Stats 102B - Week 3, Lecture 1

Miles Chen, PhD

Department of Statistics

Week 3 Monday

UCLA

Section 1

More Regularization and The Bias-Variance Tradeoff

Quick review of regularization:

- The Regularized Loss function combines the OLS Loss function with _____.
- The _____ term is used to determine which goal has greater priority.

Quick review of regularization

- The Regularized loss function combines the OLS Loss function with **a penalty for complexity**.
- The λ term is used to determine which goal has greater priority.

Review of LASSO and Ridge Regression

In the last lecture, we introduced LASSO and Ridge Regression. Both modify the loss function by adding a penalty for complexity.

$$\mathcal{L}' = \mathcal{L} + \lambda[\text{complexity}]$$

where $\mathcal{L}$ is the MSE.

The LASSO uses the ℓ_1 norm (sum of absolute values) of the vector of coefficients $\mathbf{w}$ as the measure of complexity.

$$\mathcal{L}' = \mathcal{L} + \lambda||\mathbf{w}||_1$$

The Ridge Regression uses the ℓ_2 norm squared (equal to sum of squared values) of the vector of coefficients $\mathbf{w}$ as the measure of complexity.

$$\mathcal{L}' = \mathcal{L} + \lambda||\mathbf{w}||_2^2$$

Elastic Net

Elastic Net is a hybrid of LASSO and Ridge regression. It uses both ℓ_1 and ℓ_2 norms as a measure of complexity.

$$\mathcal{L}' = \mathcal{L} + \lambda \left[\alpha ||\mathbf{w}||_1 + (1 - \alpha) ||\mathbf{w}||_2^2 / 2 \right]$$

The parameter λ determines how much penalty is applied.

The parameter α determines the balance between using the ℓ_1 and ℓ_2 norms. When $\alpha = 0$, it is equal to ridge regression. When $\alpha = 1$, it is equal to LASSO.

A full description https://cran.r-project.org/web/packages/glmnet/vignettes/glmnet_beta.pdf

LASSO vs Ridge Regression

From glmnet vignette:

"It is known that the ridge penalty shrinks the coefficients of correlated predictors towards each other while the LASSO tends to pick one of them and discard the others. The elastic-net penalty mixes these two; if predictors are correlated in groups, an $\alpha = 0.5$ tends to select the groups in or out together. This is a higher level parameter, and users might pick a value upfront, else experiment with a few different values."

Because LASSO tends to push coeffecients down to zero, it can be used for variable selection.

Library glmnet

The elastic net has been implemented in the package `glmnet`.

```
library(glmnet)
```

To use `glmnet()`, specify a matrix X of predictors and a matrix y of outcomes. The default settings of glmnet is `alpha = 1`, which will perform LASSO. (As lambda increases, LASSO will push some coefficients to zero.)

```
fit = glmnet(X, y, alpha = 1)  # The default is alpha = 1 (LASSO)
```

Generating some artificial data

I generate some artificial data. The artificial data consists of 4 X variables. X1 and X2 are correlated (r = 0.8). X3 and X4 are correlated (r = 0.8). X1 is independent of X3 and X4. X2 is also independent of X3 and X4.

The values of Y in the population depend only on X1 and X3 (`true_y`).

I add random noise to `true_y` to get our observed y values.

```
library(mvtnorm)
set.seed(8)
n <- 50
sigma <- matrix(c(1, 0.85, 0.85, 1), nrow = 2) # covariance matrix
x1x2 <- rmvnorm(n, sigma = sigma) # rmvnorm generates an n x 2 matrix
x3x4 <- rmvnorm(n, sigma = sigma)
X <- cbind(x1x2, x3x4) # X is n x 4 matrix
true_y <- 3 * X[,1] - 1.8 * X[,3] # creation of true_y
error <- rnorm(n, sd = 1)
y <- true_y + error
```

LASSO using glmnet on artificial data

We use glmnet to fit a LASSO model to this artificial data

```
fit = glmnet(X, y, alpha = 1)  # LASSO is alpha = 1
```

The results of the LASSO fit using glmnet

```
fit
```

```
##
## Call:  glmnet(x = X, y = y, alpha = 1)
##
##    Df  %Dev  Lambda
## 1   0  0.00 3.02200
## 2   1 10.99 2.75300
## 3   1 20.12 2.50900
## 4   1 27.70 2.28600
## 5   2 35.19 2.08300
## 6   2 45.41 1.89800
## 7   2 53.90 1.72900
## 8   2 60.94 1.57500
## 9   2 66.79 1.43600
## 10  2 71.65 1.30800
## 11  2 75.68 1.19200
## 12  2 79.03 1.08600
## 13  2 81.81 0.98940
## 14  2 84.12 0.90150
## 15  2 86.03 0.82150
## 16  2 87.62 0.74850
## 17  2 88.94 0.68200
## 18  2 90.04 0.62140
## 19  2 90.95 0.56620
## 20  2 91.70 0.51590
## 21  2 92.33 0.47010
## 22  2 92.85 0.42830
## 23  2 93.28 0.39030
## 24  2 93.64 0.35560
## 25  2 93.94 0.32400
## 26  2 94.19 0.29520
## 27  2 94.39 0.26900
```

glmnet LASSO results

The results are shown as a listing with three columns.

The first column is the additional Degrees of Freedom spent in this model vs the empty model. Each additional parameter you estimate uses an additional degree of freedom. Effectively, this is the number of variables used in the model.

The middle column is the percent of deviation explained. It is equivalent to the model's R-squared value, or $\frac{SS_{reg}}{SS_{total}}$.

The last column is the value of the lambda term used by glmnet.

When lambda is large, model complexity is punished greatly. This "pushes" all of the coefficients to 0 and we have DF = 0, meaning that the model has fit only a constant (the mean of Y) and uses no X predictors.

As lambda decreases, models are punished less for complexity. This allows the model to incorporate more terms, and the percentage of variation explained increases. If lambda reaches 0, no regularization is applied and the model coefficients would be equivalent to the OLS model coefficients.

OLS model just for comparison

The R-squared value for OLS is 0.5836. Also note the coefficients for the OLS model.

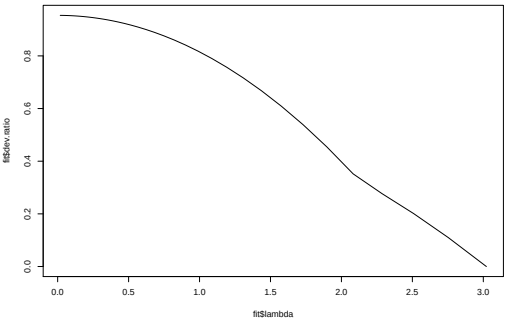
```
ols_model <- lm(y ~ X)
summary(ols_model)
```

```
##
## Call:
## lm(formula = y ~ X)
##
## Residuals:
##      Min       1Q   Median       3Q      Max
## -1.55332 -0.60928  0.01683  0.70611  1.50195
##
## Coefficients:
##             Estimate Std. Error t value Pr(>|t|)
## (Intercept) -0.01935    0.12193  -0.159    0.875
## X1           3.04059    0.18963  16.034  < 2e-16 ***
## X2          -0.01087    0.20417  -0.053    0.958
## X3          -1.84808    0.23204  -7.965 3.88e-10 ***
## X4           0.05432    0.23743   0.229    0.820
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 0.8487 on 45 degrees of freedom
## Multiple R-squared:  0.954,  Adjusted R-squared:  0.9499
## F-statistic: 233.4 on 4 and 45 DF,  p-value: < 2.2e-16
```

glmnet - R-squared vs choice of lambda

When lambda is 0, the coefficient estimates are the same as OLS estimates and the R-squared is the same as the OLS estimates. As lambda increases, the R-squared value decreases until SS_{reg} becomes 0.

```
plot(fit$lambda, fit$dev.ratio, type = "l")
```



glmnet model coefficients

The model coefficients are available in a sparse matrix called `$beta`. A sparse matrix means that if the value is 0, it doesn't bother storing the value. You can get the zeros back with `as.matrix()`

The left-most column is equivalent to having a maxed out lambda where all the coefficients are 0. Read the columns left to right. As you move to the next column, the lambda value decreases and the size of the coefficients increase.

```
options(width = 130)
printSpMatrix (fit$beta[ , 1:10], col.names = TRUE)
```

```
##    s0        s1        s2        s3          s4         s5         s6         s7         s8         s9
## V1  . 0.2736845 0.5230556 0.7502733  0.95638827  1.1409905  1.3091932  1.4624533  1.6020981  1.7293373
## V2  . .         .         .          .           .          .          .          .          .
## V3  . .         .         .         -0.03563035 -0.1924745 -0.3353851 -0.4655999 -0.5842468 -0.6923534
## V4  . .         .         .          .           .          .          .          .          .
```

```
as.matrix(fit$beta[ , 1:10])
```

```
##    s0        s1        s2        s3          s4         s5         s6         s7         s8         s9
## V1  0 0.2736845 0.5230556 0.7502733  0.95638827  1.1409905  1.3091932  1.4624533  1.6020981  1.7293373
## V2  0 0.0000000 0.0000000 0.0000000  0.00000000  0.0000000  0.0000000  0.0000000  0.0000000  0.0000000
## V3  0 0.0000000 0.0000000 0.0000000 -0.03563035 -0.1924745 -0.3353851 -0.4655999 -0.5842468 -0.6923534
## V4  0 0.0000000 0.0000000 0.0000000  0.00000000  0.0000000  0.0000000  0.0000000  0.0000000  0.0000000
```

glmnet model coefficients - tail end of matrix

The right-most column show the coefficients if the lambda was very small. This puts only a tiny penalty on model complexity, so the coefficient estimates are almost identical to the OLS coefficient estimates.

```
max_col <- dim(fit$beta)[2]
as.matrix(fit$beta[ , (max_col - 9):max_col])
```

```
##          s46       s47       s48       s49       s50       s51       s52       s53       s54       s55
## V1  2.99262  2.996329  2.999708  3.002788  3.005593  3.008150  3.010479  3.012602  3.014536  3.016298
## V2  0.00000  0.000000  0.000000  0.000000  0.000000  0.000000  0.000000  0.000000  0.000000  0.000000
## V3 -1.76568 -1.768831 -1.771703 -1.774319 -1.776703 -1.778875 -1.780854 -1.782657 -1.784301 -1.785798
## V4  0.00000  0.000000  0.000000  0.000000  0.000000  0.000000  0.000000  0.000000  0.000000  0.000000
```

```
ols_model$coefficients
```

```
## (Intercept)          X1          X2          X3          X4
## -0.01935256  3.04058976 -0.01086653 -1.84807521  0.05431909
```

glmnet resulting coefficents for a particular lambda

If you want to see the coefficients for a particular value of lambda, you can use the `coef()` function. Specify the value of lambda with the argument `s`.

```
coef(fit, s = 0.2)
```

```
## 5 x 1 sparse Matrix of class "dgCMatrix"
##                      1
## (Intercept) -0.03566925
## V1           2.83482236
## V2           .
## V3          -1.63161009
## V4           .
```

```
coef(fit, s = 1)
```

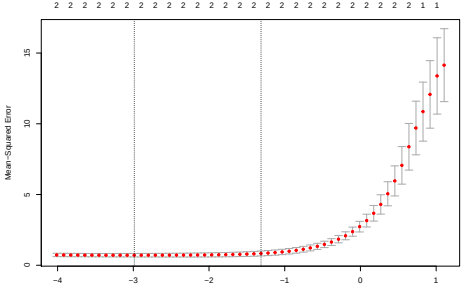
```
## 5 x 1 sparse Matrix of class "dgCMatrix"
##                      1
## (Intercept) -0.1072979
## V1           2.0366273
## V2           .
## V3          -0.9534372
## V4           .
```

Again, we see that a larger lambda means more penalty for larger coefficients, so the model with with lambda $= 1$ has smaller coefficients than the model fit with lambda $= 0.2$.

glmnet Cross Validation

You can perform cross-validation with glmnet. glmnet will automatically create train/test splits of the data and attempt to fit models using various values of lambda. You can create a plot of the results. The results show that when lambda is large (little to no predictors), the mean squared error is large and this model does not fit the data well. On the other hand, picking a very small lambda actually increases the MSE a little bit (it's hard to see on this graph).

```
cvfit = cv.glmnet(X, y, nfolds = 5)
plot(cvfit)
```



glmnet Cross Validation results

You can ask directly for the value of lambda that minimized the MSE in cross-validation. You can also get the coefficients of this particular model.

```
cvfit$lambda.min
```

```
## [1] 0.05040357
```

```
coef(cvfit, s = "lambda.min")
```

```
## 5 x 1 sparse Matrix of class "dgCMatrix"
##                       1
## (Intercept) -0.02227502
## V1           2.98408127
## V2           .
## V3          -1.75842540
## V4           .
```

glmnet Cross Validation results

You can also ask for the largest value of lambda that is within 1 standard error of the minimum lambda.

```
cvfit$lambda.1se
```

```
## [1] 0.2689887
```

```
coef(cvfit, s = "lambda.1se")
```

```
## 5 x 1 sparse Matrix of class "dgCMatrix"
##                       1
## (Intercept) -0.04184621
## V1           2.76598931
## V2           .
## V3          -1.57312726
## V4           .
```

lambda.min and lambda.1se explained

The lambda.min is the value of lambda that produced the smallest cross-validation MSE. This cross-validation MSE, however, has a standard error because it is subject to the random train/test splits. Therefore, the smallest cross-validation error may not be significantly different from another cross-validation error that is slightly larger.

The lambda.1se is a larger value of lambda that still produced a fairly small cross-validation MSE. It didn't produce the smallest cross-validation MSE, but it produced a cross-validation MSE that was still within 1 standard error of the minimum cross-validation MSE.

Section 2

Benefits of regularization

Sampling Distributions of Model Coefficients

When you fit a model to observed data, the estimated coefficients are subject to the random noise in the observed data.

You can imagine taking another random sample from the population and getting similar but different coefficient estimates.

OLS models produce the best linear unbiased estimators (BLUE). This means that the coefficient estimates of OLS models are unbiased. In the long run, after fitting many random sets of data, the OLS coefficients will have an expected value equal to the true coefficient that exists in the population.

"Best" means that these estimates have the lowest amount of variance among all other unbiased estimates.

There is a known issue with OLS. If certain variables are highly correlated, the coefficients of these variables exhibit high variance.

Benefits of regularization

Regularization methods like LASSO and Ridge regression reduce the variance in the fitted models at the expense of introducing bias.

This means that in the long run, after fitting many random sets of data, the LASSO or Ridge Regression coefficients will have an expected value that is **not equal** to the true coefficient that exists in the population.

The benefit is that the variance in these estimates is reduced.

Lower Variance is a good thing

Having less variance in the coefficients is a desirable trait of models.

It means that the model estimates are less likely to be swayed by the noise that exists in the observed data.

If the coefficients of a model change drastically with each random sample, then you will not be very confident in your current set of coefficient estimates.

On the other hand, if the coefficients of a model change very little with different random samples, then you can have more confidence in your current set of coefficient estimates.

Simulation study

In the next few slides, I'll perform a simulation study showing how LASSO and Ridge regression produce coefficient estimates that have lower variance but introduce bias.

Generating the population data

I generate a population. The true relationship between y and X is: X1 has a coefficient of 3 and X3 has a coefficient of -1.8.

```
set.seed(8)
n <- 50
sigma <- matrix(c(1, 0.85, 0.85, 1), nrow = 2) # covariance matrix
x1x2 <- rmvnorm(n, sigma = sigma) # rmvnorm generates an n x 2 matrix
x3x4 <- rmvnorm(n, sigma = sigma)
X <- cbind(x1x2, x3x4) # X is n x 4 matrix
true_y <- 3 * X[,1] - 1.8 * X[,3] # creation of true_y
```

Generating many random samples and fitting OLS, LASSO, and Ridge models

Next, I simulated random samples drawn from the population with a for loop to generate a sampling distribution.

For each iteration of the loop, I generate random noise. The random noise is added to the true values of y which we generated on the previous slide. This simulates a random sample drawn from the population.

Then we fit three models: OLS, LASSO, and Ridge regression.

For LASSO and Ridge regression, I use glmnet's cross-validation feature to select the model with the largest lambda that is within 1 standard error of the lambda that produced the minimum cross-validation score.

We store the coefficients of the resulting models in a few matrices.

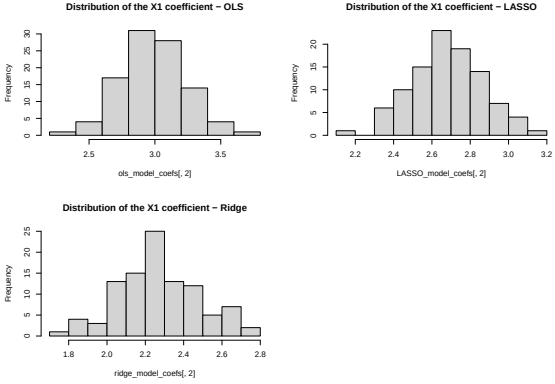
Generating many random samples and fitting OLS, LASSO, and Ridge models

```
reps = 100
ols_model_coefs <- matrix(NA, nrow = reps, ncol = 5)
LASSO_model_coefs <- matrix(NA, nrow = reps, ncol = 5)
ridge_model_coefs <- matrix(NA, nrow = reps, ncol = 5)

for(i in seq_len(reps)){
  set.seed(i)
  error <- rnorm(n, sd = 1)
  y <- true_y + error

  ols_model <- lm(y ~ X)
  ols_model_coefs[i, ] <- ols_model$coefficients

  cvfit_lasso = cv.glmnet(X, y, nfolds = 5)
  LASSO_model_coefs[i, ] <- as.matrix(t(coef(cvfit_lasso, s = "lambda.1se")))

  cvfit_ridge = cv.glmnet(X, y, nfolds = 5, alpha = 0)
  ridge_model_coefs[i, ] <- as.matrix(t(coef(cvfit_ridge, s = "lambda.1se")))
}
```

Sampling distribution comparison

We can now compare the sampling distribution of some of these coefficients. The true X1 coefficient is 3.

```
par(mfrow = c(2,2))
hist(ols_model_coefs[,2], main = "Distribution of the X1 coefficient - OLS")
hist(LASSO_model_coefs[,2], main = "Distribution of the X1 coefficient - LASSO")
hist(ridge_model_coefs[,2], main = "Distribution of the X1 coefficient - Ridge")
```



Properties of the OLS coefficients

Recall: The true relationship has a coefficient of 3.

The mean of the X1 coefficient from OLS is very close to the actual coefficient value in the population. It is unbiased. The variance of this coefficient is 0.064.

```
mean(ols_model_coefs[,2])
```

```
## [1] 2.997233
```

```
var(ols_model_coefs[,2])
```

```
## [1] 0.06358404
```

Properties of the LASSO and Ridge coefficients

The mean of the X1 coefficient from LASSO is biased - it has a mean of 2.68. The variance of this coefficient is smaller 0.034.

```
mean(LASSO_model_coefs[,2])
```

```
## [1] 2.683491
```

```
var(LASSO_model_coefs[,2])
```

```
## [1] 0.03383473
```

The mean of the X1 coefficient from Ridge is also biased and also smaller variance than the OLS coefficients.

```
mean(ridge_model_coefs[,2])
```

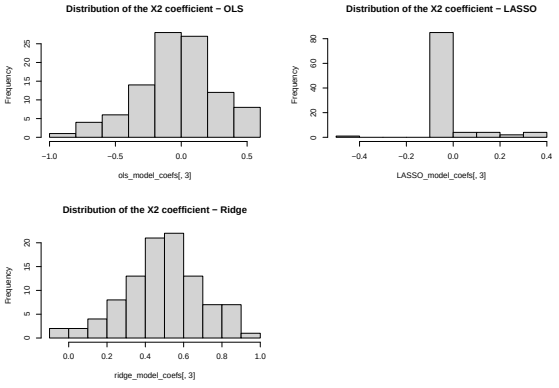
```
## [1] 2.262738
```

```
var(ridge_model_coefs[,2])
```

Sampling distribution comparison - X2

We can now compare the sampling distribution of some of these coefficients. The true X2 coefficient is 0.

```
par(mfrow = c(2,2))
hist(ols_model_coefs[,3], main = "Distribution of the X2 coefficient - OLS")
hist(LASSO_model_coefs[,3], main = "Distribution of the X2 coefficient - LASSO")
hist(ridge_model_coefs[,3], main = "Distribution of the X2 coefficient - Ridge")
```



Properties of the OLS coefficients

Recall: The true relationship has a coefficient of 3.

The mean of the X2 coefficient from OLS is fairly close to 0. The variance of this coefficient is 0.087.

```
mean(ols_model_coefs[,3])
```

```
## [1] -0.02687434
```

```
var(ols_model_coefs[,3])
```

```
## [1] 0.08745929
```

Properties of the LASSO and Ridge coefficients

The mean of the X2 coefficient from LASSO is also close to 0. The variance of this coefficient is much smaller than the variance of the OLS estimates.

```
mean(LASSO_model_coefs[,3])
```

```
## [1] 0.02280993
```

```
var(LASSO_model_coefs[,3])
```

```
## [1] 0.008747569
```

The mean of the X2 coefficient from Ridge is significantly biased. But it has a smaller variance than the OLS coefficients.

```
mean(ridge_model_coefs[,3])
```

```
## [1] 0.4925364
```

```
var(ridge_model_coefs[,3])
```

Section 3

Bias-Variance Trade off

The Bias-Variance Tradeoff

We will explore the relationship between Bias and Variance, and the idea of underfitting and overfitting our data.

If you do a quick search for "Bias Variance Tradeoff," you'll inevitably run into an equation that looks like this:

$$\text{Expected Prediction Error} = Bias^2 + Var + \sigma^2$$

The wikipedia page https://en.wikipedia.org/wiki/Bias%E2%80%93variance_tradeoff provides the derivation for this equation.

Bias-Variance Tradeoff - Definitions

- Expected Prediction Error, refers to the total squared error between the actual values and the fitted values for a *new* dataset.
- Error $= E(y - \hat{y})^2 =$ Expected value of (True Relationship + random error - fitted values)
- The Bias is the expected difference between the predicted value and the actual value for the new values of x.
- The Variance is the variance of the predicted values.
- Sigma is the standard deviation of the noise in generating data

Loess regression

We have not covered the concept explicitly in our course, but we will use local (loess) regression to illustrate the concept. The important things you need to know about local regression:

- the resulting model is not a mathematical function. There is no functional form like $\hat{y}_n = w_0 + w_1 x_n$
- the resulting model consists only of a sequence of estimated $\hat{y}$s for each x
- the $\hat{y}$s are estimated by calculating a weighted average (technically, weighted regression) of the y-values surrounding each x.
- you can control the smoothness of the loess curve by deciding how many values around x to include in the regression
  - using more values will result in a smoother fit
  - using fewer values will result in a more jagged fit (the extreme is using only one value, and the fitted $\hat{y}$ for each x_n is the corresponding y_n)
- we can make predictions by interpolating the values between points for which we have $\hat{y}$s

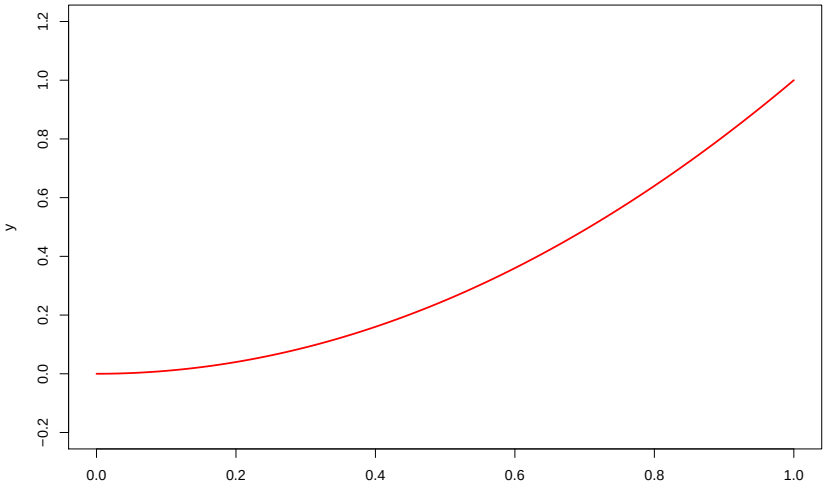
R has a nice function `loess()`, which performs local regression for us and allows us to use the fitted model to make predictions.

The true model

We begin by stating that the true relationship between x and y is a quadratic relationship.

```
options(width = 80)
f <- function(x) x^2  # true model
x <- seq(0,1, by = 0.01)
y <- f(x)
# plot of the true function:
plot(x, y, type = "l", col = "red", ylim = c(-0.2,1.2), lwd = 2)
```

The true model



Under fitting model - high bias, low variance

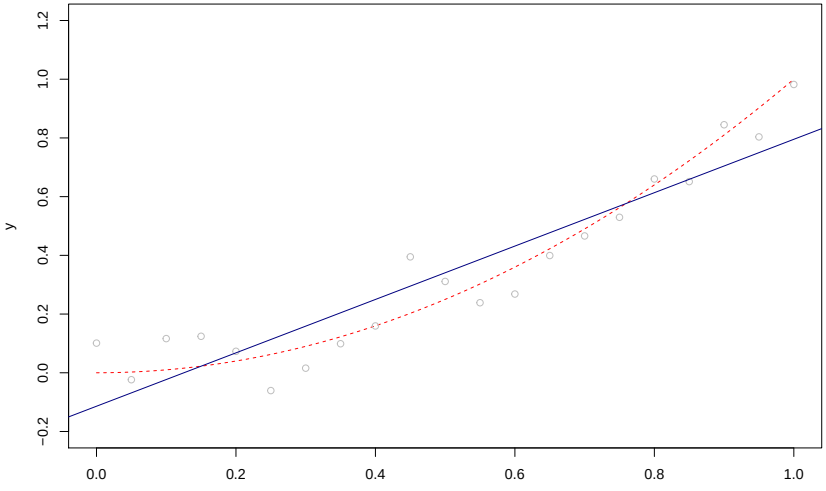
We will say our dataset consists of 21 evenly spaced values of x ranging from 0 to 1. We will generate the observed y value by adding random error to the true values associated with each x. We plot these points to our graph.

We will also fit a linear model to the data. This is clearly the wrong model to fit, but we can use it as a beginning illustration of the bias-variance tradeoff.

Under fitting model - high bias, low variance

```
n  <- 21      # sample size
sigma  <- 0.08    # sd of errors
obs_x <- seq(0, 1, length.out = n)

set.seed(0)
errors <- rnorm(n, 0, sigma)  # random errors, have standard deviation sigma
obs_y <- f(obs_x) + errors    # observed y = true_model + error
plot(x, y, type = "l", col = "red", ylim = c(-0.2,1.2), lty = 2) # true relationship
points(obs_x, obs_y, col = "gray")  # plot of n 'noisy' values
model1 <- lm(obs_y ~ obs_x)   # fit a linear model to the observed values
abline(model1, col = "navy")  # plot the fitted model
```

Under fitting model - high bias, low variance



Under fitting model - high bias, low variance

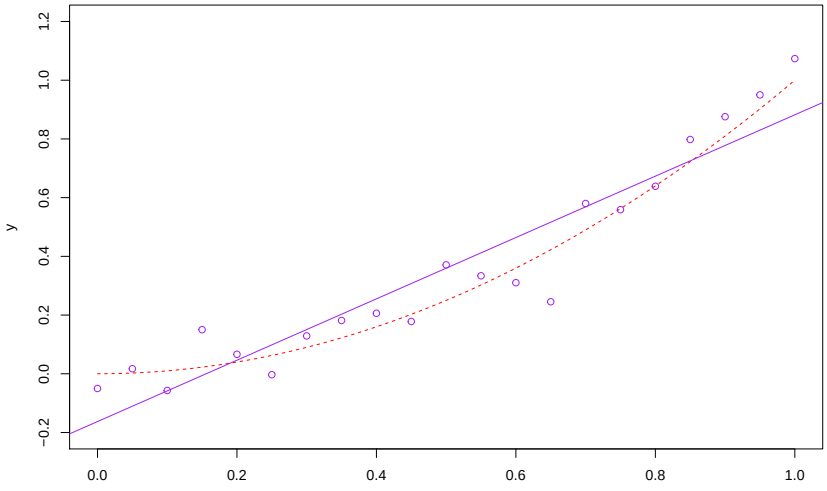
The model we produced is clearly wrong. At its best, the linear model simply can't fit the quadratic function. The difference between the model and the true relationship is the bias.

On the other hand, if we drew a new random sample and fit another linear model, we'd probably get a similar fit line. Yes, they will be slightly different, but they will not differ by much.

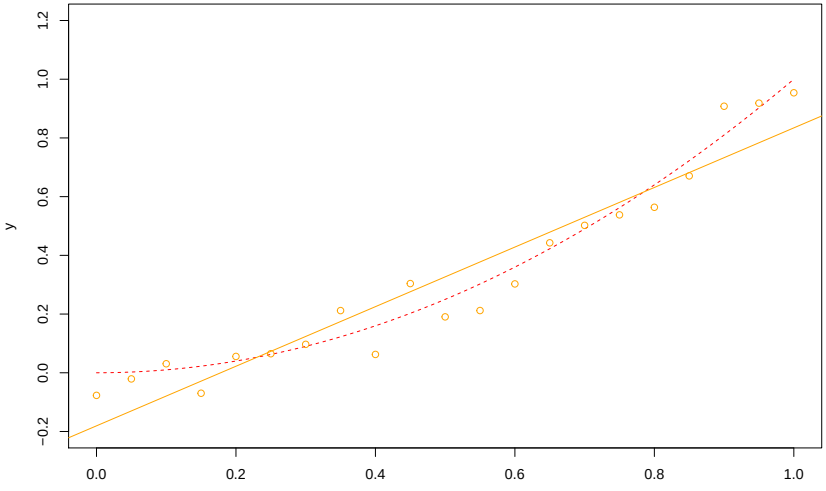
I've reproduced the same plot, but now I have sample 20 new points, and fit a different model in purple. We can see how they are different, but only slighlty.

So the linear models do a bad job of approximating the true model (high bias). However, the fit lines differ from each other by only a little bit (low variance).

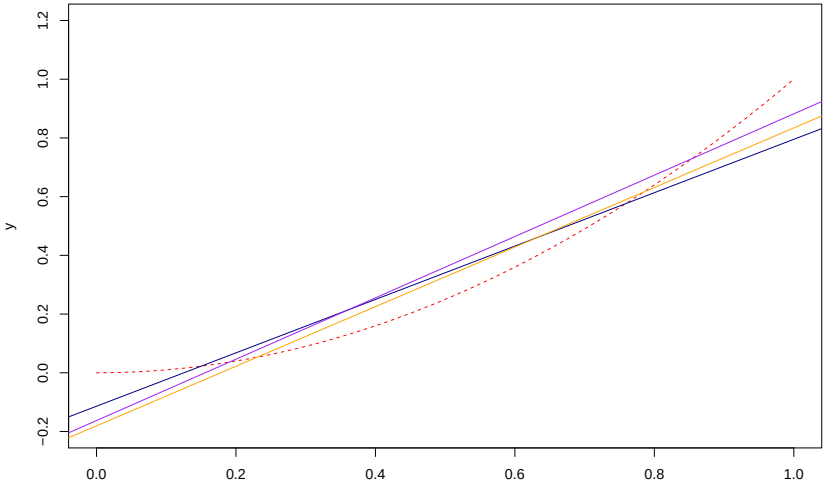
Under fitting model - new sample of data



Under fitting model - new sample of data



Comparison of linear models fit to different samples of random data

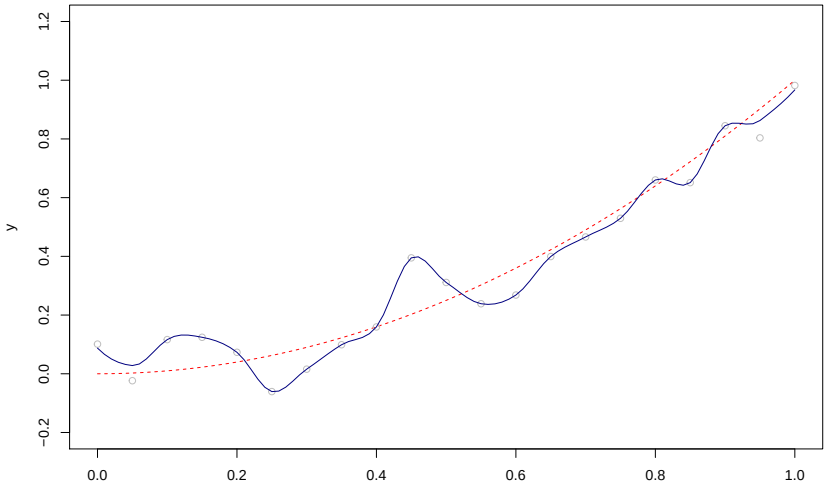


Overfitting model - low bias, high variance

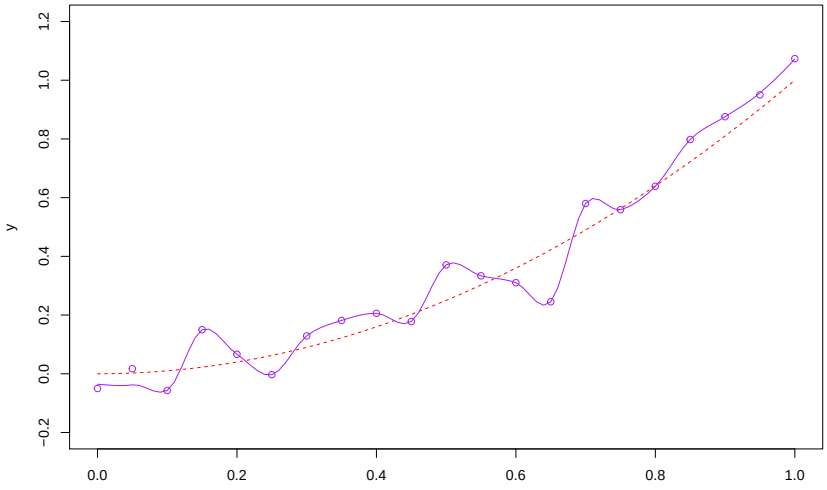
Now, I will fit a loess model that overfits the data.

```
set.seed(0)
errors <- rnorm(n, 0, sigma)        # random errors, have standard deviation s
obs_y <- f(obs_x) + errors          # observed y = true_model + error
plot(x,y, type = "l", col = "red", ylim = c(-0.2,1.2), lty = 2)  # plot of th
points(obs_x, obs_y, col = "gray")       # plot of n 'noisy' values

# fit a loess curve
model1 <- loess(obs_y ~ obs_x, span = 0.25) # overfitting loess model

# drawing the loess curve
lines(x, predict(model1, x), col = "navy", lwd = 1) # loess fit
```

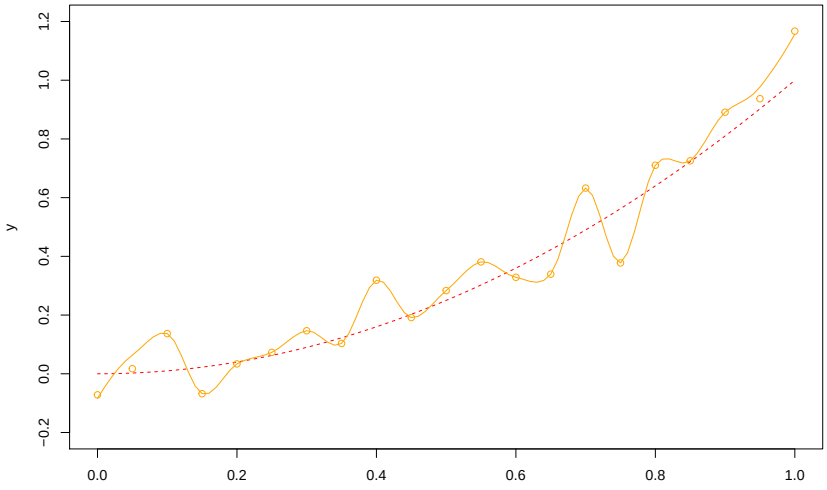
Overfitting model - low bias, high variance



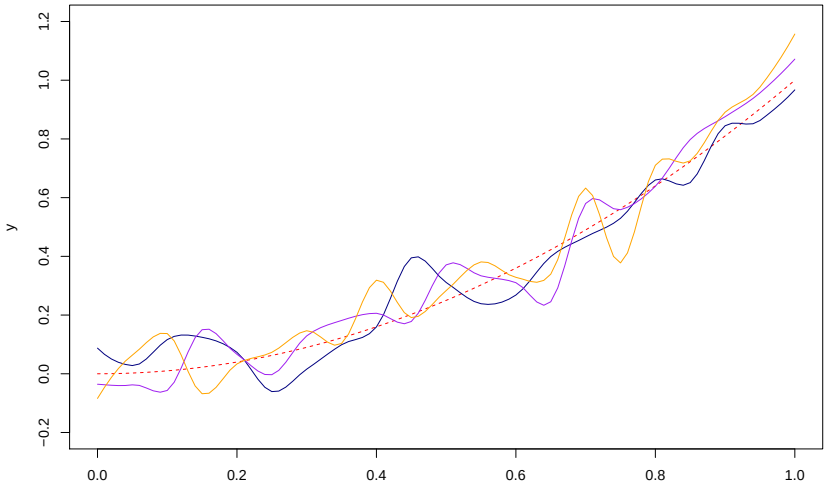
Overfitting model - new sample of data



Overfitting model - new sample of data



Comparison of loess models fit to different samples of random data



Comparison of loess models fit to different samples of random data

What we see here are models that do a very good job of fitting their observed data.

If you take the average between these models, you would probably get something close to the true model (low bias).

However, the predictions they make differ drastically from one model to the other (high variance).

Many Repetitions

The idea of the bias and variance revolve around the concept of how much our model depends on the current random sample, and how well can it generalize to a new sample.

- To estimate the variance of our predictions, we will do the following: We will draw 100 different random samples from the true model. For each random sample, we will fit a model. We will record all of the different fitted y-hat values we get for each x in our dataset. In total, there will be 100 different y-hats associated with each of the 21 x-values. For each of the 21 x-values, we will calculate the variance of the 100 y-hats.
- To estimate the bias, we will do the following: for each of the 100 different random samples, we will measure the difference between the y-hat from the true model ($f(x) = x^2$). We do this for all random samples and for each x-value. Again, there will be a total of 100 bias-values for each of the 21 x-values. We will average the 100 values, and will get 21 different expected bias values. The bias represents how much we expect our model to differ from the true value.

Many Repetitions

- To estimate the total squared error for a new sample, it would be found as follows: for each model, we would randomly generate a new set of y-values (at our 21 current locations of x). We will measure the difference between the y-hat from the actual y. We do this for all 100 models and for each value of x, producing 100 error values for each of the 21 x-values. We will calculate the mean squared value for each of these 21.
- It is important to note that the difference between the y-hat and the actual y is not the residual of the model. The residual of the model is the difference between y of the sample used to estimate the model and y-hat. The total error that we want is the difference between y of a new sample (drawn after the current model was estimated) and the y-hat.

Keep in mind that we are conducting only 100 repetitions, so our empirical estimates will not exactly match the results of the theoretic relationship.

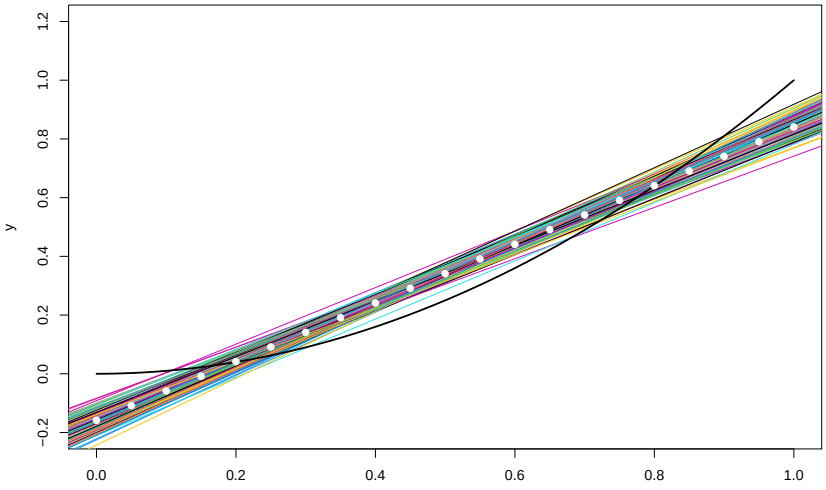
Many Repetitions

```
reps <- 100
model <- list()  # an empty list to store our models
y_hat <- matrix(rep(NA, reps*n), ncol = n)       # to store the y-hats
bias <- matrix(rep(NA, reps*n), ncol = n)        # to store values for bias
total_error <- matrix(rep(NA, reps*n), ncol = n) # to store total error
```

Underfitting

```
# plot the true relationship
plot(x,y, type = "l", col = "red", ylim = c(-0.2,1.2), lty = 2)  # true function

for(i in 1:reps){
  set.seed(i)
  errors <- rnorm(n, 0, sigma)
  obs_y <- f(obs_x) + errors  # add random errors to get obs_y
  points(obs_x, obs_y) # plot the points
  model[[i]] <- lm(obs_y ~ obs_x)  # fit a linear model and store it
  abline(model[[i]], col = i) # add the fitted line to the plot
  y_hat[i,] <- model[[i]]$fitted.values   # store the fitted y-hat into y_hat
  bias[i,] <- f(obs_x) - model[[i]]$fitted.values # store the difference between the
        # true value of f(x) and the fitted values in bias
  new_y <- f(obs_x) + rnorm(n, 0, sigma) # generate a new sample of y values
  total_error[i,] <- new_y - model[[i]]$fitted.values # store the difference between
        # y-values and the fitted values from the current model into the ith row of t
}
```

Underfitting



Underfitting

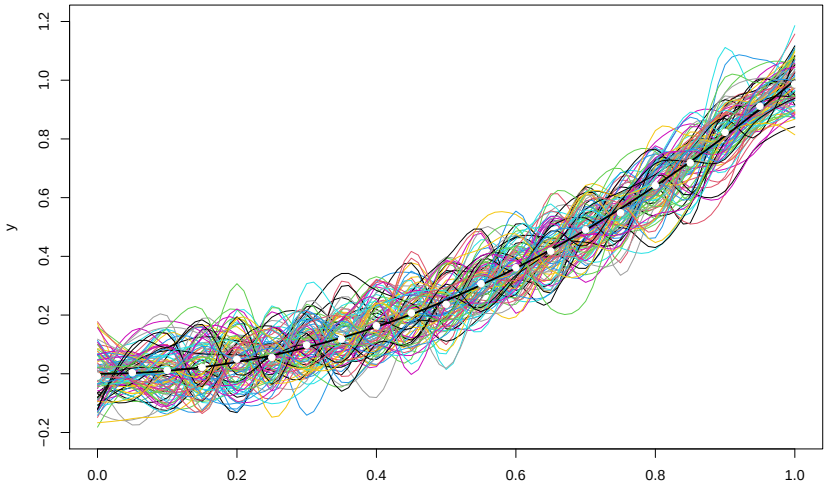
The black line represent the true relationship in the population.

The different colored straight lines represent all of the different linear models that were fit to different random samples of data.

The white dots represent the average of the predicted values from these different linear models.

We can see **the white dots do not fit the true relationship well** (high bias). On the other hand, **the different colored straight lines do not vary drastically from each other** (low variance).

Overfitting



Overfitting

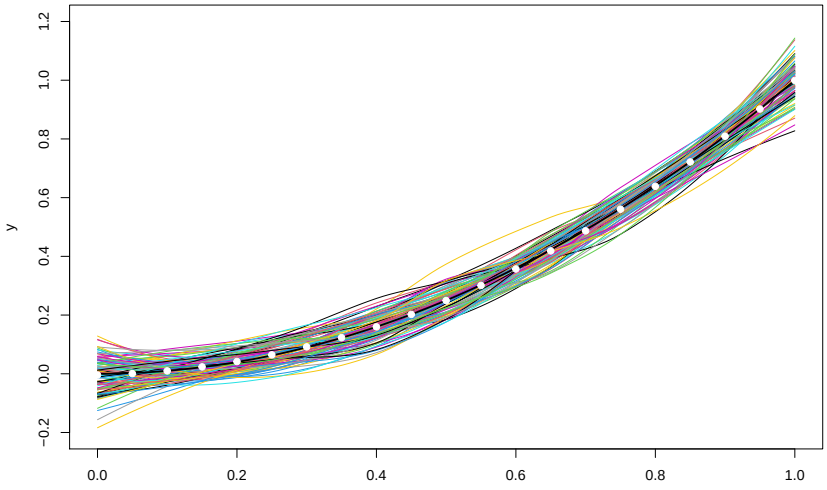
The black line represent the true relationship in the population.

The different colored straight lines represent all of the different loess models that were fit to different random samples of data.

The white dots represent the average of the predicted values from these different loess models.

We can see the **white dots do fit the true relationship well** (low bias). On the other hand, **the different colored lines vary drastically from each other** (high variance).

A balance between the two extremes?



A balance?

The goal is to achieve some balance between bias and variance.

You do not want to introduce too much bias in your predicted values.

At the same time, you do not want your models to have so much variance that the model will change dramatically with each random sample.

The tricky part is that the previous slides were generated using simulation. In real life, you do not get to have many different samples and get to compare all of these models against each other.