 2, we get a

linear inside approximation for g(Z), and as the figure shows, the quality of this approximation is still very poor. With P = 3, we get a quadratic in Z approximation for g(Z), and now, the quality is quite good all of a sudden. This explains why using the linear transformations over [INAUDIBLE] regressors is a good idea. Now with P equals 4 we get a cubic in Z approximation for g(Z) and the quality of approximation becomes simply excellent.



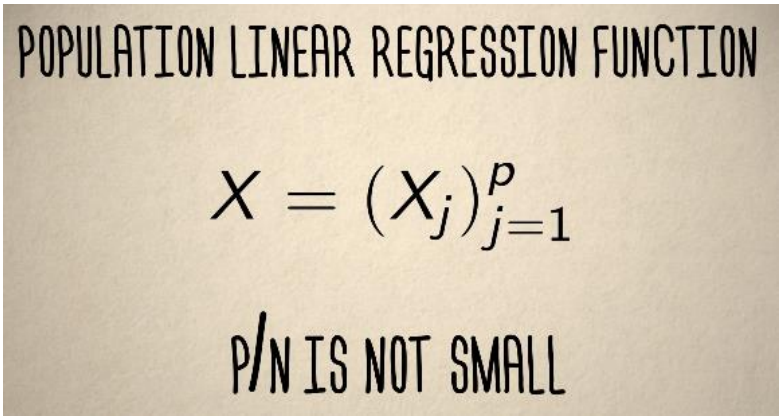
2.2.2 High-Dimensional Sparse Models and Lasso

In this segment, we will talk about high-dimensional sparse models and a penalized regression method called the Lasso.

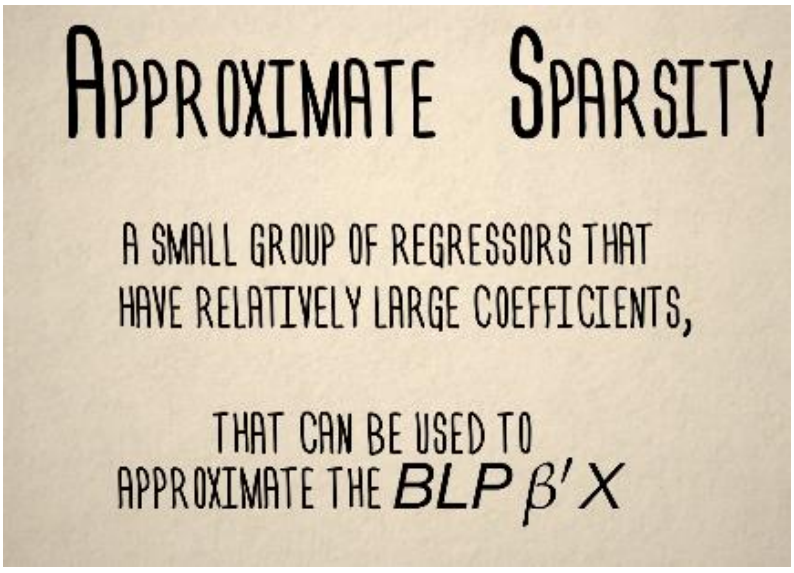


Here, we consider the regression model Y = beta X + epsilon, where beta X is a population-best linear predictor of Y using X. Or simply the population linear regression function.

The regresssor X is p-dimensional, with components denoted by Xj. That is, there are p regressors, and p is large, possibly much larger than N, where N is the sample size.



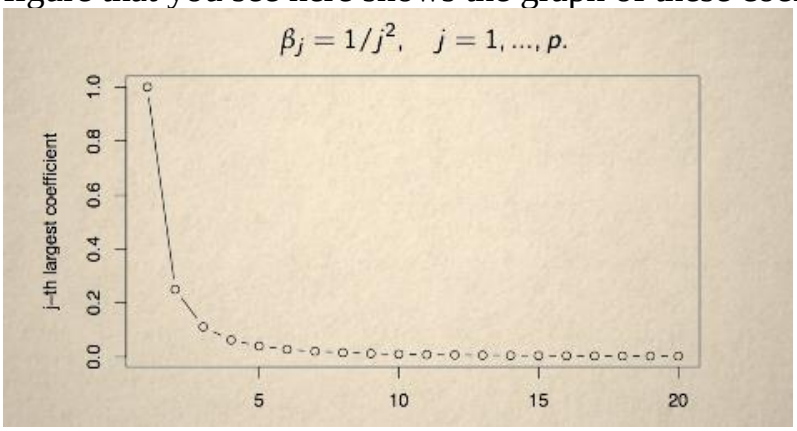
In order to simplify the discussion, we assume that each Xj has a unit variance in the sample.
Here, we are dealing with the high-dimensional setting, where p/N is not small. Classical linear regression, or ordinary least squares fails in these settings, because it overfits the data, as we have seen in part one of our module. We need to make some assumptions and modify the classical regression method to deal with the high-dimensional case.



One intuitive assumption is approximate sparsity, which informally means that there is a small group of regressors that have relatively large coefficients, that can be used

to approximate the BLP beta 'X quite well. The rest of regressors have relatively small coefficients.

An example of an approximately sparse model is the linear regression model with the regression coefficients beta j given by 1/j-squared as j ranges from 1 to p. The figure that you see here shows the graph of these coefficients.

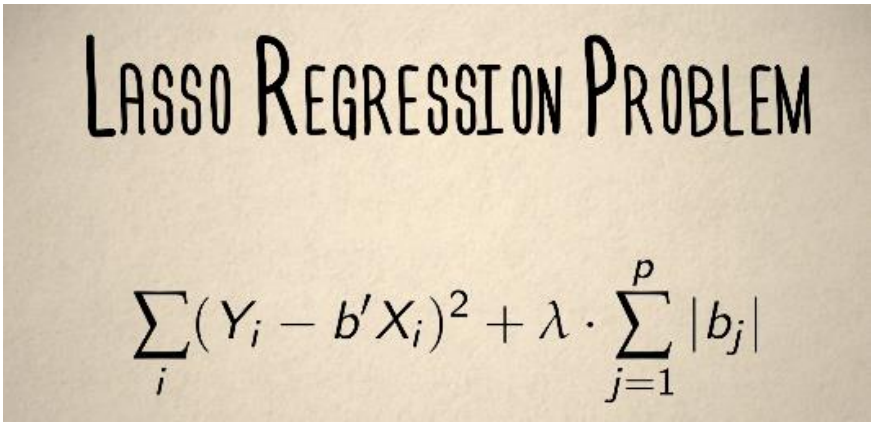


As we can see, the coefficients has decreased quite fast, with only three or four regression coefficients that appear to be large.

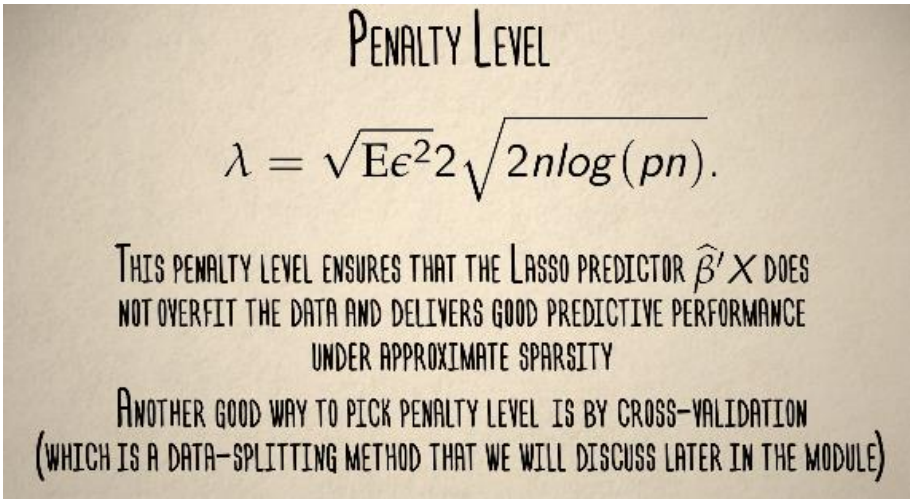
$$|\beta|_{(j)} \leqslant Aj^{-a}, \quad a > 1/2,$$

Formally, the approximate sparsity means that the sorted absolute values of the coefficients decrease to zero fast enough. Namely, the js largest in absolute value coefficient is at most of size j into the power of minus a times a constant, where a is greater than one half. Here, the constant a measures the speed of decay.

For estimation purposes, we have a random sample of Yi and Xi, where i ranges from 1 to n. We seek to construct a good linear predictor, head-beta X, which works well when p over N is not small. We can construct head-beta as a solution of the following penalized regression problem called Lasso.



Here, we are minimizing the sample mean-squared error that results from predicting Yi with bX plus a penalty term, which penalizes the size of the coefficients, bjs, by their absolute values. We control the degree of penalization by the penalty level lambda. A theoretically justified penalty level for Lasso is given by the formula that you see here.



Another good way to pick penalty level is by cross-validation, which uses repeated splitting of data into training and testing samples to measure predictive performance.

LASSO IMPOSES THE APPROXIMATE SPARSITY ON THE COEFFICIENTS $b = \widehat{\beta}$, JUST LIKE IN THE ASSUMPTION

IT PRESSES DOWN ALL OF THE COEFFICIENTS TO ZERO, AS MUCH AS POSSIBLE AND AT AN EQUAL RATE, WITHOUT SACRIFICING TOO MUCH FIT, AND IT ENDS UP SETTING MANY OF THESE COEFFICIENTS TO ZERO

Simulation Example:

$$Y = \beta' X + \epsilon, \quad X \sim N(0, I_p), \quad \epsilon \sim N(0, 1),$$

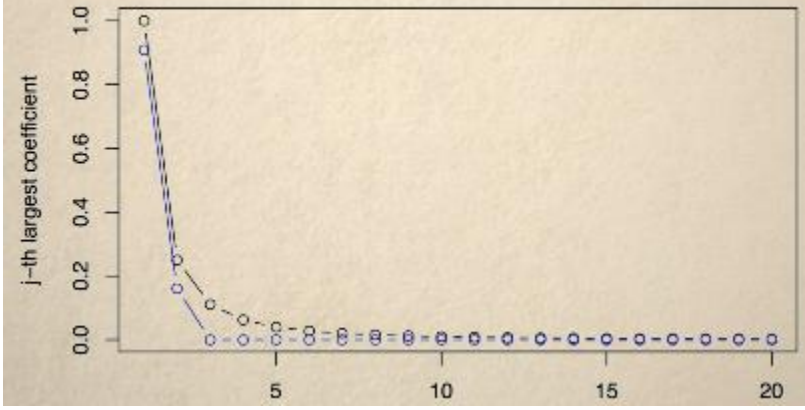
where

$$\beta_j = 1/j^2, \quad j = 1, \ldots, p$$

and

$$n = 300, \quad p = 1000.$$

The figure shows that $\widehat{\beta}$ is sparse and is close to β.

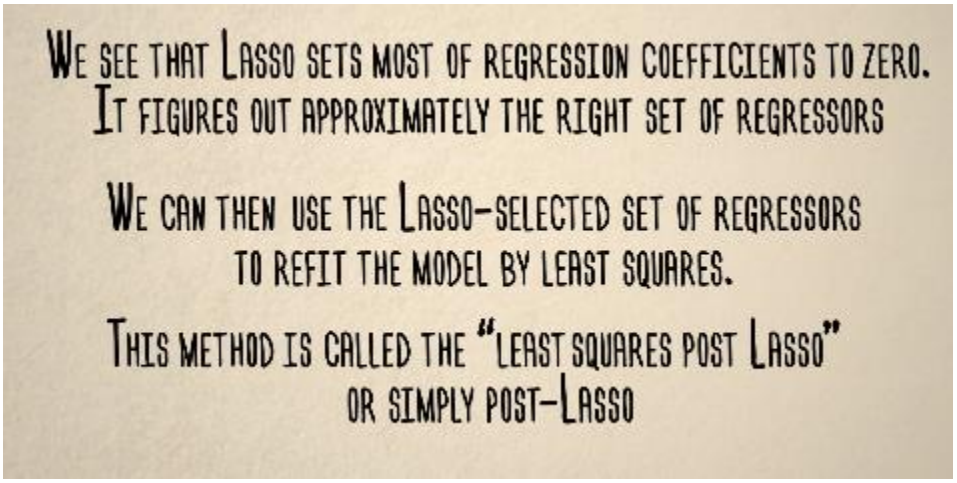


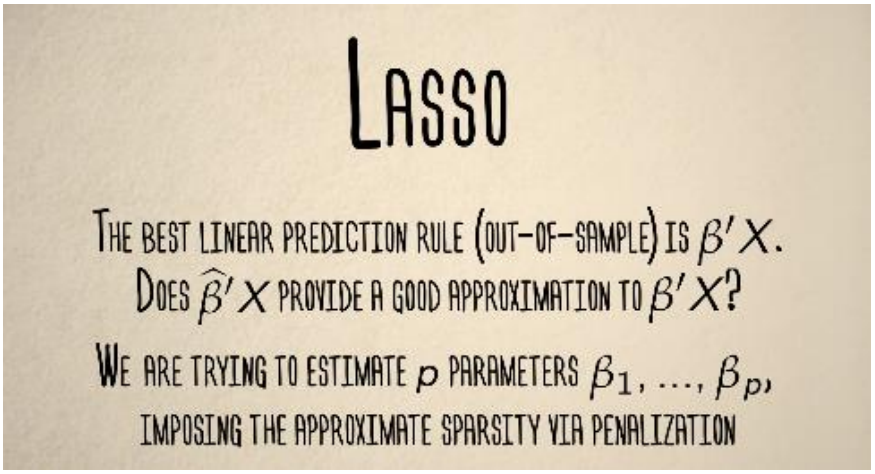
The figure shows that hat beta, in blue, is indeed sparse and is close to the true coefficient vector beta, in black, indeed. Most of hat-beta js are set to 0, except for several coefficients that align quite well with the largest coefficient beta j, or the true coefficient vector.

This really shows that Lasso is able to leverage approximate sparsity to deliver good approximation to the true coefficients.

From the figure, we see that Lasso sets most of the regression coefficients to zero. It basically figures out approximately, though not perfectly, the right set of regressors.

In practice, we often use the Lasso-selected set of regressors to refit the model by least squares. This method is called the least squares post Lasso, or simply post-Lasso.





Lasso

Under sparsity, only a few, say s, parameters will be "important".
We can call s the **effective dimension**

Lasso will approximately figure out which parameters are important

Intuitively, to estimate each of the "important" s parameters well, we need many observations per such parameter

Lasso

This means that n/s must be large, or, equivalently s/n must be small

Theorem

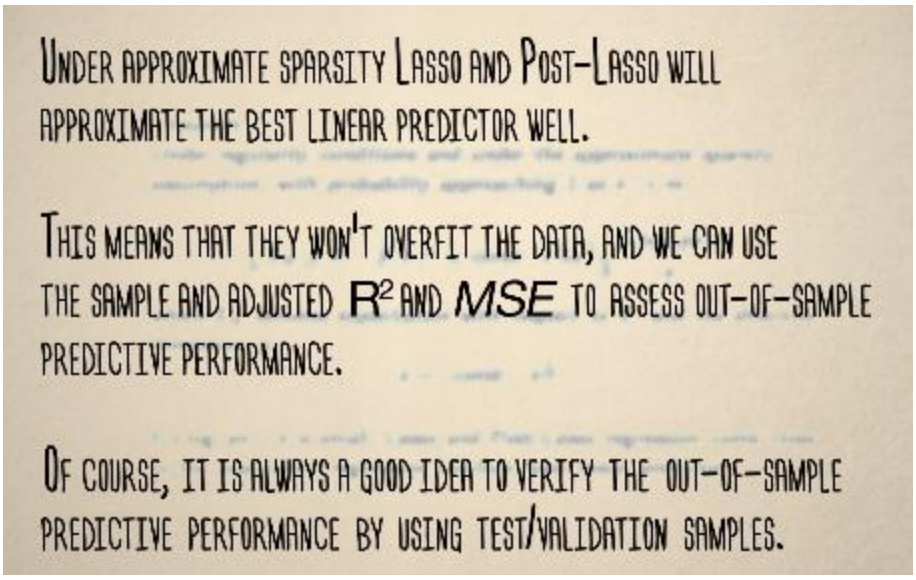
Under regularity conditions and under the approximate sparsity assumption, with probability approaching 1 as $n \to \infty$,

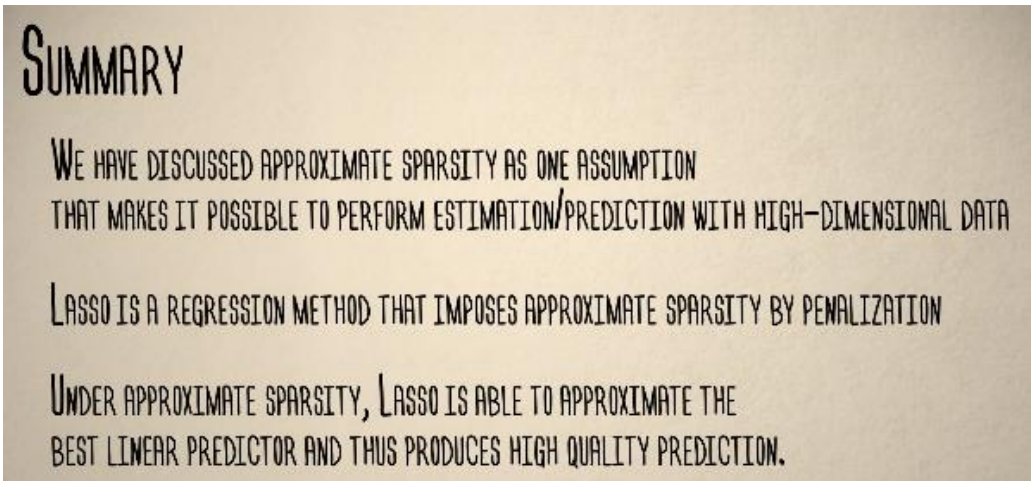
$$\sqrt{\mathrm{E}_X(\beta' X - \widehat{\beta}' X)^2} \leqslant const \cdot \sqrt{\mathrm{E}\epsilon^2} \sqrt{\frac{s \log(pn)}{n}},$$

where E_X denotes expectation with respect to X, and the effective dimension is

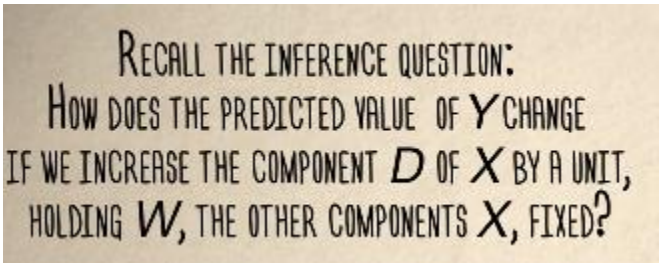
$$s = const \cdot n^{\frac{1}{2a}}.$$

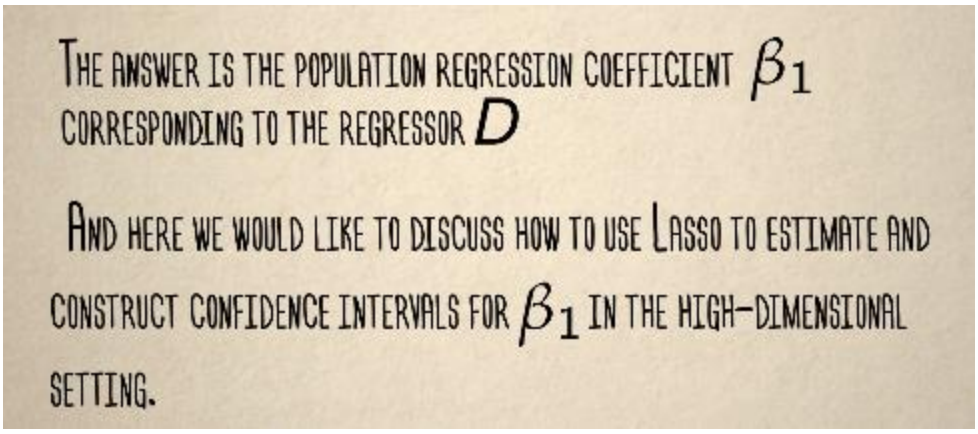
If $s \log(pn)/n$ is small, Lasso and Post-Lasso regression come close to the population regression function/best linear predictor.





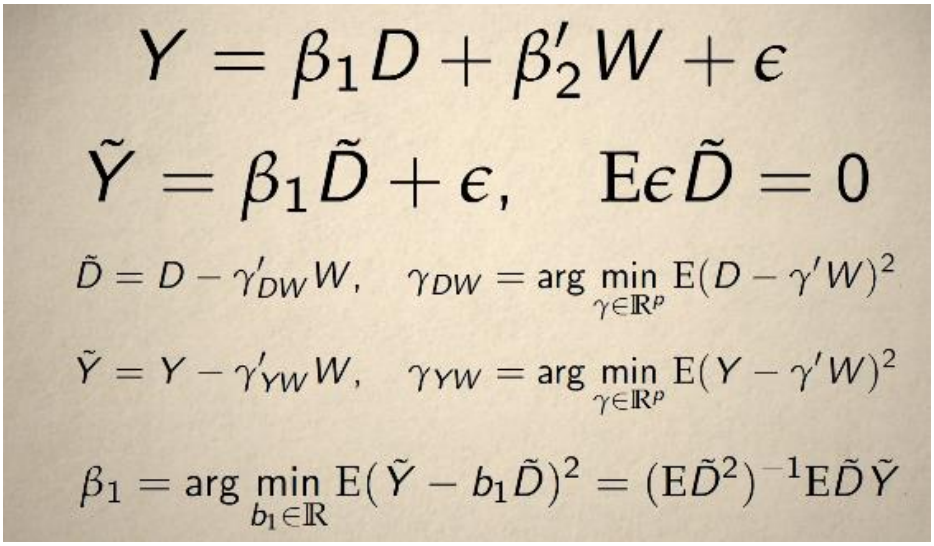
2.2.3 Inference with Lasso





We write the regression equation as Y equals beta 1D plus beta 2 w plus epsilon, where d is the target regressor and w's our d controls. We recall from part one of our module that after portioning out we ended up with a simplified regression equation. Tilda Y = tilda d beta 1 + epsilon where epsilon is uncorrelated with tilda d. Here, tilde Y and tilde D, are the residuals that are left after parceling out the linear effect of W.

As we argued in part one, this allows us to obtain beta 1 as a coefficient in the linear regression of tilde Y and tilde D. D in the population recovers the partialling-out procedure. For estimation purposes, we have a random sample of Yis and Xis. Our main idea here is that, we will mimic in the sample the partialling-out procedure in the population.



$$(Y_i, X_i)_{i=1}^n$$

Our main idea here is that, we will mimic in the sample the partialling-out procedure in the population.

Previously, when p over n was small, we employed least squares as the prediction method in the partialling-out steps.

Here p over n is not small, and we employ instead the Lasso method in the partialling-out steps.

So let us explain this in more detail. First we run the lasso regression of Yi on Wi and of Di on Wi and keep the resulting residuals called check Wi and check Di.

$$\widehat{\gamma}_{YW} = \arg\min_{\gamma \in \mathbb{R}^p} \sum_i (Y_i - \gamma' W_i)^2 + \lambda_1 \sum_j |\gamma_j|$$

$$\widehat{\gamma}_{DW} = \arg\min_{\gamma \in \mathbb{R}^p} \sum_i (D_i - \gamma' W_i)^2 + \lambda_2 \sum_j |\gamma_j|$$

$$\check{Y}_i = Y_i - \widehat{\gamma}'_{YW} W_i$$

$$\check{D}_i = D_i - \widehat{\gamma}'_{DW} W_i$$

Second we run the list squares of check Wi on check Di. The resulting estimated regression confusion is our estimator check beta one.

$$\check{\beta}_1 = \arg\min_{b_1 \in \mathbb{R}} \mathbb{E}_n(\check{Y}_i - b_1 \check{D}_i)^2 = (\mathbb{E}_n \check{D}_i \check{D}_i)^{-1} \mathbb{E}_n \check{D}_i \check{Y}_i$$

Our portioning out procedure was loss all. It relies on approximate sparcity in the two portioning out steps. Indeed, theoretically, the procedure will work well if the population regression coefficients and the two parceling out steps are approximately sparse, with a sufficiently high speed of degrees of assorted values, as shown in the conditions that you see here.

We need approximate sparsity of the population regression coefficients γ_{YW} and γ_{DW}, with a sufficiently high speed of decrease in the sorted coefficients, namely,

$$|\gamma_{YW}|_{(j)} \leqslant Aj^{-a}, \quad |\gamma_{DW}|_{(j)} \leqslant Aj^{-a} \quad a > 1, \quad j = 1, \ldots, p$$

Remark. Approximate sparsity of γ_{YW} follows from approximate sparsity of β_2, namely

$$|\beta_2|_{(j)} \leqslant Aj^{-a}, \quad a > 1, \quad j = 1, \ldots, p$$

Theorem (Inference in High-Dimensional Regression)

Under the stated approximate sparsity and other regularity conditions, the estimation error in $\check{D}_i$ and $\check{Y}_i$ has no first order effect on $\check{\beta}_1$, and

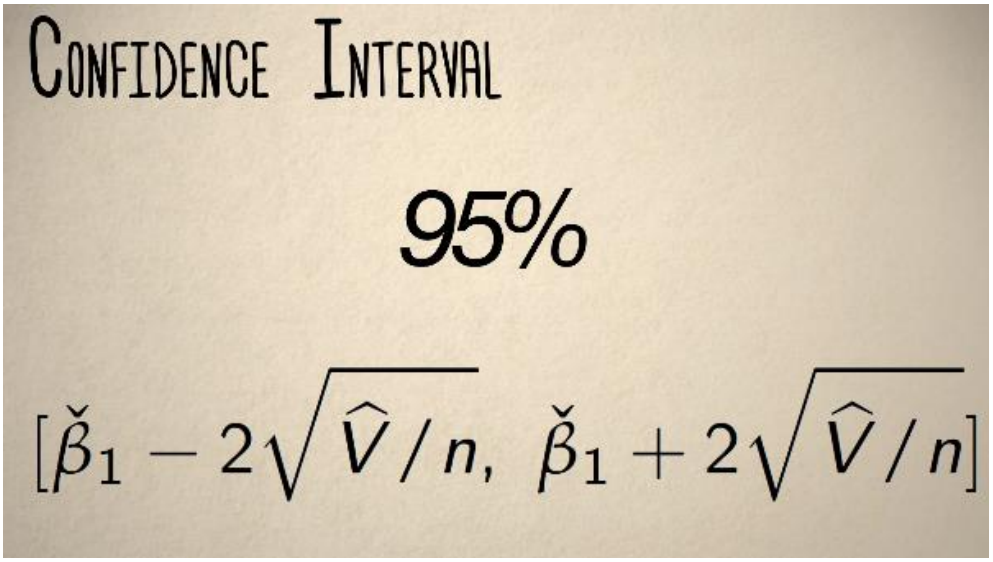
$$\check{\beta}_1 \overset{a}{\sim} N(\beta_1, V/n)$$

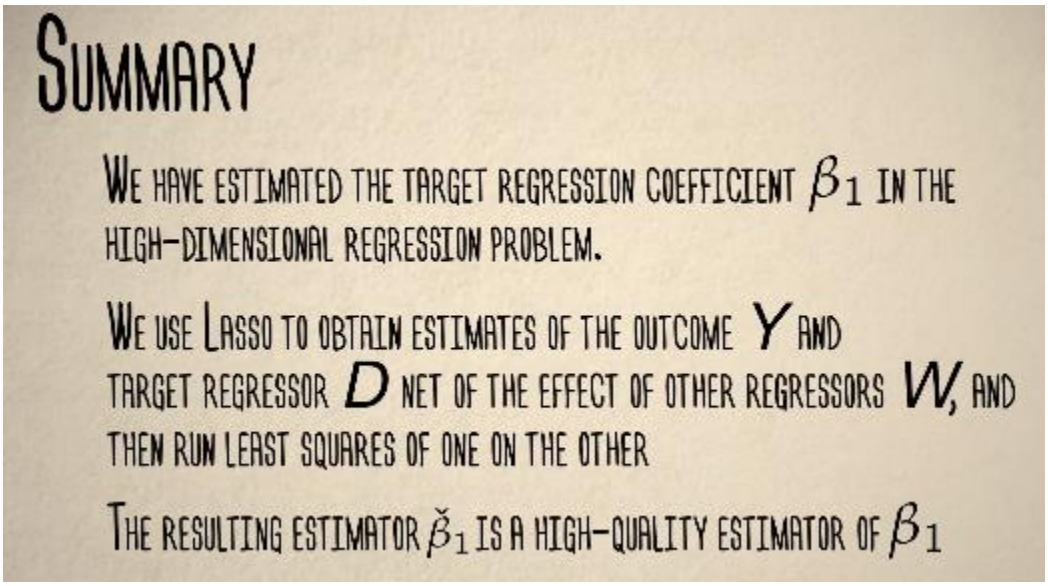
where

$$V = (\mathrm{E}\check{D}^2)^{-1}\mathrm{E}(\check{D}^2\epsilon^2)(\mathrm{E}\check{D}^2)^{-1}.$$

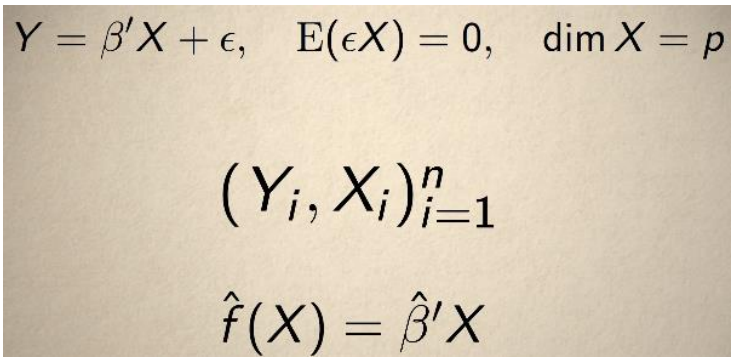
The above statement means that $\check{\beta}_1$ concentrates in a $\sqrt{V/n}$-neighborhood of β_1, with deviations controlled by the normal law.

We can use this result just like we used the analogous result for the low dimensional case in block one. We can define the standard error of check beta one as square root of hat v over n. Where hat v is an estimator of v. This is our quantification of uncertainty about that one. We then provide the approximate 95% confidence interval for beta 1, which is given by the estimate check beta 1 + or- two standard errors.





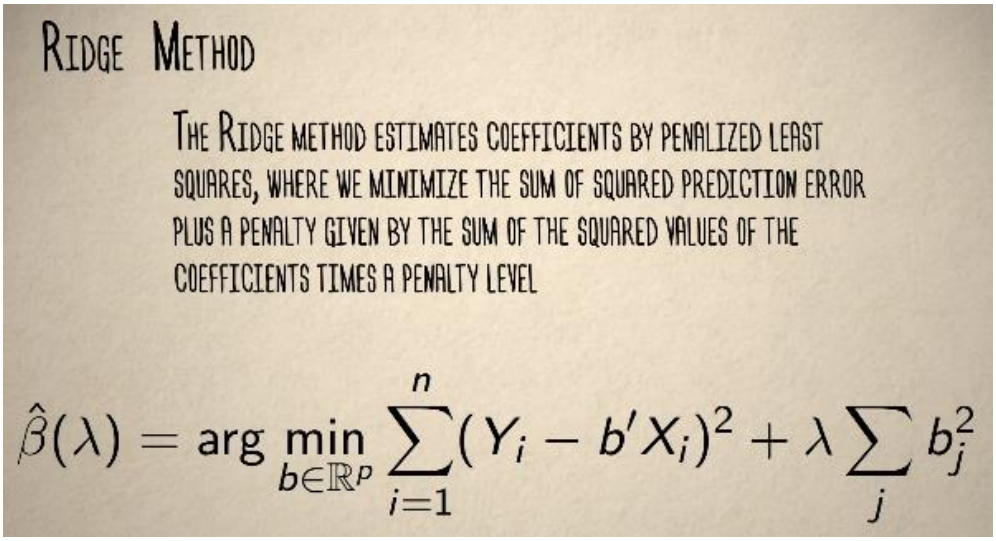
2.2.5 Other Penalized Regression Methods. Cross-Validation



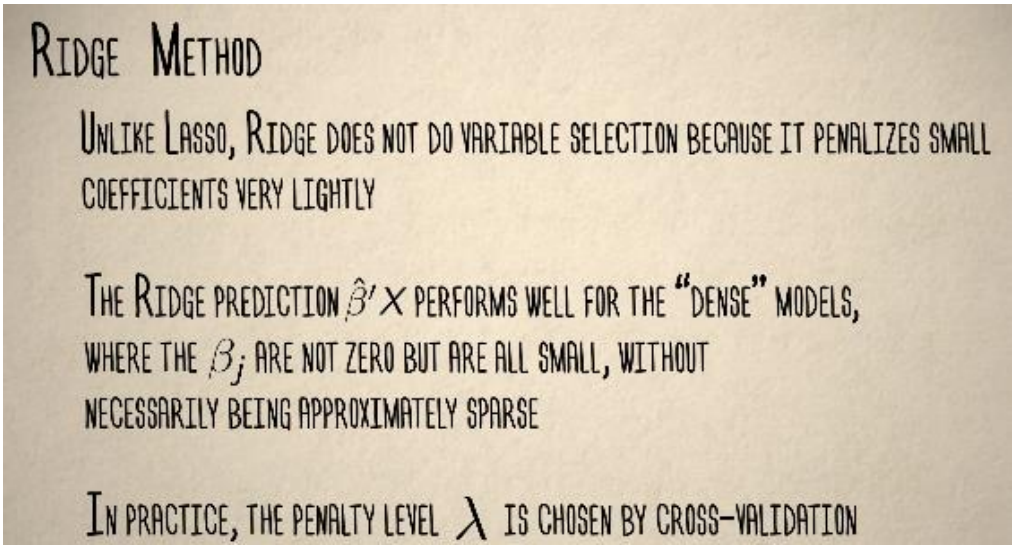
In this segment, we overview other penalized regression methods and also discuss cross-validation, which is a method to choose tuning parameters for the prediction rules. We are interested in prediction in the linear model Y = beta X + epsilon, where epsilon isn't correlated with X. And we have random sample of Yi and Xis. Our generic predictor will take the linear form hat f of x = hat beta x. The idea of penalized regression is to choose the coefficients hat beta, to avoid overfitting in the sample. This is achieved by penalizing the size of the coefficients by various penalty functions.

Ideally, we should choose the level of penalization to minimize the out-of-sample mean squared prediction error. We first consider the Ridge method.

The Ridge method estimates coefficients by penalized least squares, where we minimize the sum of squared prediction error plus the penalty term given by the sum of the squared values of the coefficients times a penalty level, lambda.

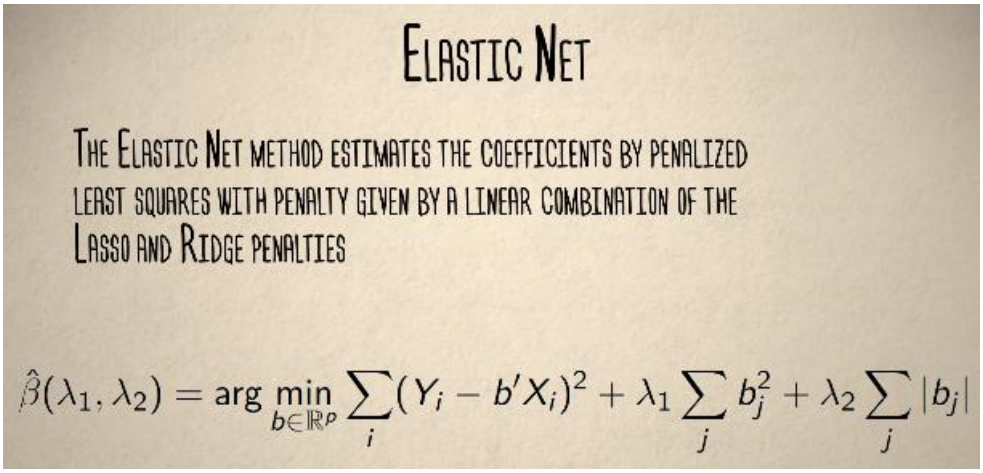


We can see this in the formula given here. If we look at the formula, we notice that analogous to the Lasso, Ridge penalty presses down or penalizes the coefficients without sacrificing too much fit. In contrast to Lasso, Ridge penalizes the large values of coefficients much more aggressively and small values much less aggressively.

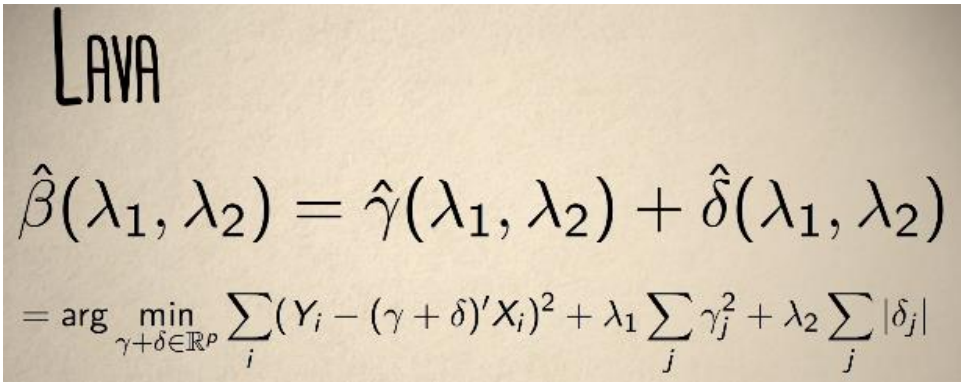


Because Ridge penalizes small coefficients very lightly, the Ridge fit is never sparse. And unlike Lasso, Ridge does not set estimated coefficients to zero, and so it does not do variable selection. Because of this property, Ridge predictor hat beta X is especially well suited for prediction in the dense models, where the beta js are all small without necessarily being approximately sparse. In this case, it can easily outperform the Lasso predictor. Finally, we noted in practice, we can choose the penalty level lambda in Ridge, by cross-validation which we will discuss later in this segment.

One popular modification is the method called the elastic net. Here we estimate the coefficients by penalized least squares with penalty given by the linear combination of the Lasso and the Ridge penalties as you see in this formula. In the formula, we see that the penalty function has two penalty levels, lambda 1 and lambda 2, which could be chosen by cross-validation in practice.

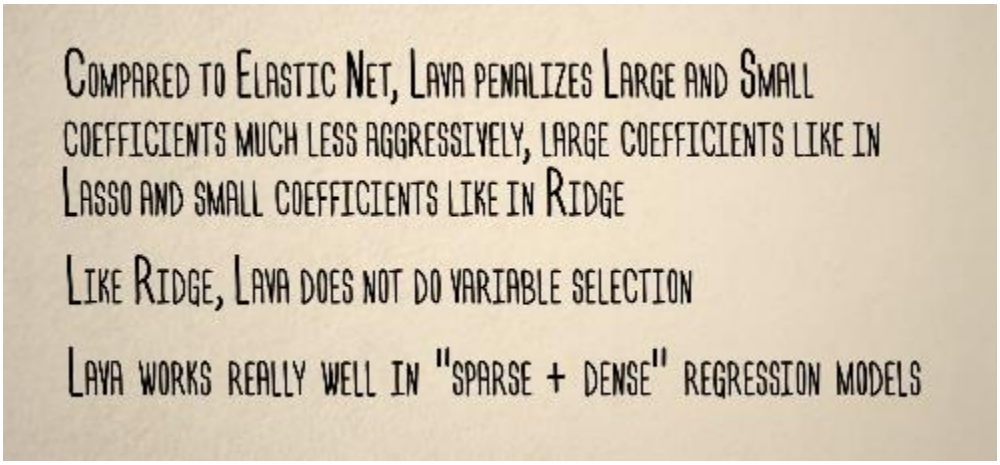


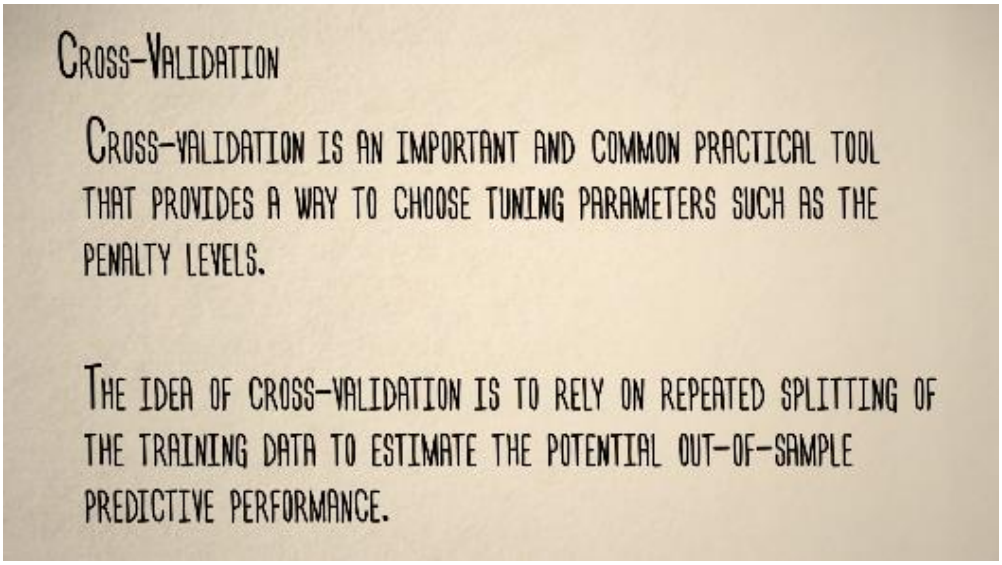
Now, let us look at the formula carefully. We see that the elastic net penalizes large coefficients as aggressively as Ridge. And we also see that it penalizes small coefficients as aggressively as Lasso. By selecting different values of penalty levels, lambda 1 and lambda 2, we could have more flexibility for building a good prediction rule than with just Ridge or with just Lasso. We also note that the elastic net doesn't perform variable selection unless we completely shut down the Lasso penalty by setting penalty level lambda 1 equals zero.



Another useful modification of Lasso and Ridge is the lava method. In the lava method, we estimate the coefficients by the penalized least squares as shown in this form. If we look at the formula carefully, we see that, just like previously, we are minimizing the sum of squared prediction errors from predicting outcome observations Yi with a linear and Xi prediction rule plus a penalty term. However, unlike previously, we are splitting the parameter components into gamma j, + delta j, and penalize gamma j like in Ridge and delta j like in Lasso. There are two corresponding penalty levels, lambda 1 and lambda 2, which can be chosen by cross-validation in practice.

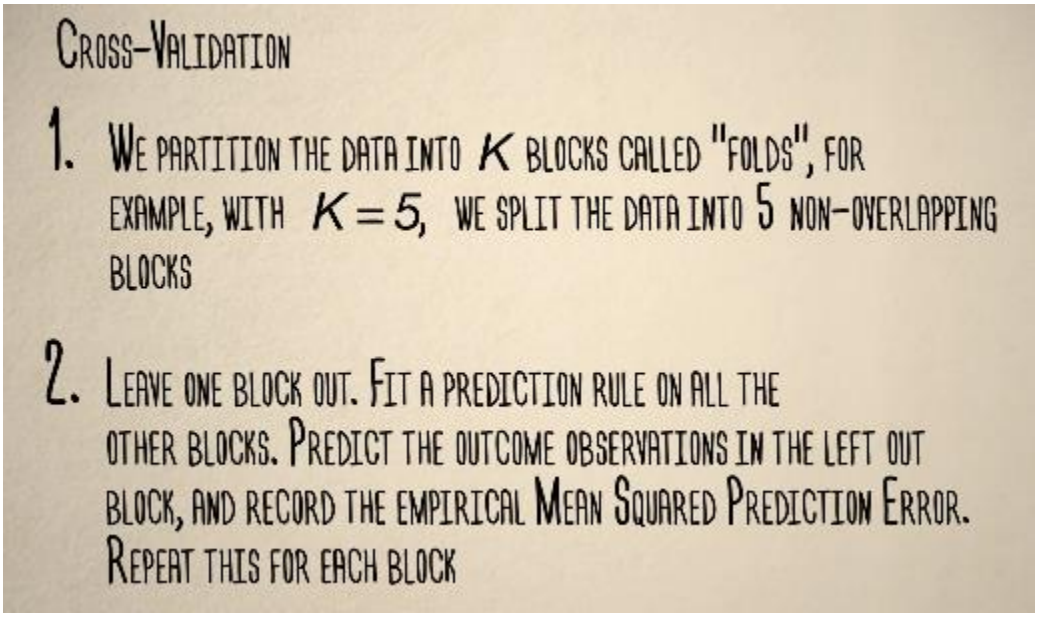
Now, because of the splitting, lava penalizes coefficients least aggressively compared to Ridge, Lasso, or elastic net, because it penalizes large coefficients like in Lasso and small coefficients like in Ridge. Lava never sets estimated coefficients to zero, and so it doesn't do variable selection. The lava method works really well in sparse + dense models, where there are several large coefficients and many small coefficients, which are not necessarily sparse. In these types of models, lava significantly outperforms Lasso, Ridge or elastic net.

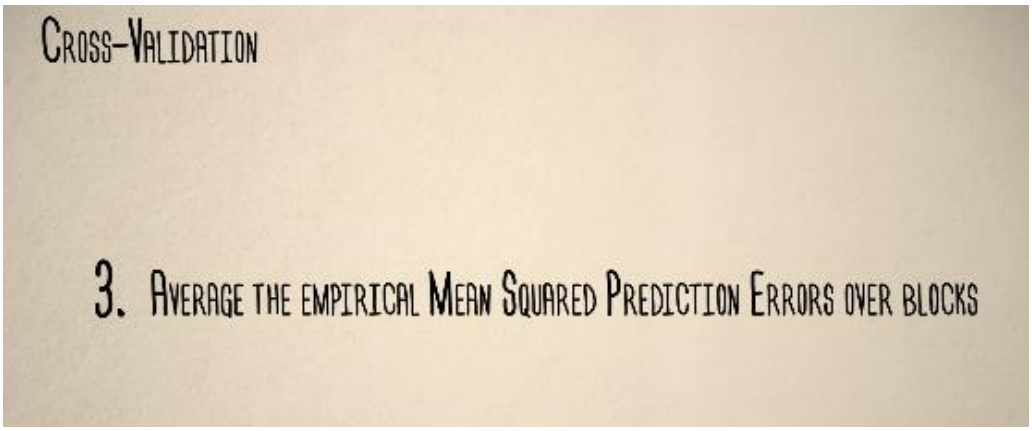




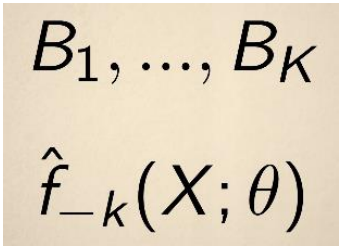
In step 1, we divide the data into K blocks called folds. For example was, K equal to five, we split the data into five parts.
In step 2, we begin by leaving one block out. We fit the prediction rule on all other blocks, we then predict outcome observations in the left out block and record the empirical mean squared prediction error. We repeat this procedure for each block.

In step 3, we average the empirical mean squared prediction errors over blocks. We do these steps for several or many values of the tuning parameters, and we choose the best tuning parameter to minimize the average mean squared prediction error.





Let us now consider more formal description of cross-validation. We randomly select equal sized blocks B1 through Bk to randomly partition the observation indices one through M.



We then fit a prediction rule denoted by hat f sub -k or X and theta. Where theta denotes tuning parameters such as penalty levels, and hat f sub -k depends only on observations that are not in the block Bk.

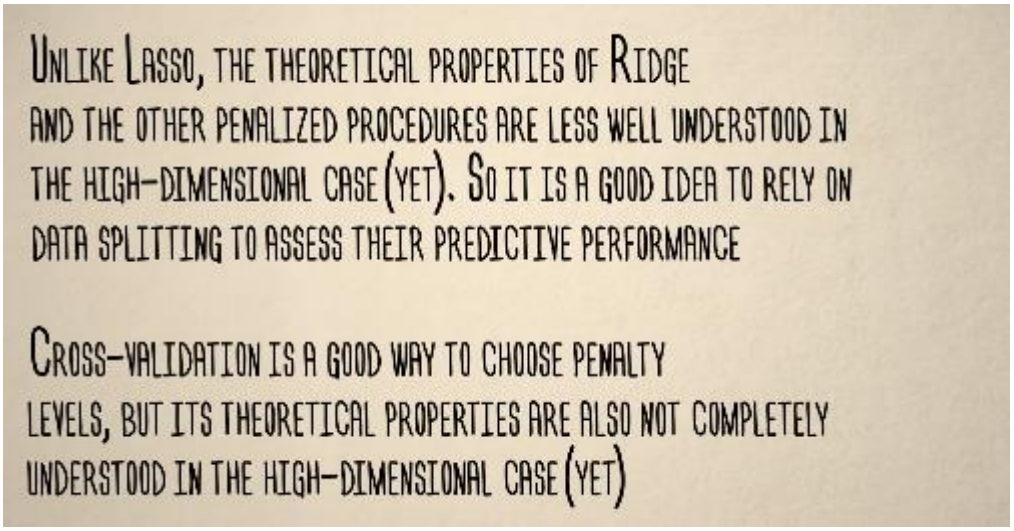
The empirical mean squared error for the block of observations Bk is given by the average squared prediction error for this block, as shown in this formula. In this formula M is the size of the block.

$$\mathrm{MSE}_k(\theta) = \frac{1}{m} \sum_{i \in B_k} (Y_i - \hat{f}_{-k}(X_i; \theta))^2$$

The cross validated MSE is the average of MSEs for each block as shown again, in this formula.

$$\text{CV-MSE}(\theta) = \frac{1}{K} \sum_{k=1}^{K} \mathrm{MSE}_k(\theta)$$

Finally, the best tuning parameter theta is chosen by minimizing the cross validated MSE.



We now provide some concluding remarks.

First, we note that in contrast to Lasso, the theoretical properties of Ridge and other penalized procedures are less well understood in the high-dimensional case, yet. So it is a good idea to rely on test data to assess their predictive performance.

Second, we note that cross validation is a good way to choose penalty levels, but its theoretical properties are not completely understood in high-dimensional case yet. So again, it is a good idea to rely on task data to assess the predictive performance of cross-validated rules. Finally, we may want to ask a question here.

How do the penalize regression methods work in practice?
In the next part of our module we will assess the predictive performance of these methods in a real example, where we will also compare these methods to modern nonlinear regression methods.

2.3.1 Modern Nonlinear Regression. Trees, Random Forests, and Boosted Trees

We begin a new block of our module devoted modern nonlinear regression. Here we are interested in predicting the outcome y using the raw regressor Zed which are k-dimensional.
Recall that the best prediction rule g(Z) is the conditional expectation of y given Z. In this module so far, we have used best linear prediction rules to approximate g(Z) and linear regression or lost regression for estimation. Now we consider nonlinear prediction rules to approximate g(Z), including tree-based methods and neural networks. In this segment we discussed tree-based methods.

Modern Nonlinear Regression

We are interested in predicting the outcome Y using the raw regressors Z, which are K-dimensional

Recall that the best prediction rule $g(Z)$ is the conditional expectation of Y given Z:

$$g(Z) = \mathrm{E}(Y|Z)$$

Now we consider nonlinear prediction rules to approximate $g(Z)$, including tree-based methods and neural networks

Regression Trees

The Idea is to partition the regressor space where Z takes values into a set of regions, and then provide a predicted value for each region

Suppose we have a sample of N observations $(Z_i, Y_i)_{i=1}^n$

Regression Trees

Suppose we have a sample of N observations $(Z_i, Y_i)_{i=1}^n$

Given a partition of the regressor space into M regions $R_1, \ldots, R_M$, which will be determined by data, the regression tree is a prediction rule of the form:

$$\hat{g}(Z) = \sum_{m=1}^{M} \hat{\beta}_m 1(Z \in R_m)$$

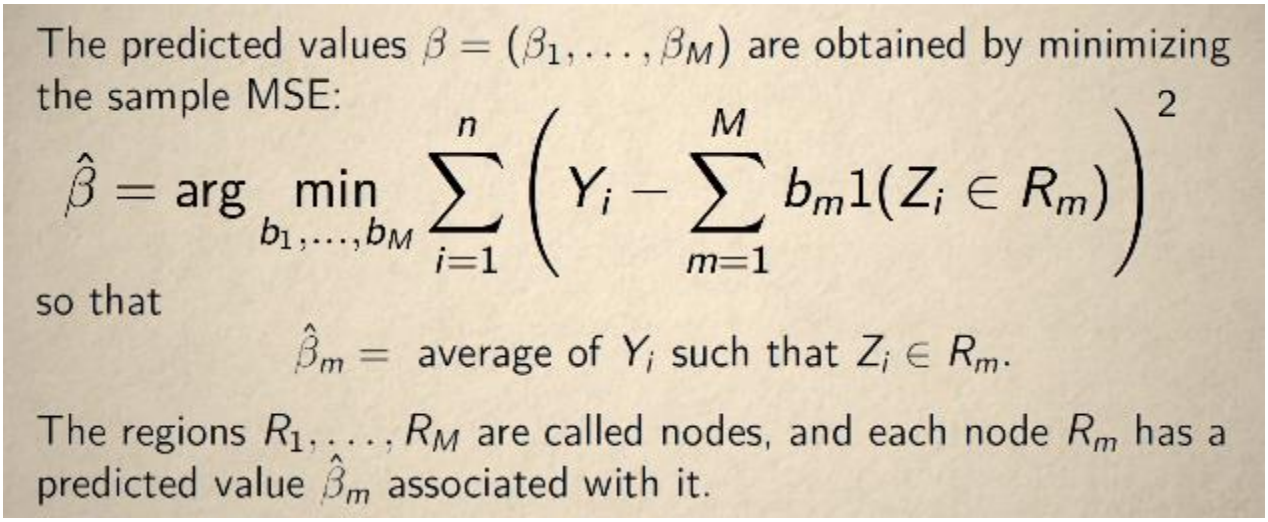
where $1(Z \in R_m)$ is an indicator for the region R_m, and $\hat{\beta}_m$ is the predicted value for the region R_m.

In this segment we discussed tree-based methods. The idea of regression trees is to partition the regressor space where Z takes values into a set of rectangles, and then for each rectangle provide a predicted value. Suppose we have n observations, Zi, Yi for i ranging from 1 to n.

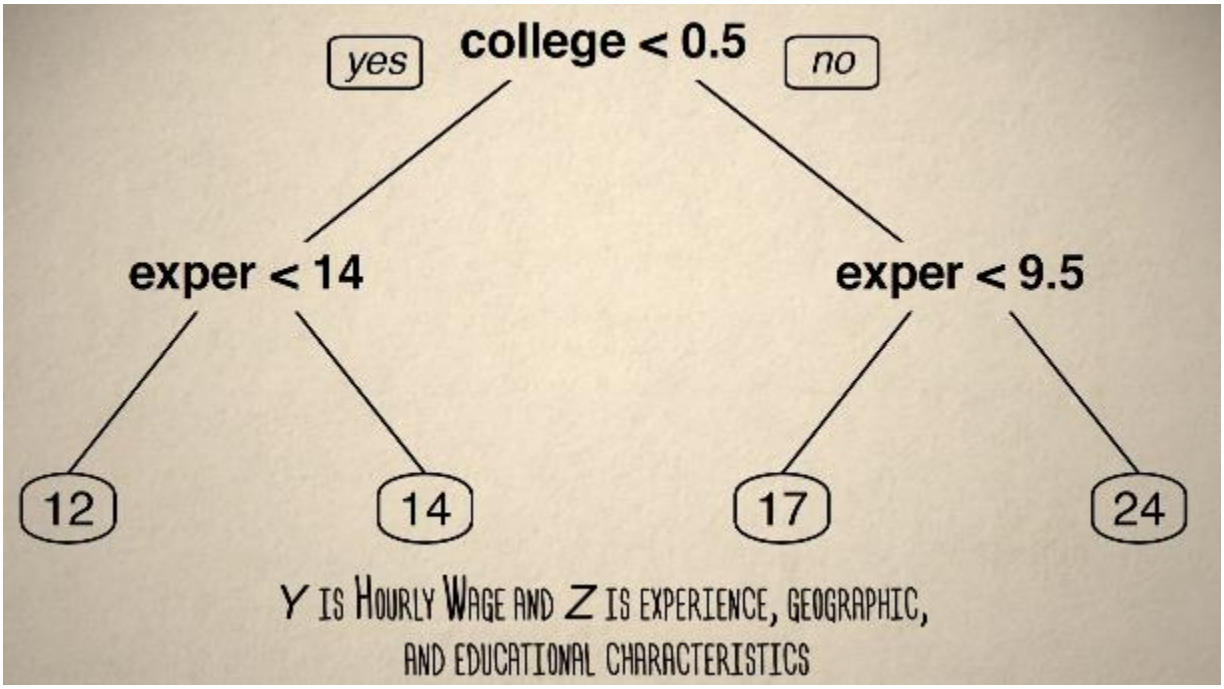
Given the partition of the space into M rectangles, R1 through Rm, which will be determined by data. The regression rule is a prediction of the four. Hat g(Z) is equal to the sum over m from 1 to m of hat beta m times the indicator that Z belongs to the rectangle Rm.

The estimated coefficients are obtained by minimizing the sample MSE, as shown in this formula. From the formula we can conclude that hat Beta m is set equal to average of Yi with Zi falling in the rectangle Rm, as shown in the formula that you see.

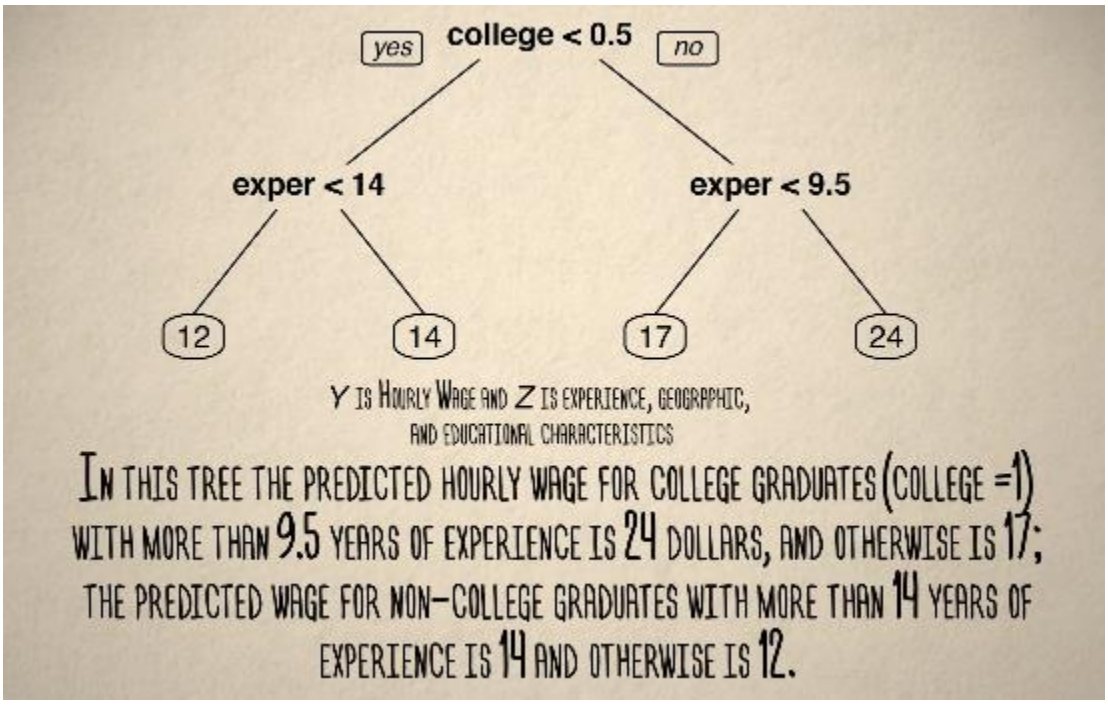
The rectangles or regions Rm are called nodes and each nodes has predicted value had Rm associated with it.



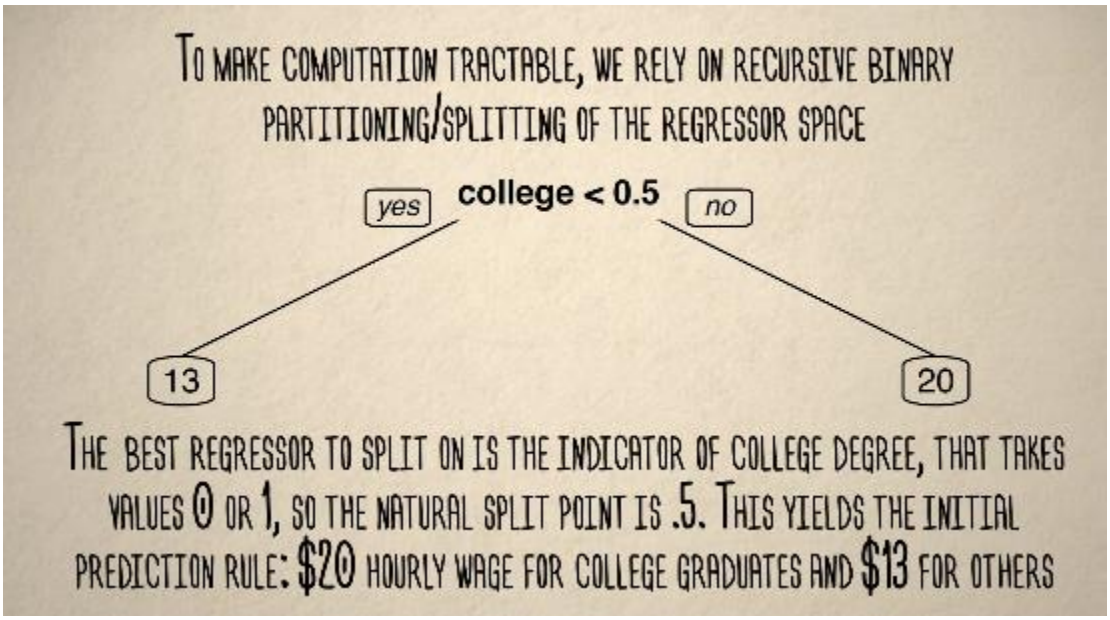
And nice thing about regression trees is that you get to draw cool pictures like this one. Here we show a tree based prediction rule for our wage example where Y is wage and Z are experience, geographic, and educational characteristics.

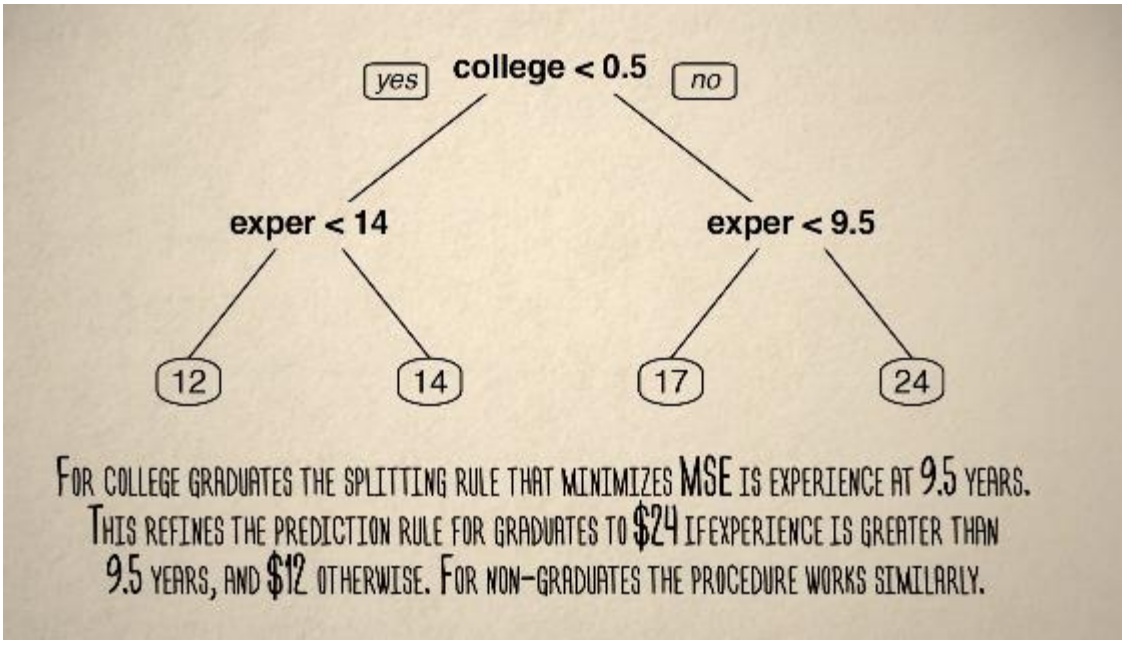


As you can see from looking at the terminal nodes of the tree, in this tree, the predicted hourly wage for college graduates with more than 9.5 years of experience is $24, and otherwise it is 17. The predicted wage for non-college graduates with more than 14 years of experience is 14, and otherwise it is 12. Now how do we grow this tree?

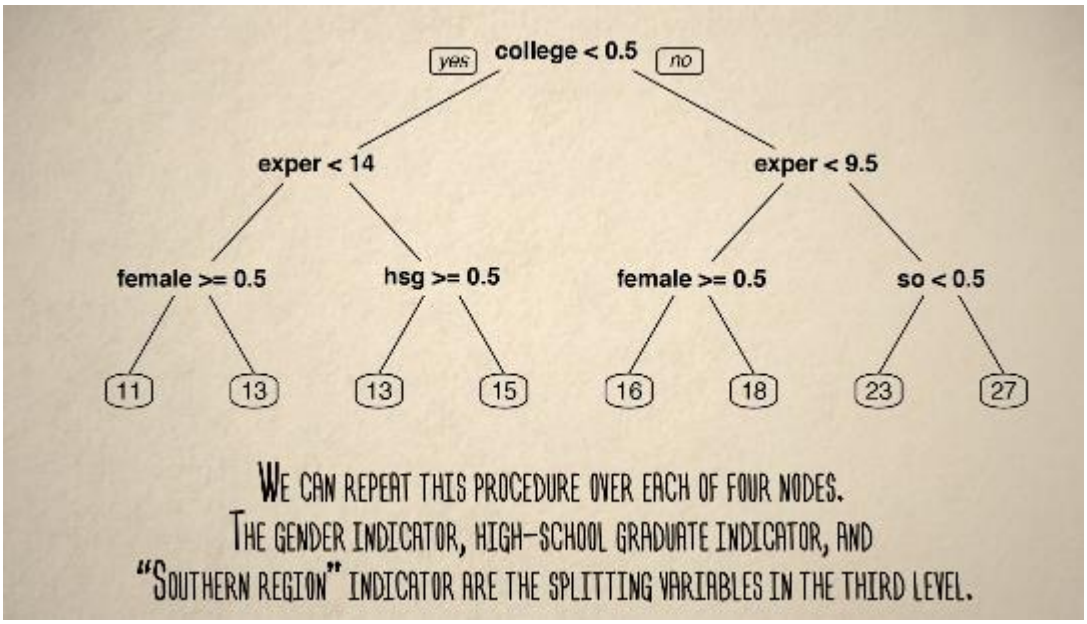


First, we cut that regressor space into two regions by choosing a regressor and a split point that achieve the best improvement in the sample MSE. This gives us the tree of depth one, that you see in the figure. The best variable to split on here is the indicator of college degree and it takes values of 0 or 1, so the natural split point is 0.5. This gives us a starting tree-based prediction rule, which predicts a $20 hourly wage for college graduates and 13 for all others.





Second, we repeat this procedure over two resulting regions or nodes, college graduates and non-college graduates, obtaining in this step four new nodes that we see in this figure. For college graduates the splitting rule that minimizes MSE is the experience regressor at 9.5 years. This refines the prediction rule for graduates to $24 if experience is greater than 9.5 years and $12 otherwise. For non-graduates the procedure works similarly.



Third, we repeat this procedure over each of the four nodes. The tree of that three now has eight nodes. We see that in the final level we are now splitting our gender indicator, high school graduate indicator, and the south indicator. Finally, we stop growing the tree when the desired depths of the tree is reached. Or when the minimal number of observations per node, called minimal node, size is reached.

The deeper we grow the tree, the better is our approximation to the prediction function $g(Z)$.

On the other hand, the deeper the tree, the more noisy our estimate $\hat{g}(Z)$ becomes, since there are fewer observations per terminal node to estimate the predicted values.

From a prediction point of view we can try to find the right depth or structure of the tree by cross-validation.

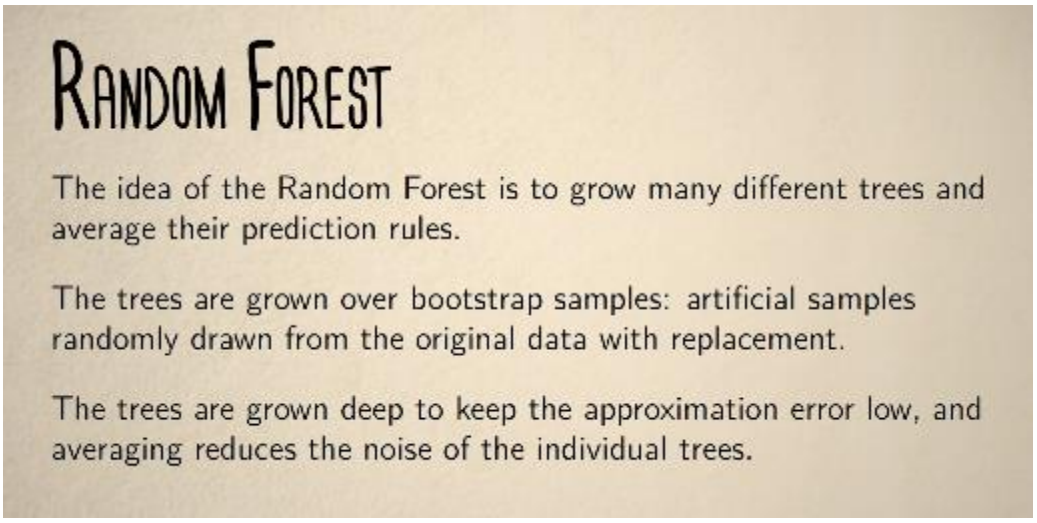
For example, in the wage example the tree of depth 2 performs better in terms of cross-validated MSE than the trees of depth 3 or 1.

Pruning the Tree

The process of cutting down the branches of the tree to improve predictive performance is called Pruning The Tree.
However in practice pruning the tree often doesn't give satisfactory performance because a single prune tree provides a very crude approximation to the regression function g(Z).

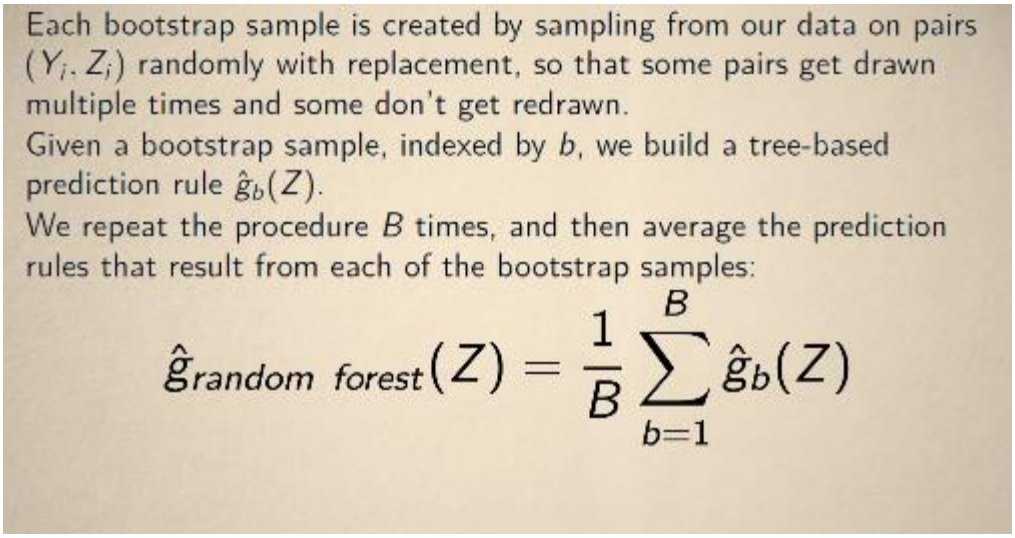
Pruning the tree rarely gives satisfactory performance in practice, because a single pruned tree provides a very crude approximation to the regression function $g(Z)$

A much more powerful and one use approach to improve simple regression trees is to build the Random Forest.



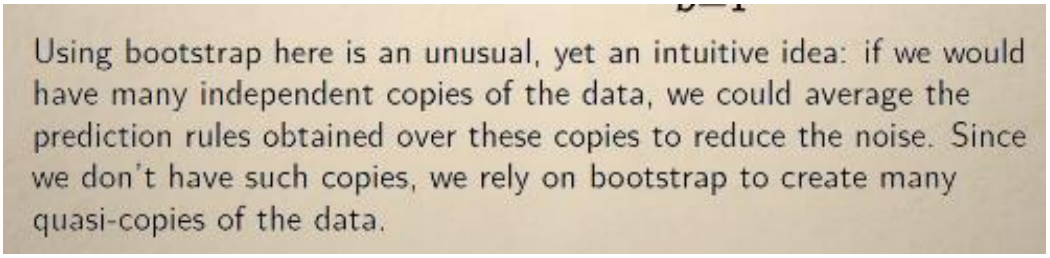
The idea of Random Forest is to grow many different deep trees and then average prediction rules based on them. The trees are grown over different artificial data samples generated by sampling randomly with replacement from the original data. This way of creating artificial samples is called the bootstrap statistics.

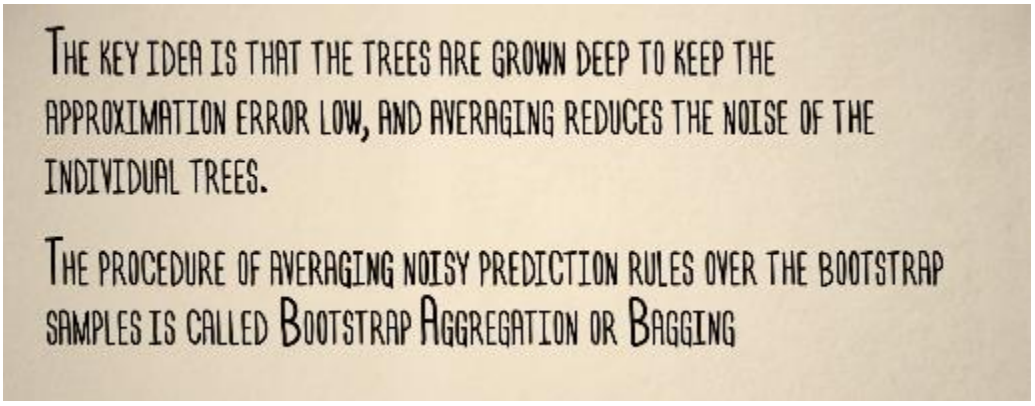
The trees are growing deep to keep the approximation error low and averaging is meant to reduce the noisiness of the individual trees.



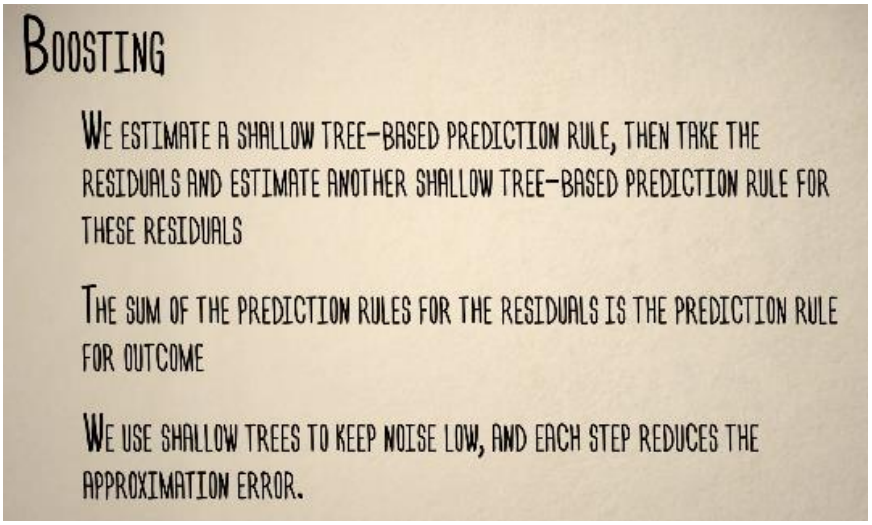
Each bootstrap sample is created by sampling from our data on pairs (Yi, Zi) randomly with replacement, so some observations get drawn multiple times and some don't get redrawn at all. Given a bootstrap sample numbered by numbered by numeral b, we build a tree-based prediction rule hat gb (Z). We repeat the procedure

capital B times in total and then average the prediction rules, that result from each of the bootstrap samples.



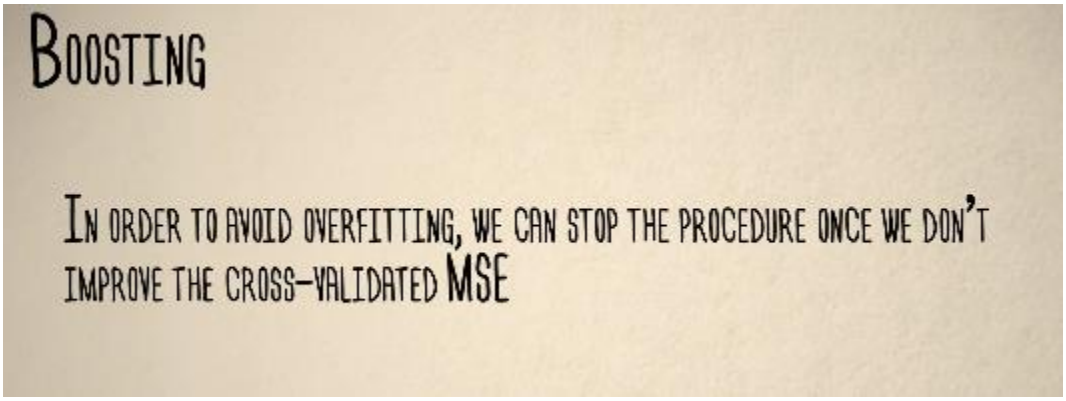


Using bootstrap here is an intuitive idea. If we could have many independent copies of the data, we could average the prediction rules obtained over these copies to reduce the noisiness. Since we don't have such copies, we rely on bootstrap copies of the data. The key underlying idea here is that the trees are grown deep to keep the approximation error low and averaging is meant to reduce the noisiness of the individual trees. The procedure of averaging noisy prediction rules over the bootstrap samples is called Bootstrap Aggregation or Bagging.

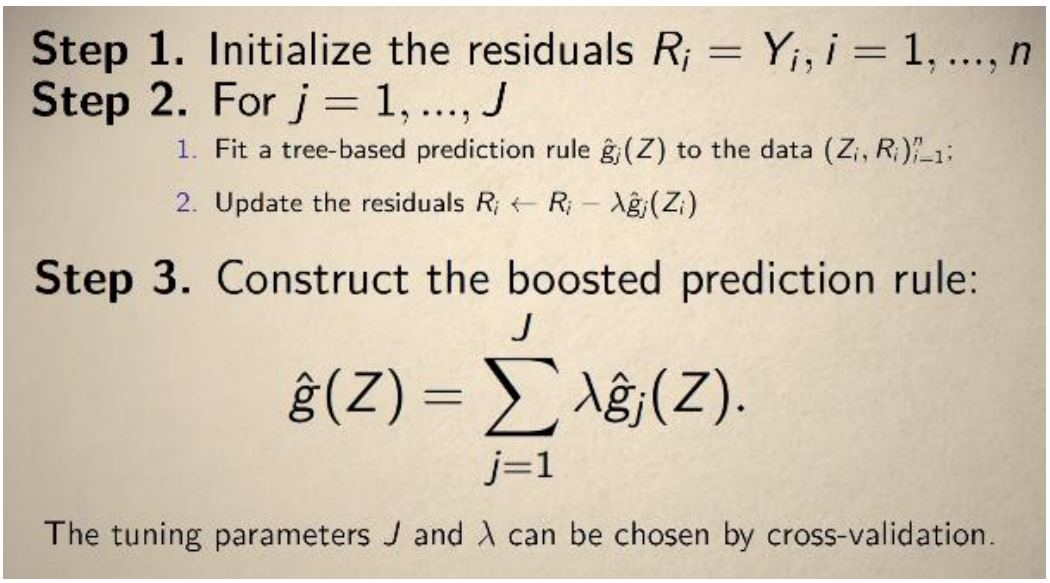


Another approach to improve simple regression tree is by boosting. The idea of boosting is that of request of fatigue where estimated tree-based prediction rule, then we take the residuals and estimate another shallow tree-based prediction rule for these residuals and so on. Summing up the prediction rules for the residuals gives us the prediction rule for the outcome.

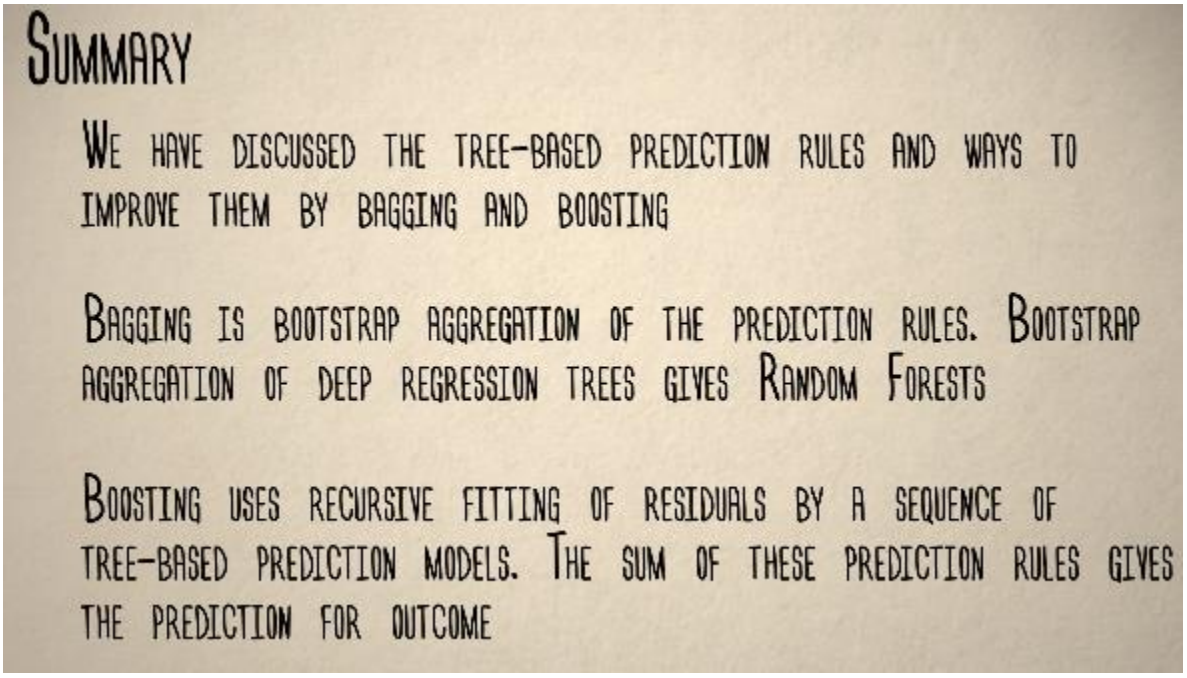
Unlike in a random forest, we use shallow trees, rather than deep trees, to keep the noise low and each step of boosting aims to reduce the approximation error. In order to avoid overfitting and boosting, we can stop the procedure once we don't improve the cross-validated mean square error.



In order to avoid overfitting and boosting, we can stop the procedure, once we don't improve the cross-validated mean square error.



Formerly, the boosting algorithm looks as follows.
In step 1, we initialize the residuals Ri = Yi, for i that runs from 1 to n.
In step 2, we perform the following operation over index j running from 1 to capital J.
We fit a tree-based prediction rule, gj(Z) to the data (Zi,Ri) with i from 1 to n and we have data residuals as R1 equals previous Ri minus lambda times hat g sub j (Zi).
Finally, in step 3, we output the boosted prediction rule. Hat g(Z) equals sum over J of lambda hat g sub j(Z). The capital J and lambda that you see here are the tuning parameters. Representing the number of boosting steps, and the degree of updating the residuals. In particular, lambda equals 1, gives us the full update. We can choose j and lambda by cross validation.



So, let us summarize. We discussed the tree-based prediction rules, and ways to improve them by bagging or boosting. Bagging is bootstrap aggregation of the prediction rules. Bootstrap aggregation of deep regression trees gives us random forests. Boosting uses recursive fitting of residuals by a sequence of tree-based prediction models. The sum of these prediction rules gives us the prediction for outcome.

2.3.2 Modern Nonlinear Regression. Neural Networks

The idea of the neural network is to use parameterized nonlinear transformations of linear combinations of the raw regressors as constructed regressors, called neurons. And produce the predicted value as a linear function of these regressors.

WE ARE INTERESTED IN PREDICTING THE OUTCOME Y USING THE RAW REGRESSORS Z, WHICH ARE K-DIMENSIONAL. RECALL THAT THE BEST PREDICTION RULE IS THE CONDITIONAL EXPECTATION OF Y GIVEN Z

$$g(Z) = \mathrm{E}(Y|Z)$$

IDEA: USE PARAMETERIZED NONLINEAR TRANSFORMATIONS OF LINEAR COMBINATIONS OF THE RAW REGRESSORS AS CONSTRUCTED REGRESSORS (CALLED NEURONS), AND PRODUCE THE PREDICTED VALUE AS A LINEAR FUNCTION OF THESE REGRESSORS

The prediction rule $\hat{g}(Z)$ is nonlinear in some parameters and with respect to the raw regressors. With sufficiently many neurons, $\hat{g}(Z)$ can approximate the best prediction rule $g(Z)$.

In part 2 of our module, we already saw that many constructed regressors are useful in the high-dimensional linear setting to approximate $g(Z)$.

Neural networks also rely on many constructed regressors to approximate $g(Z)$.

The method and the name "neural networks" were loosely inspired by the mode of operation of the human brain, and developed by scientists working on Artificial Intelligence.

They can be represented by cool graphs and diagrams that we will discuss shortly, so please stay tuned.

Here we focus on single layer neural network to discuss the idea.

The estimated prediction rule will take the form:

$$\hat{g}(Z) = \sum_{m=1}^{M} \hat{\beta}_m X_m(\hat{\alpha}_m)$$

where the $X_m(\hat{\alpha}_m)$'s are constructed regressors called neurons

The M neurons are generated by

$$X_m(\alpha_m) = \sigma(\alpha'_m Z), \quad m = 1, \ldots, M,$$

where α_m's are neuron-specific vectors of parameters called weights, and σ is the activation function, for example:

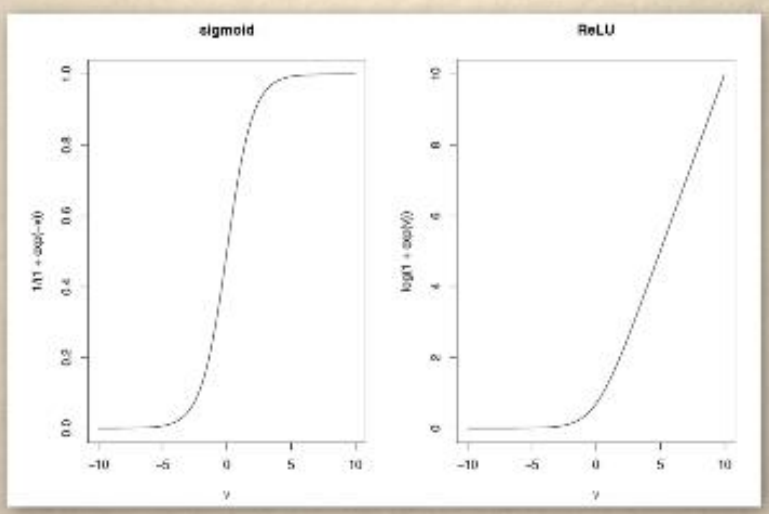
the sigmoid function:

$$\sigma(v) = \frac{1}{1 + e^{-v}}$$

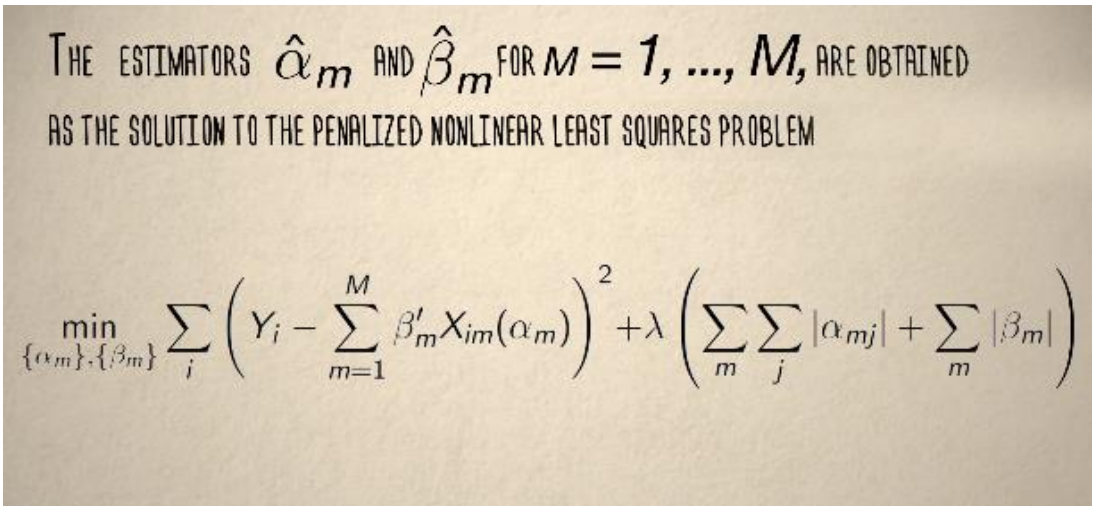
or the rectified linear unit function (ReLU):

$$\sigma(v) = \log(1 + exp(v))$$

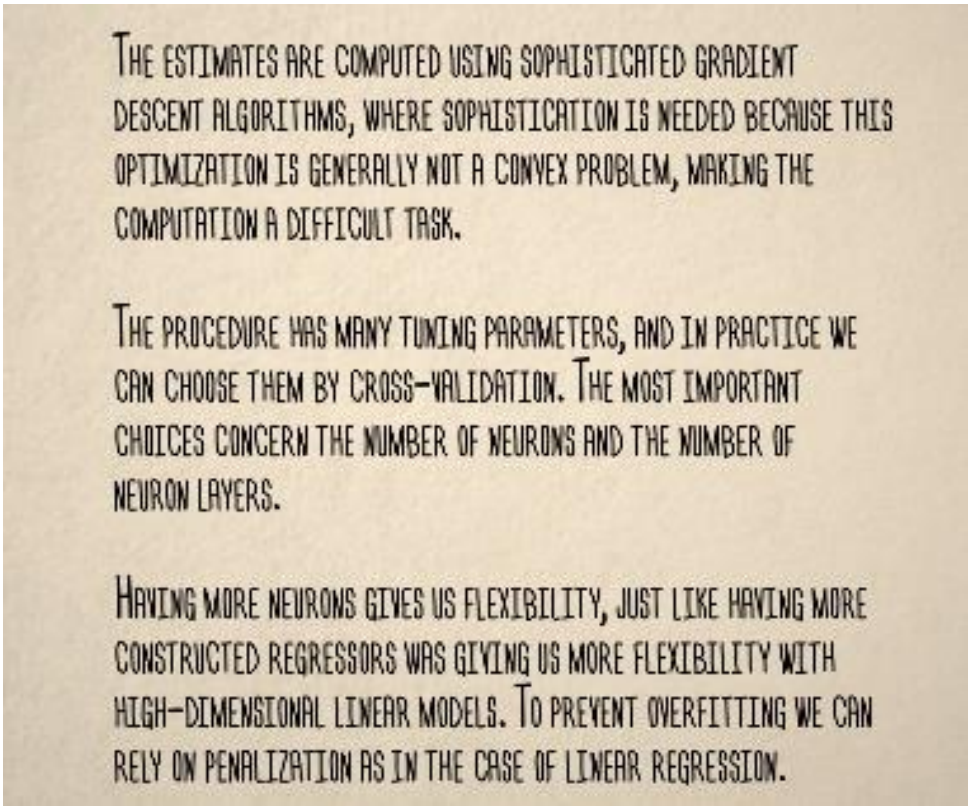
Activation Functions

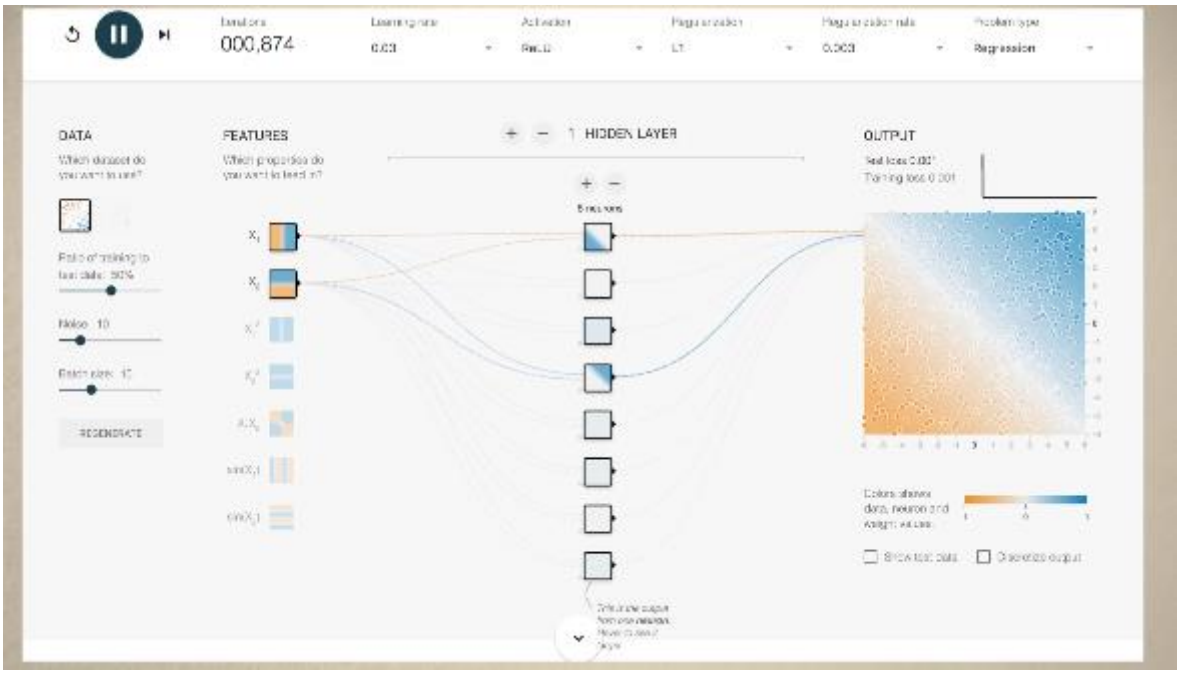


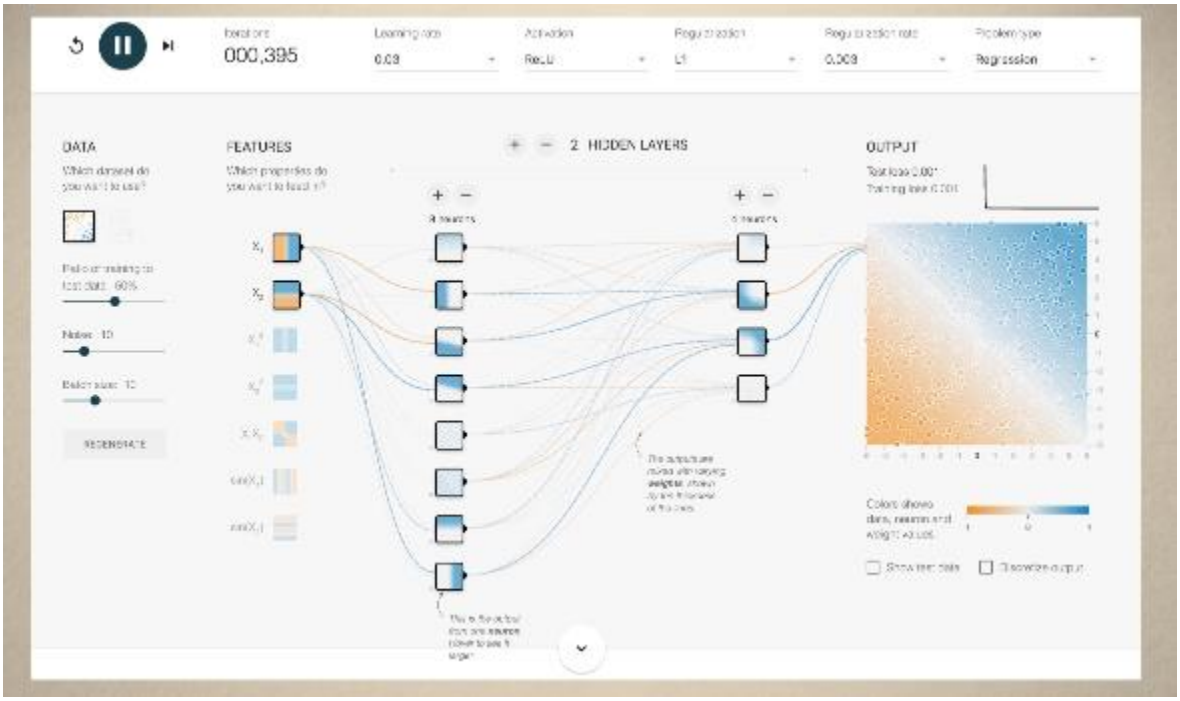
The figure shows the two graphs of the two Activation Functions. The sigmoid function and the rectified linear unit function. The horizontal axis shows the value of the argument. And the vertical axis the value of the function.



The estimators had alpha m, and beta m for each M are obtained as the solution to the penalized nonlinear least squares problem shown by this formula. Here, we are minimizing the sum of squared prediction erros in the sample, plus a penalty term given by the sum of the absolute values of components of alpha m and beta m, multiplied by the penalty level, lambda. In this formula, we use the lasso type penalty but we can also use the ridge, another type of penalties.







In order to visualize working of the neural network, we rely on the resource called Playground.TensorFlow.org using which we produce a prediction model given by simple single layer neural network model. We now see the graphical representation of this network.

Here we have a regression problem and the network depicts the process of taking row regressors and transforming them into predicted values. In the second column on the left, we see the inputs are features. These features are our two row regressors. The third column shows eight neurons. The neurons are constructed as linear combinations of the row regressors transformed by an activation function. That is, the neurons are along linear transformations of the row regressors. Here, we set the activation function, to be the rectified linear unit function, RELU.

The neurons are connected to the inputs and the connections represent the coefficients hat alpha M, which are coefficients of the neuron specific linear transformations of raw regressors. The coloring represents the sign or the coefficients, orange negative and the blue positive.

And with all the connections represents the size of the coefficients. The neurons are then linearly combined to produce the output, the prediction rule. In the diagram we see the connections going outwards from the neurons to the output. These connections represent the coefficients hat beta M, or the linear combination of the neurons that produce the final output. The coloring and the widths represent the sign and the size of these coefficients.

In the diagram, the prediction rule is shown by the heatmap in the box on the right. On the horizontal and vertical axis, we see the values of the two inputs. The color and its intensity in the heatmap represent the predicted value.

We also see on the top that the penalty function is L1, which stands for the lasso type penalty. Another option is to use L2, which stands for the rich type penalty. The penalty level is called here the regularization rate. In this example we are using a single layer neural network.

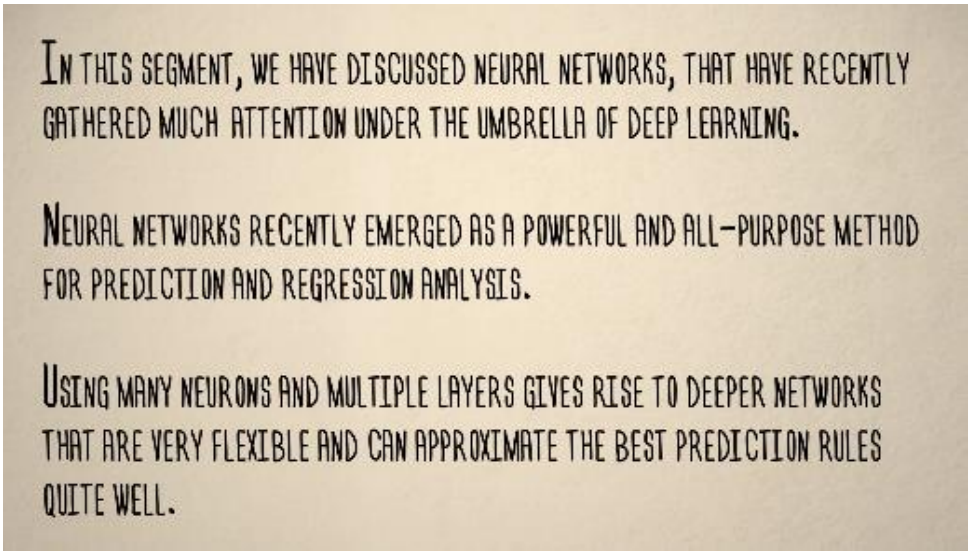
If we add one or two additional layers of neurons constructed from the previous layer of neurons, we get a different network, which we show in the following diagram.

Prediction methods based on neural networks with several layers of neurons called the deep learning methods.

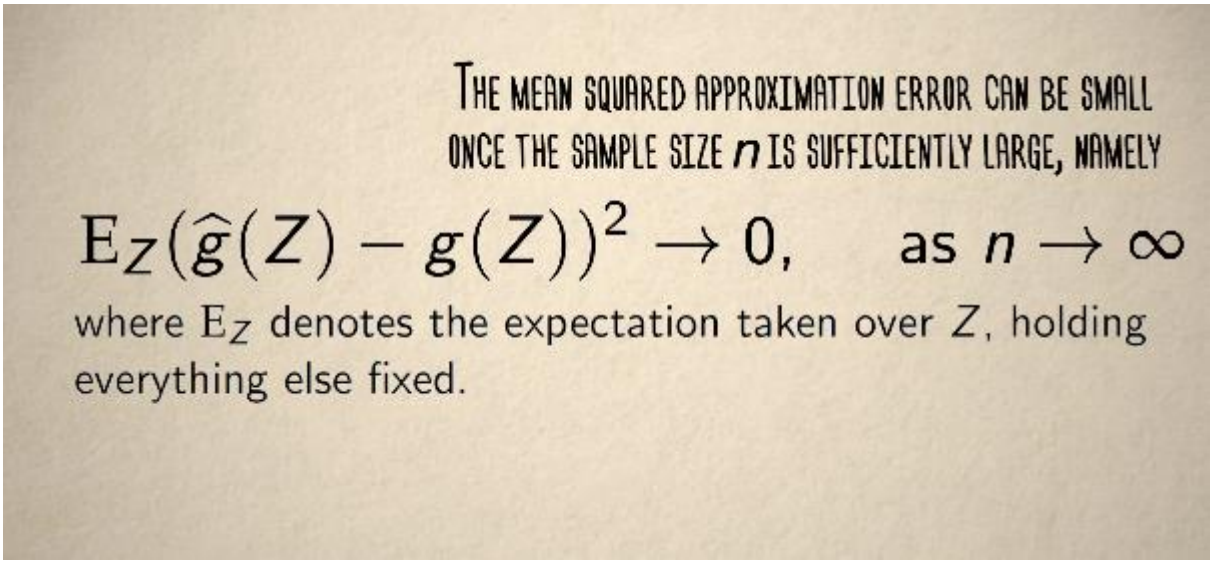
This diagrams showcases that one of the major benefits of doing prediction with neural networks is that you can adapt with pretty cool artwork.

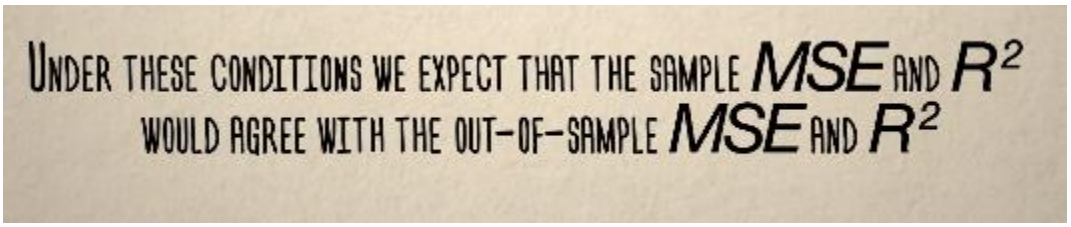
Let us summarize, in this segment we have discussed neural networks that have recently gathered much attention under the umbrella of deep learning. Neural

networks represent a powerful and all-purpose method for prediction and regression analysis. Using many neurons and multiple layers gives rise to networks that are very flexible and can approximate the best prediction rules quiet well.



Aggregation of Predictors





Data Splitting

1. We use a random part of data for estimating/training the prediction rule,

2. We use the other part to evaluate the quality of the prediction rule, recording out-of-sample mean squared error (can also look at R^2).

n observations for training

m for testing/validation

V denote the indices of the observations in the test sample