Stats 102B - Week 2, Lecture 3

Miles Chen, PhD

Department of Statistics

Week 2 Friday

UCLA

Section 1

Regularization

The Problem of Over-Fitting

In machine learning problems, we commonly run into the problem of over-fitting our data.

When we over-fit, we end up modeling odd specifics of our training data. We can predict our training data very well, but we cannot predict new data well.

We have covered **cross-validation** as a tool to avoid selecting a model that over-fits the data. It splits the available data into training/validation sets to measure a model's predictive abilities. A model's performance is not based only on how it performs fitting the data used to train parameters, but the performance is measured using data that was not part of the training process.

Regularization is another method that can be used to avoid over-fitting. It modifies the loss function to penalize complex models.

Modifying the Loss Function

For Ordinary Least-Squares regression, our loss function is the mean (or total) squared error.

$$\mathcal{L} = \frac{1}{N}(\mathbf{t} - \mathbf{X}\mathbf{w})^T(\mathbf{t} - \mathbf{X}\mathbf{w})$$

With this loss function, the 'best' model will be the model that minimizes the MSE. If we are guided only by this principle, we may end up selecting an overly complex model because the complex model has a lower MSE than a simpler model.

Regularization modifies the loss (or cost) function by including a **penalty term for the model's complexity**.

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

$\mathcal{L}'$ is the regularized loss. It is the sum of the OLS loss $\mathcal{L}$ and the penalty for complexity. The lambda λ weights how much complexity is penalized.

Analogues to Real Life

We do similar things in real life. For example, one day you may have to search for a place to live. Let's say you are considering two apartments:

- One apartment costs \$2000/month and the commute to work is 10 minute each way.
- The other apartment costs \$1700/month but the commute is 40 minutes each way.

If money is the only factor that matters, then the decision is easy: we pick the apartment with the lower cost.

However, most people value their time, and the apartment with lower rent comes with a penalty - additional time is spent each day on the commute.

Weighting the penalty

If you value your time greatly (e.g. you have an active social life outside of work) and/or hate commuting, then you'd weight the penalty of the longer commute much heavier. You select the apartment closer to work. i.e. the penalty of commuting outweighs the savings on rent.

On the other hand, if you don't do much with your time outside of work, and don't mind the commute (e.g. enjoy listening to podcasts and audiobooks), then the penalty for having a longer commute is greatly reduced, and you would select the apartment with lower rent.

$$\text{True Cost of Apartment} = \text{Rent} + \text{Preference Factor} \times \text{Commute Penalty}$$

We are still optimizing a Loss Function

Unlike the apartment selection example, we will not be comparing one model directly with another model.

What we have done is modify the loss function.

We are still optimizing the loss function.

But now, when we try find the best coefficients to use, we consider not only the resulting MSE but we also factor in the penalty.

The Regularized Least Squares Loss Function

We have modified our OLS Loss Function $\mathcal{L}$ by adding a regularization term. In the equation below, it's λ[Complexity], but more generally, we might write $\lambda R(f)$.

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

So how do we measure a model's complexity?

more: https://en.wikipedia.org/wiki/Regularized_least_squares

Please note: I am using the textbook's notation of calling the modified loss function $\mathcal{L}'$ (L prime). This does NOT represent the derivative of the loss function. This is the Ordinary Least Squares Loss function $\mathcal{L}$ plus a penalty term for the complexity.

One way to measure a model's complexity

Imagine two possible models:

- Model A fits a 4th order polynomial
- Model B fits a 5th order polynomial.

We know that the 5th order polynomial is more complex. It's function is:

$$f_A(x_n) = w_0 + w_1 x_n + w_2 x_n^2 + w_3 x_n^3 + w_4 x_n^4 + w_5 x_n^5$$

If we force $w_5 = 0$, then the model is reduced to the simpler 4th order polynomial.

$$f_B(x_n) = w_0 + w_1 x_n + w_2 x_n^2 + w_3 x_n^3 + w_4 x_n^4 + 0 x_n^5$$

So one possible way to measure model complexity is simply to count how many non-zero coefficients we have. (the ℓ_0 norm)

Optimizing the ℓ_0 norm is hard

If we use the count of non-zero elements in the vector $\mathbf{w}$, we are using the ℓ_0 norm, i.e. $||\mathbf{w}||_0$

$$\mathcal{L}' = \mathcal{L} + \lambda||\mathbf{w}||_0$$

This function, however, is very difficult to optimize (the loss function is not convex). You have to consider all possible permutations of which elements in $\mathbf{w}$ are non-zero, and when they are non-zero, what value they should be. For any model with more than a few terms, the number of calculations becomes staggering.

https://en.wikipedia.org/wiki/Regularized_least_squares#%E2%84%930_Penalization

Absolute value penalty - LASSO

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

Instead of counting which elements are non-zero, we can sum the absolute values of the elements in $\mathbf{w}$ (the ℓ_1 norm).

This serves as a measure of the complexity of the model.

Non-zero values of w_d will contribute to the penalty factor, and larger values will contribute even more to the penalty. If a coefficient is zero, then the variable is effectively not counted and does not contribute to the penalty factor.

So when we try to optimize the loss function, the reduction in the MSE must be greater than λ times the size of w for the variable in consideration.

This is known as the **LASSO** (https://en.wikipedia.org/wiki/Lasso_(statistics))

Squared value penalty - Ridge Regression

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

We can also use the sum of the squared values of the elements in $\mathbf{w}$ (the ℓ_2 norm-squared).

Larger values of w are heavily penalized. Small values of w (i.e. w less than 1) are penalized, but much less so.

This is known as **ridge regression**.

One effect is that ridge regression favors prefers when the values in $\mathbf{w}$ are similar in size to each other.

Wait. . . how does the size of a coefficient measure complexity?

The idea of measuring complexity based on the sum of the coefficients (or the squares of coefficients) seems strange.

After all, if two models have the same number of variables, aren't they equally complex? Why is one model more complex than the other model if it has a bigger coefficient? If bigger coefficients mean more "complexity," then can't we just change the unit of measure to shrink the coefficients and thus reduce the complexity? This doesn't seem to make sense.

These are valid objections.

Let me attempt to address these concerns

Centering and standardizing the input variables

It is almost always recommended that you center and standardize your X variables before fitting a LASSO model.

Centering means adding or subtracting a constant from a column of X values so that the mean of the column is 0.

Standardizing means multiplying a column of X values by a constant so that the standard deviation is 1.

(An alternative is to scale the variables X so all variables go from 0 to 1. This is used when a function cannot accept negative values.)

Standardizing variables addresses the concern that using different units will affect the size of the coefficients. After centering and standardizing, all X variables will have mean $= 0$ and variance $= 1$.

Still... how does the size of the coefficient measure complexity?

Thought experiment

Let's say there are two competing models for the same input matrix $\mathbf{X}$ which has been centered and standardized. Both models have four X variables. Both models have the same coefficients for variables X_1, X_2 and X_3.

Model A has a larger coefficient for X_4; let's say $w_4 = 1$. Model B has a tiny coefficient for X_4; let's say $w_4 = 0.01$.

With the exception of the coefficient for X_4, Model A and Model B are the same.

I argue that Model B is less complex than Model A because it has a smaller coefficient for X_4

Tiny coefficients

A tiny coefficient for a variable means changes in the predictor variable has little impact on the outcome variable.

If Model B has a tiny coefficient for X_4, then it is placing less importance to the variable X_4.

Even though both Model A and Model B use variable X_4, Model B is much closer to just ignoring X_4 altogether. And because it is that much closer to ignoring variable X_4, it is the simpler model.

Analogy: imagine two students involved in several clubs on campus. The one who is fully invested in all of their clubs has a busy schedule and lives a "complex" life. The other student may be "involved" in equal amount of clubs, but perhaps is only marginally invested in each club - skipping meetings whenever they feel like it. I argue this student's life is simpler and less complex.

So the weight placed on each predictor variable (after it has been standardized) can be used as a measure for how complex the model is.

Illustrative Example

I will generate some very simple 2D data. Because both X1 and X2 are on the same scale, I won't bother centering and scaling them.

```
set.seed(5)
x1 <- rep(1:4, each = 4)
x2 <- rep(1:4, 4)
truet <- 1.5 * x1 + 0.5 * x2  # no intercept in true model
t = truet + rnorm(16)      # add noise
model <- lm(t ~ 0 + x1 + x2)
OLSbest <- model$coefficients
OLSbest
```

```
##        x1        x2
## 1.4588837 0.4533397
```

OLS Loss Function

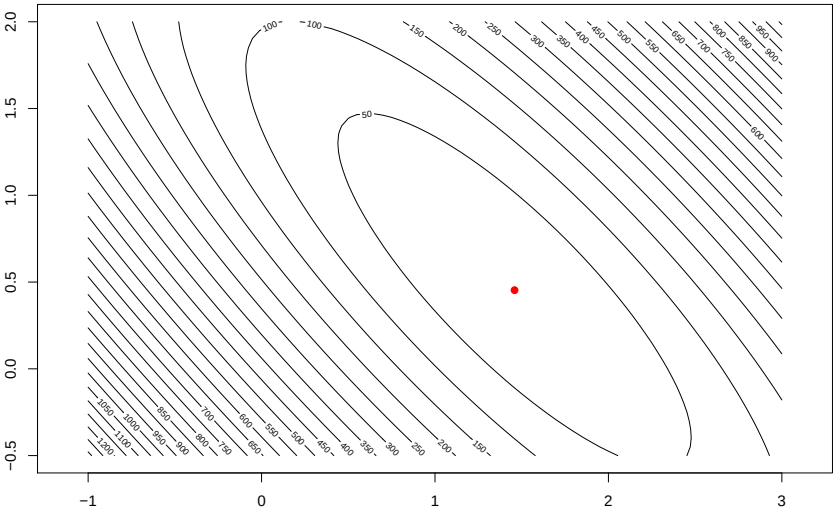
Recall: OLS Loss $= (\mathbf{t} - \mathbf{X}\mathbf{w})^T(\mathbf{t} - \mathbf{X}\mathbf{w})$

```
observed_data <- cbind(x1, x2, t)
OLSloss <- function(observed_data, w1, w2){
  w <- matrix(c(w1, w2))
  X <- observed_data[ , 1:2]
  t <- observed_data[ , 3, drop = FALSE]
  return(t((t - X %*% w)) %*% (t - X %*% w))
}

# a 'wrapper' function so I can use the optim() function
OLS_loss_op <- function(par) { OLSloss(observed_data, par[1], par[2]) }

optim(c(0,0), OLS_loss_op, method = "BFGS")$par
```

```
## [1] 1.4588833 0.4533399
```

Contours of the OLS Loss Function

```
w1_range <- seq(-1, 3, by = .05)
w2_range <- seq(-.5, 2, by = .05)
loss_contour_grid <- matrix(0, nrow = length(w1_range), ncol = length(w2_range))
for(i in seq_along(w1_range)){
  for(j in seq_along(w2_range)){
    loss_contour_grid[i,j] <- OLSloss(observed_data, w1_range[i], w2_range[j])
  }
}
contour(w1_range, w2_range, z = loss_contour_grid, asp = 1, nlevels = 30, main = "Cor
points(OLSbest[1], OLSbest[2], col = 'red', pch = 19)
```



L1 Penalty - LASSO

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

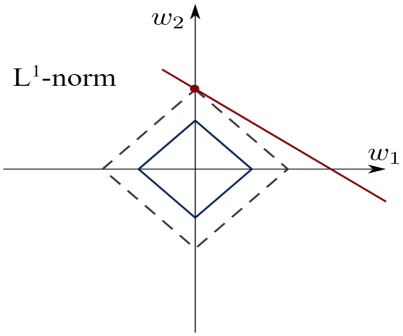
$\mathcal{L}$ is the OLS Loss

$||\mathbf{w}||_1$ is the sum of the absolute values of $\mathbf{w}$

```
L1loss <- function(observed_data, w1, w2, lambda){
  OLSloss(observed_data, w1, w2) + lambda * sum( abs( c(w1, w2) ))
}

# with a wrapper to use optim()
L1loss_op <- function(par, lambda = 1){
  L1loss(observed_data, par[1], par[2], lambda)
}
```

The $\mathcal{L}_1$ norm

If we only cared about minimizing the $\mathcal{L}_1$ norm, then the contours of the loss function would be concentric diamonds centered at the origin.



The actual loss function tries to minimize both the $\mathcal{L}_1$ norm and the OLS Loss. The resulting loss function then appears to be a bit of hybrid between the two extremes.

L1 Penalty: effect of lambda

```
print(OLSbest)
```

```
##        x1        x2
## 1.4588837 0.4533397
```

```
optim(c(0, 0), fn = L1loss_op, lambda = 10, method = "BFGS")$par
```

```
## [1] 1.4361564 0.4306124
```

```
optim(c(0, 0), fn = L1loss_op, lambda = 50, method = "BFGS")$par
```

```
## [1] 1.3452304 0.3397212
```

```
optim(c(0, 0), fn = L1loss_op, lambda = 100, method = "BFGS")$par
```

```
## [1] 1.2317712 0.2259379
```

L1 Penalty: effect of lambda

```
print(OLSbest)
```

```
##        x1        x2
## 1.4588837 0.4533397
```

```
optim(c(0, 0), fn = L1loss_op, lambda = 180, method = "BFGS")$par
```

```
## [1] 1.04979277 0.04424877
```

```
optim(c(0, 0), fn = L1loss_op, lambda = 300, method = "BFGS")$par
```

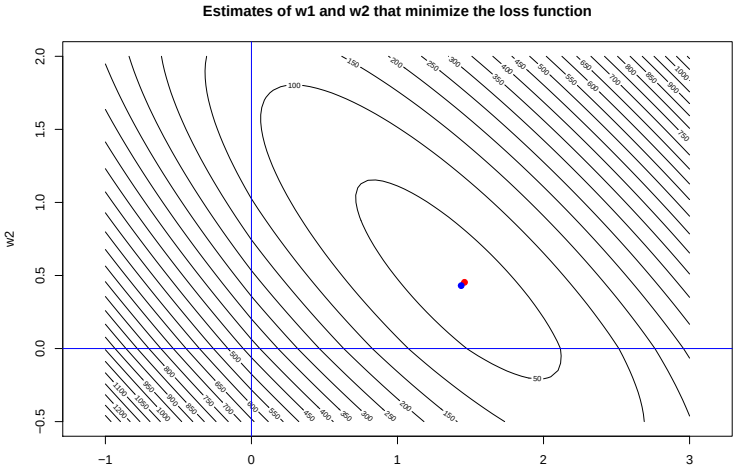
```
## [1] 0.5953567316 0.0001608288
```

```
optim(c(0, 0), fn = L1loss_op, lambda = 500, method = "BFGS")$par
```

```
## [1] 2.958159e-16 2.688235e-16
```

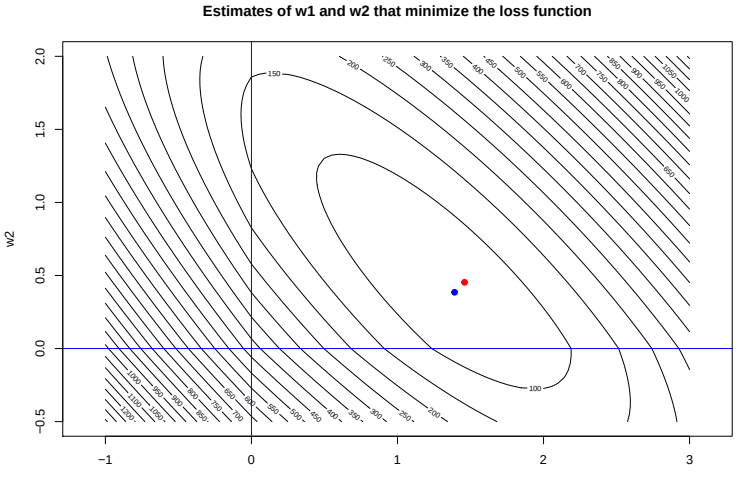
Contours of the LASSO Loss function; lambda = 10

```
lambda = 10; L1best <- optim(c(0,0),fn = L1loss_op, lambda = 10, method = "BFGS")$par
```



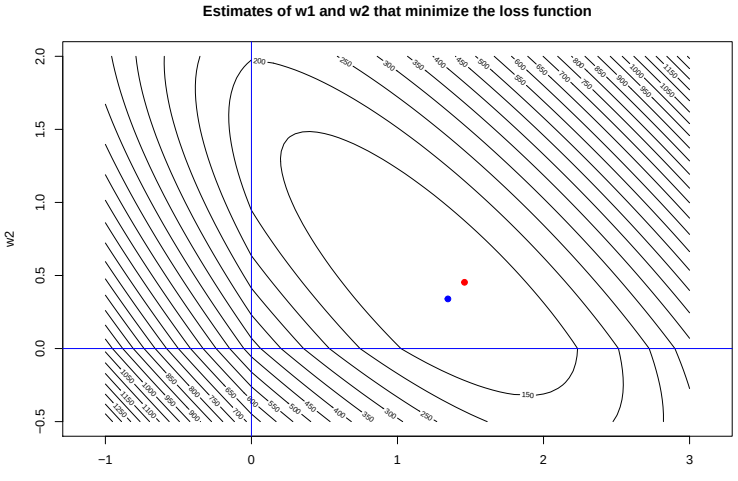
Contours of the LASSO Loss function; lambda = 30

```
lambda = 30; L1best <- optim(c(0,0),fn = L1loss_op, lambda = 30, method = "BFGS")$par
```



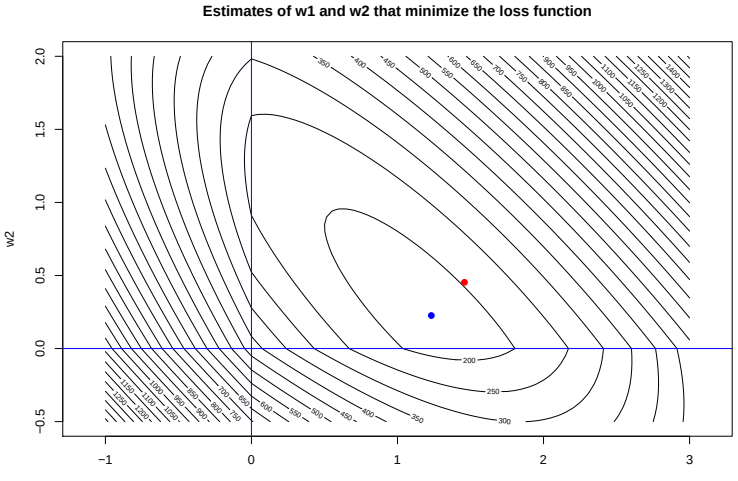
Contours of the LASSO Loss function; lambda = 50

```
lambda = 50; L1best <- optim(c(0,0),fn = L1loss_op, lambda = 50, method = "BFGS")$par
```



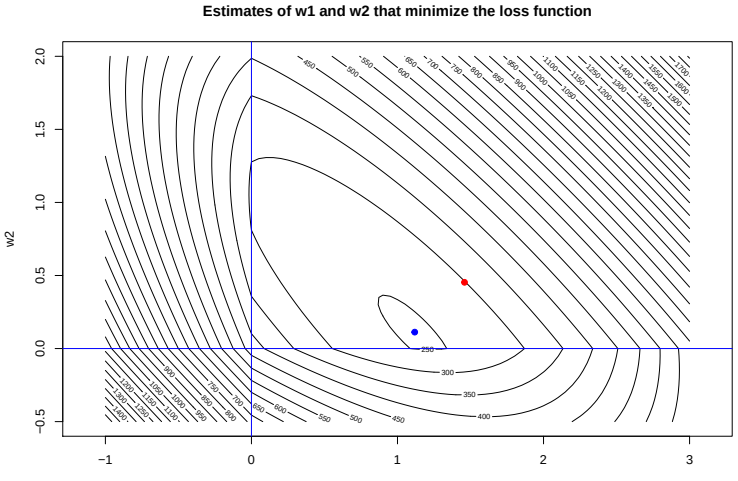
Contours of the LASSO Loss function; lambda = 100

```
lambda = 100; L1best <- optim(c(0,0),fn = L1loss_op, lambda = 100, method = "BFGS")$par
```



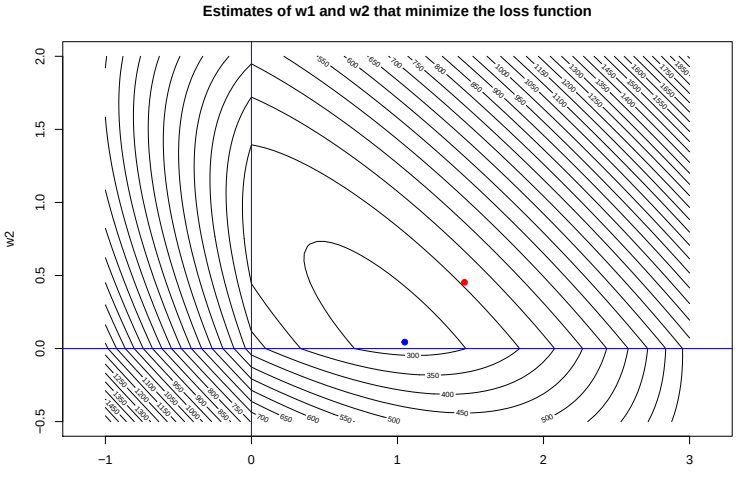
Contours of the LASSO Loss function; lambda = 150

```
lambda = 150; L1best <- optim(c(0,0),fn = L1loss_op, lambda = 150, method = "BFGS")$par
```



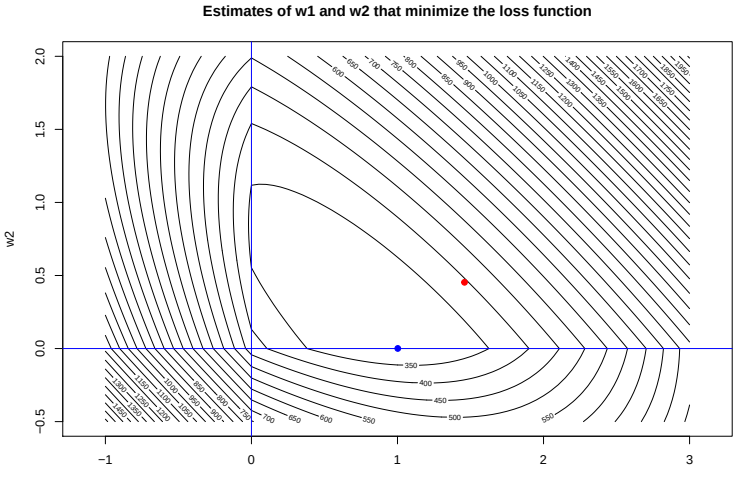
Contours of the LASSO Loss function; lambda = 180

```
lambda = 180; L1best <- optim(c(0,0),fn = L1loss_op, lambda = 180, method = "BFGS")$par
```



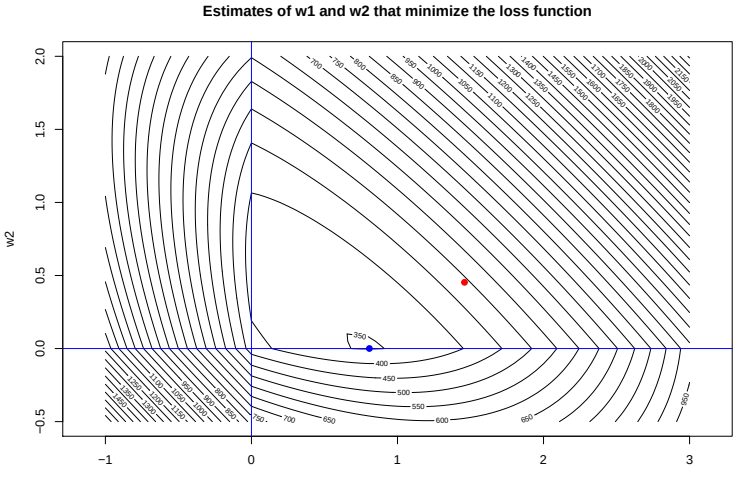
Contours of the LASSO Loss function; lambda = 200

```
lambda = 200; L1best <- optim(c(0,0),fn = L1loss_op, lambda = 200, method = "BFGS")$par
```



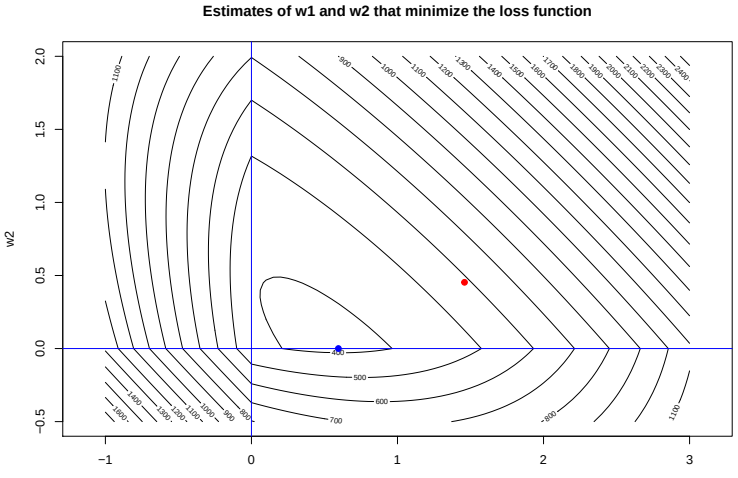
Contours of the LASSO Loss function; lambda = 250

```
lambda = 250; L1best <- optim(c(0,0),fn = L1loss_op, lambda = 250, method = "BFGS")$par
```



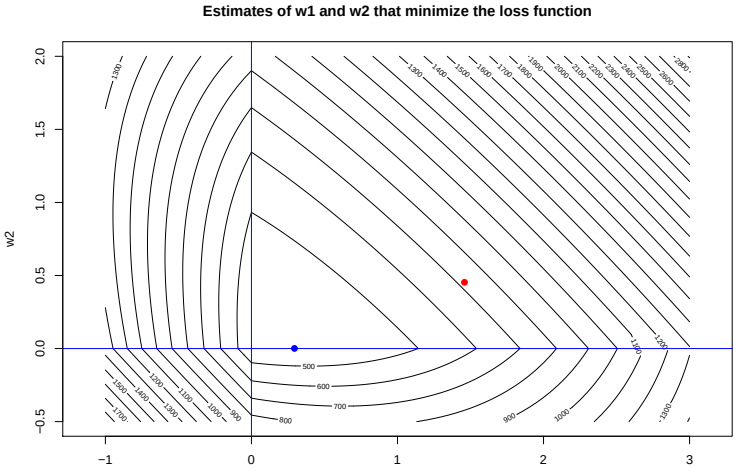
Contours of the LASSO Loss function; lambda = 300

```
lambda = 300; L1best <- optim(c(0,0),fn = L1loss_op, lambda = 300,  method = "BFGS")$par
```



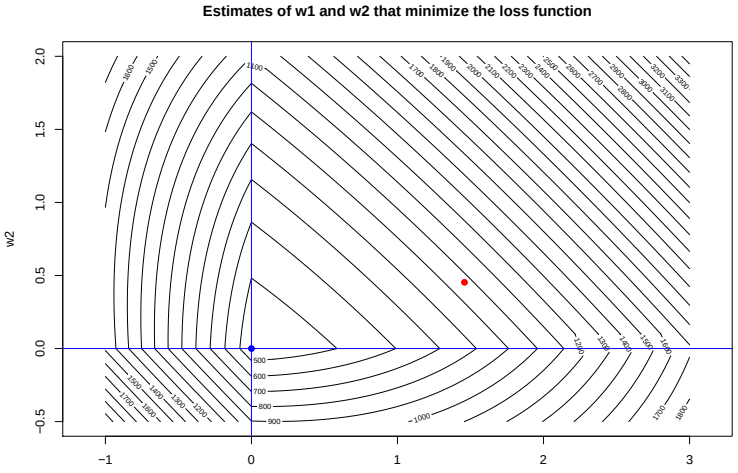
Contours of the LASSO Loss function; lambda = 370

```
lambda = 370; L1best <- optim(c(0,0),fn = L1loss_op, lambda = 370, method = "BFGS")$par
```



Contours of the LASSO Loss function; lambda = 500

```
lambda = 500; L1best <- optim(c(0,0),fn = L1loss_op, lambda = 500, method = "BFGS")$par
```



L1 Penalty Observations

As the lambda increased, the contour lines began having sharp corners and straigh-ish edges.

As lambda increased, we saw the optimal parameter values travel diagonally until it hit the x-axis, and then was 'pulled' towards 0.

The diagonal "motion" means that both the slope and intercept parameters decrease by the same amount when lambda increases. Both parameters decrease by the same amount until one of them shrinks to 0 (hits an axis). When this happens, we cannot decrease the penalty further by shrinking that coefficient (making it negative will add to the penalty). Only the non-zero coefficient continues to shrink.

L2 Penalty Function - Ridge Regression

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

$\mathcal{L}$ is the OLS Loss

$||\mathbf{w}||_2$ is the sum of the squares of $\mathbf{w}$

```
L2loss <- function(observed_data,w1, w2, lambda){
  OLSloss(observed_data, w1, w2) + lambda * sum( c(w1, w2)^2 )
}

# with a wrapper to use optim()
L2loss_op <- function(par, lambda = 1){
  L2loss(observed_data, par[1], par[2], lambda)
}
```

L2 Penalty: effect of lambda

```
print(OLSbest)
```

```
##        x1        x2
## 1.4588837 0.4533397
```

```
lambda = 10; optim(c(0, 0),fn = L2loss_op, lambda = lambda, method = "BFGS")$par
```

```
## [1] 1.2497226 0.5793606
```

```
lambda = 50; optim(c(0, 0),fn = L2loss_op, lambda = lambda, method = "BFGS")$par
```

```
## [1] 0.9227031 0.6354048
```

```
lambda = 100; optim(c(0, 0),fn = L2loss_op, lambda = lambda, method = "BFGS")$par
```

```
## [1] 0.7411221 0.5735315
```

L2 Penalty: effect of lambda

```
print(OLSbest)
```

```
##        x1        x2
## 1.4588837 0.4533397
```

```
lambda = 180; optim(c(0, 0),fn = L2loss_op, lambda = lambda, method = "BFGS")$par
```

```
## [1] 0.5761386 0.4755842
```

```
lambda = 300; optim(c(0, 0),fn = L2loss_op, lambda = lambda, method = "BFGS")$par
```

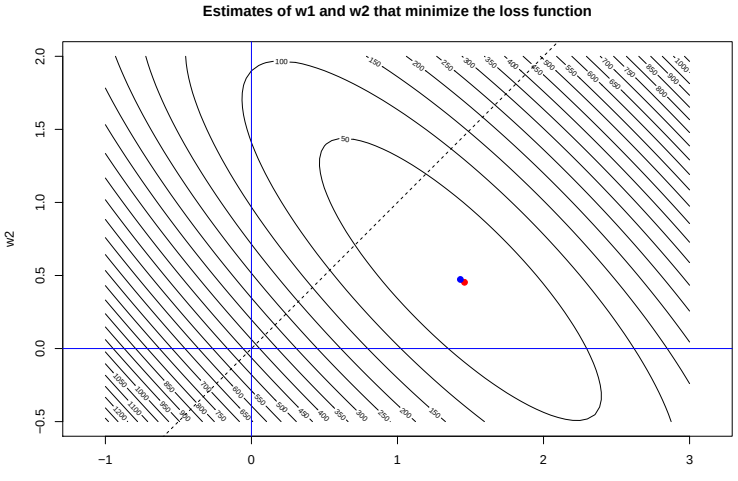
```
## [1] 0.4359320 0.3730855
```

```
lambda = 500; optim(c(0, 0),fn = L2loss_op, lambda = lambda, method = "BFGS")$par
```

```
## [1] 0.3114826 0.2728079
```

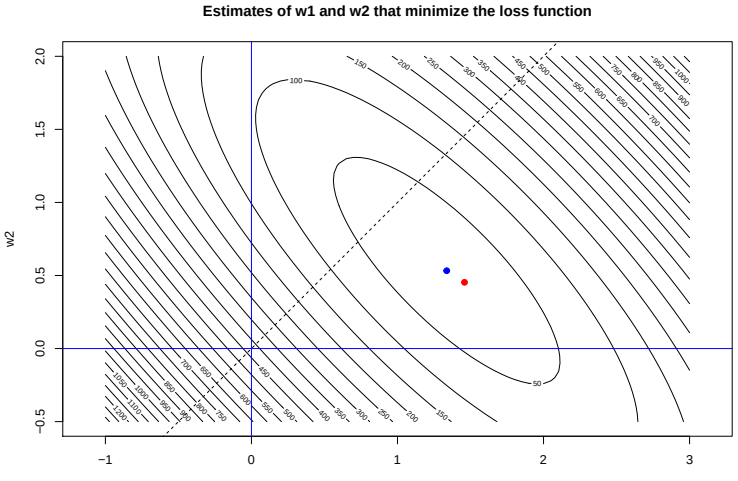
L2 Loss Contours; lambda = 1

```
lambda = 1; L2best <- optim(c(0,0),fn = L2loss_op, lambda = lambda, method = "BFGS")$par
```



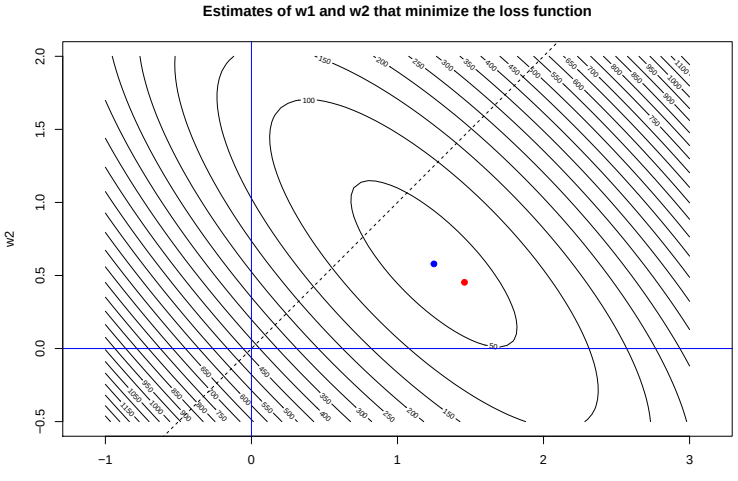
L2 Loss Contours; lambda = 5

```
lambda = 5; L2best <- optim(c(0,0),fn = L2loss_op, lambda = lambda, method = "BFGS")$par
```



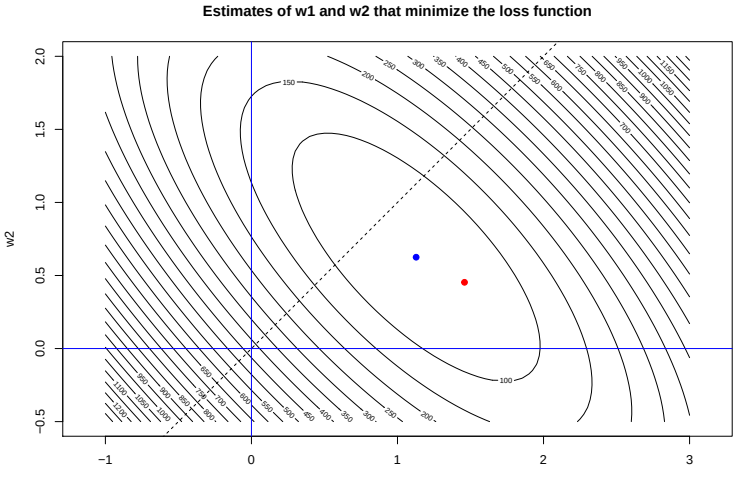
L2 Loss Contours; lambda = 10

```
lambda = 10; L2best <- optim(c(0,0),fn = L2loss_op, lambda = lambda, method = "BFGS")$par
```



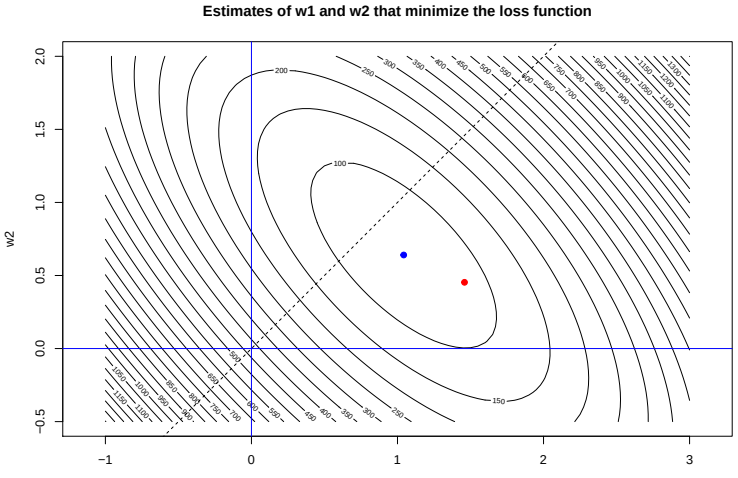
L2 Loss Contours; lambda = 20

```
lambda = 20; L2best <- optim(c(0,0),fn = L2loss_op, lambda = lambda, method = "BFGS")$par
```



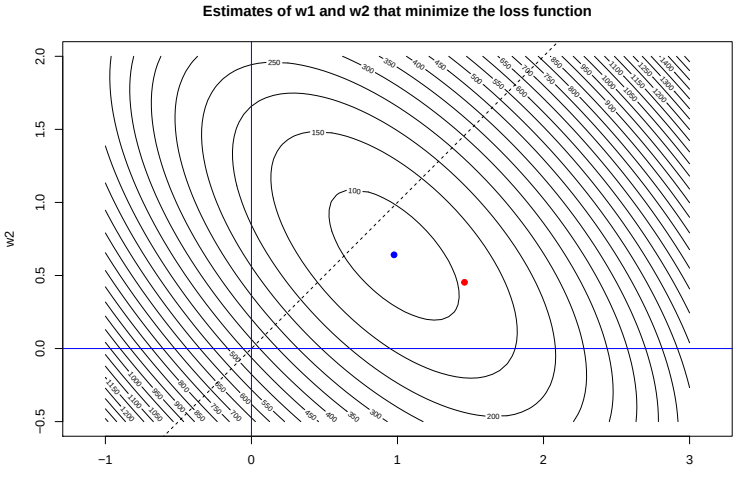
L2 Loss Contours; lambda = 30

```
lambda = 30; L2best <- optim(c(0,0),fn = L2loss_op, lambda = lambda, method = "BFGS")$par
```



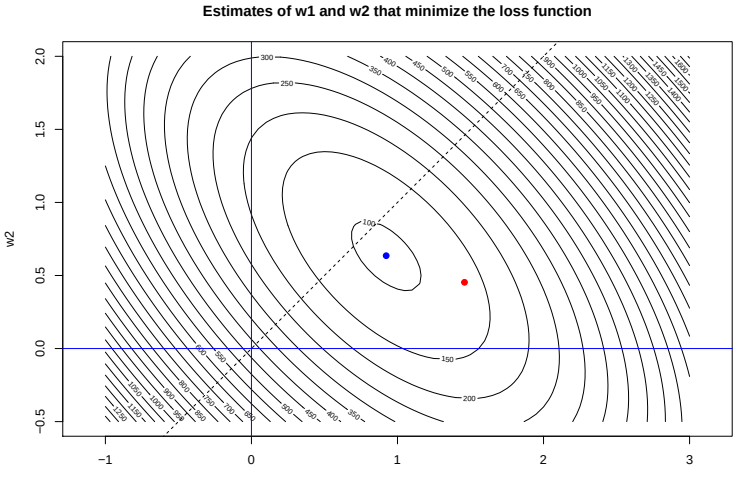
L2 Loss Contours; lambda = 40

```
lambda = 40; L2best <- optim(c(0,0),fn = L2loss_op, lambda = lambda, method = "BFGS")$par
```



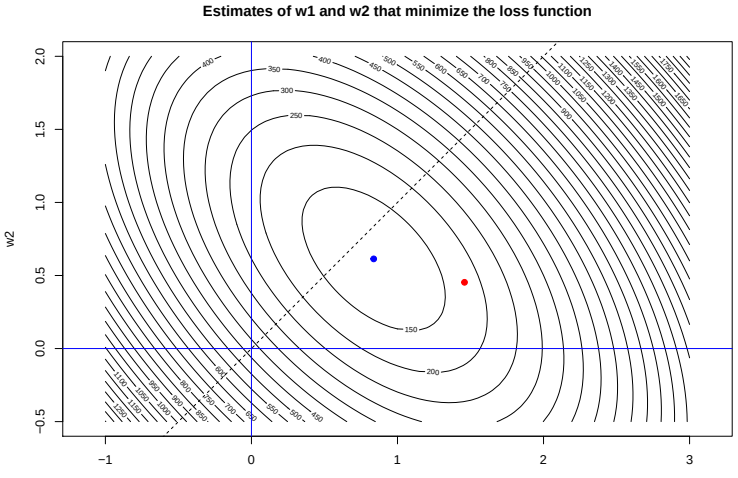
L2 Loss Contours; lambda = 50

```
lambda = 50; L2best <- optim(c(0,0),fn = L2loss_op, lambda = lambda, method = "BFGS")$par
```



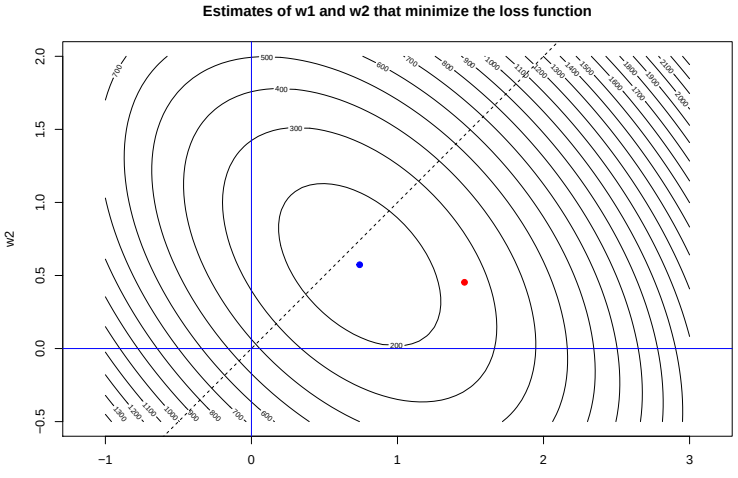
L2 Loss Contours; lambda = 70

```
lambda = 70; L2best <- optim(c(0,0),fn = L2loss_op, lambda = lambda, method = "BFGS")$par
```



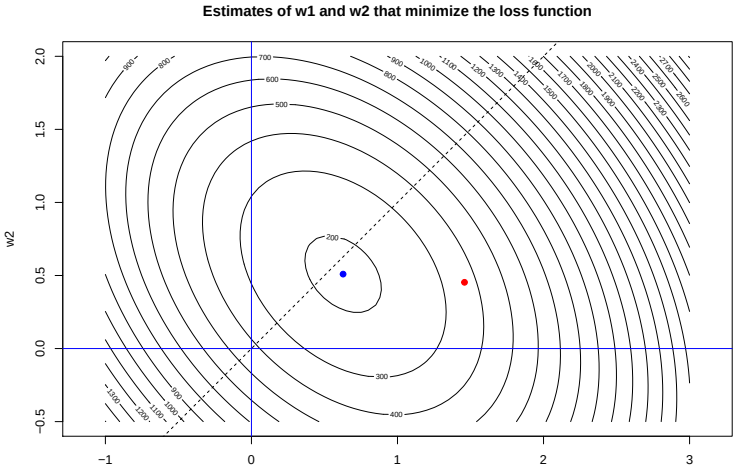
L2 Loss Contours; lambda = 100

```
lambda = 100; L2best <- optim(c(0,0),fn = L2loss_op, lambda = lambda, method = "BFGS")$par
```



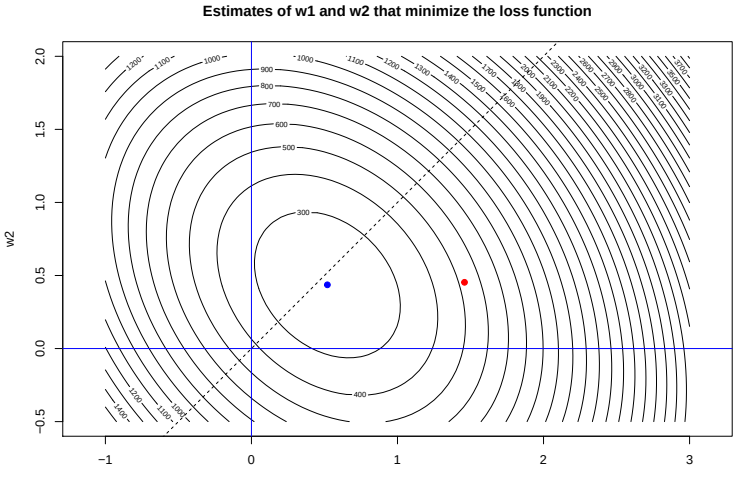
L2 Loss Contours; lambda = 150

```
lambda = 150; L2best <- optim(c(0,0),fn = L2loss_op, lambda = lambda, method = "BFGS")$par
```



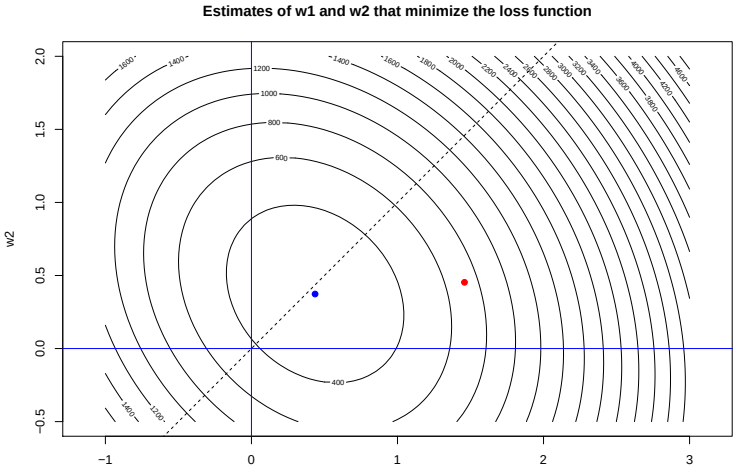
L2 Loss Contours; lambda = 220

```
lambda = 220; L2best <- optim(c(0,0),fn = L2loss_op, lambda = lambda, method = "BFGS")$par
```



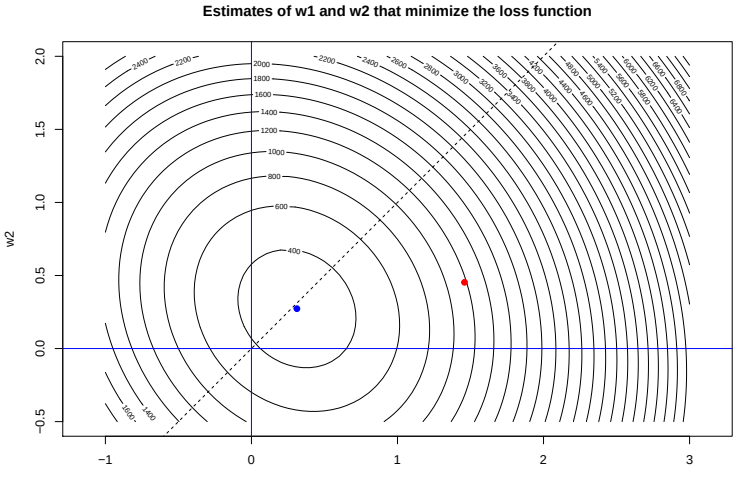
L2 Loss Contours; lambda = 300

```
lambda = 300; L2best <- optim(c(0,0),fn = L2loss_op, lambda = lambda, method = "BFGS")$par
```



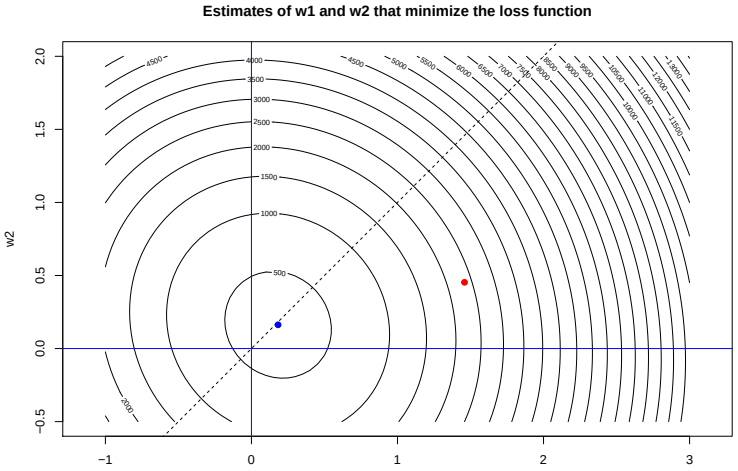
L2 Loss Contours; lambda = 500

```
lambda = 500; L2best <- optim(c(0,0),fn = L2loss_op, lambda = lambda, method = "BFGS")$par
```



L2 Loss Contours; lambda = 1000

```
lambda = 1000; L2best <- optim(c(0,0),fn = L2loss_op, lambda = lambda, method = "BFGS")$par
```



L2 Penalty Observations

As the lambda increased, the contour lines looked less elliptical and more circular.

As lambda increased, we saw the optimal parameter values travel towards the line $w1 = w2$ until it got close, and then was 'pulled' towards 0.

The larger coefficient (w1, the intercept) is penalized more heavily than the smaller coefficient. Shrinking the intercept, however, will cause the OLS loss function to grow. For this example's data, we can reduce the OLS loss function by increasing the w2 (slope) coefficient. Even though increasing the w2 coefficient will increase the penalty term, the amount the penalty term grows is not as big as the reduction caused by shrinking the w1 coefficient.

With squared terms, shrinking large values reduces the penalty much more than shrinking small terms. Going from 11 to 10 reduces the squared penalty from 121 to 100, while going from 3 to 2 reduces the squared penalty from 9 to 4. In some cases (like this example), we may see one large coefficient shrink while the smaller coefficient gets bigger.

Section 2

Regularization and variance of parameter estimates

Regularization reduces the variance of parameter estimates

When we estimate the slope and intercept parameter of a data set we must take into account that our observed data is random.

Two people might try to study the same population. If the each gather different random samples from the same population, their slope and intercept estimates will likely be slightly different from each other. This is **variation of parameter estimates**.

It is undesirable if there is great variation in parameter estimates.

We can see how regularization reduces the variation of parameter estimates.

Effect of Regularization on parameter estimates

We can see the effect of regularization on the selection of parameter estimates.

On the next slide, I will take generate 1000 different samples of data. They are all based on the true model, $z = 1.5x_1 + 0.5x_2$. Each sample is different because I add random noise.

For each of the 1000 randomly generated samples, I will find the best model according to:

- the OLS Loss function
- the OLS Loss + L1 penalty with $\lambda = 10$
- the OLS Loss + L1 penalty with $\lambda = 100$
- the OLS Loss + L1 penalty with $\lambda = 250$
- the OLS Loss + L1 penalty with $\lambda = 500$
- the OLS Loss + L2 penalty with $\lambda = 10$
- the OLS Loss + L2 penalty with $\lambda = 100$
- the OLS Loss + L2 penalty with $\lambda = 250$
- the OLS Loss + L2 penalty with $\lambda = 500$

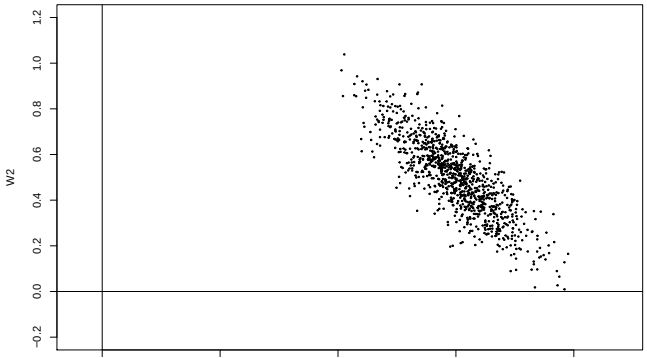
This will allow us to see what kind of variation we get in the parameter estimates

Code

```
N = 1000; x1 <- rep(1:4, each = 4); x2 <- rep(1:4, 4)
truez <- 1.5 * x1 + 0.5 * x2  # no intercept in true model
OLSbest_m <- matrix(NA, nrow = 1000, ncol = 2)
L1best_m_l10 <- OLSbest_m
L2best_m_l10 <- OLSbest_m
L1best_m_l100 <- OLSbest_m
L2best_m_l100 <- OLSbest_m
L1best_m_l250 <- OLSbest_m
L2best_m_l250 <- OLSbest_m
L1best_m_l500 <- OLSbest_m
L2best_m_l500 <- OLSbest_m
for(i in 1:N){
  set.seed(i)
  z = truez + rnorm(16)     # add noise
  observed_data <- cbind(x1,x2,z)
  model <- lm(z ~ 0 + x1 + x2)
  OLSbest_m[i,] <- model$coefficients
  L1best_m_l10[i,] <- optim(c(0,0), fn = L1loss_op, lambda = 10, method = "BFGS")$par
  L2best_m_l10[i,] <- optim(c(0,0), fn = L2loss_op, lambda = 10, method = "BFGS")$par
  L1best_m_l100[i,] <- optim(c(0,0),fn = L1loss_op, lambda = 100, method = "BFGS")$par
  L2best_m_l100[i,] <- optim(c(0,0),fn = L2loss_op, lambda = 100, method = "BFGS")$par
  L1best_m_l250[i,] <- optim(c(0,0),fn = L1loss_op, lambda = 250, method = "BFGS")$par
  L2best_m_l250[i,] <- optim(c(0,0),fn = L2loss_op, lambda = 250, method = "BFGS")$par
  L1best_m_l500[i,] <- optim(c(0,0),fn = L1loss_op, lambda = 500, method = "BFGS")$par
  L2best_m_l500[i,] <- optim(c(0,0),fn = L2loss_op, lambda = 500, method = "BFGS")$par
}
```

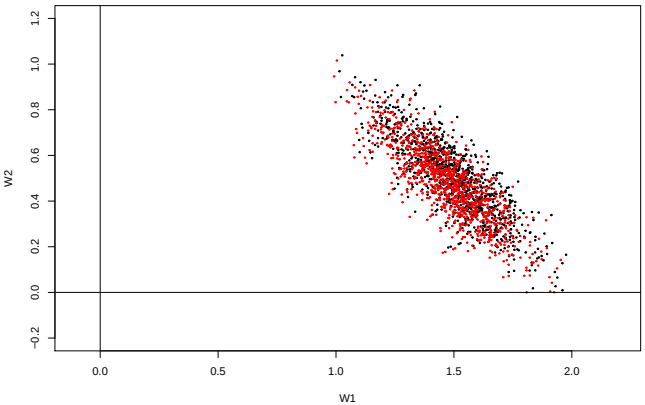
Variation in OLS Parameter Estimates

The true model has $w_1 = 1.5$ and $w_2 = 0.5$. Because of randomness, when we draw 16 random points, we get different estimates of w_1 and w_2. This plot shows all of the different estimates of w_1 and w_2 that we found for 1000 different random datasets of 16 points. The "cloud" of estimates is centered at the true values of $w_1 = 1.5$ and $w_2 = 0.5$.



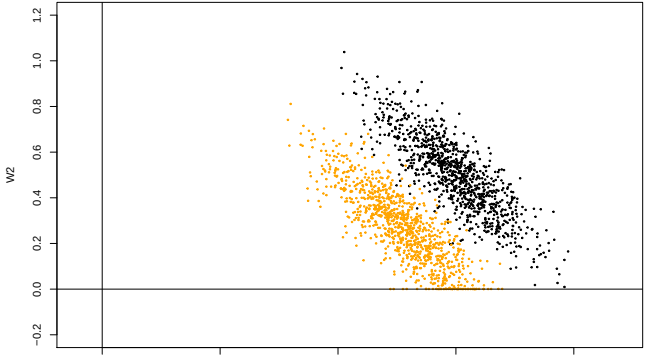
L1 Penalized Estimates (OLS - Black, $\lambda = 10$ red)

This plot shows the original cloud of estimates in black and the LASSO estimates in red. We can see the cloud has shifted diagonally toward the origin where w1 and w2 both equal 0.



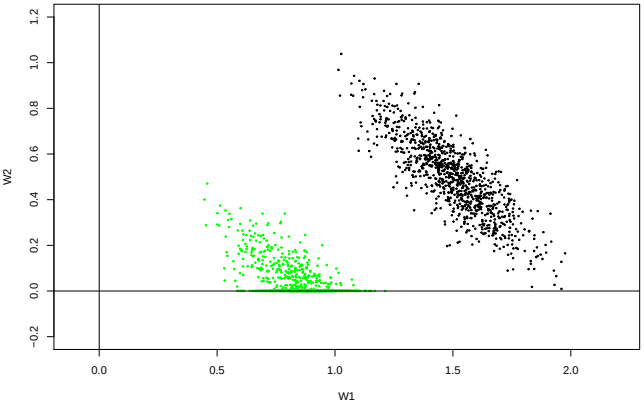
L1 Penalized Estimates (OLS - Black, $\lambda = 100$ orange)

This plot shows the original cloud of estimates in black and the LASSO estimates with a larger $\lambda = 100$ in orange. We can see the cloud has shifted diagonally toward the origin where w1 and w2 both equal 0. Where part of the cloud would be negative, those estimates of w2 are set to 0.



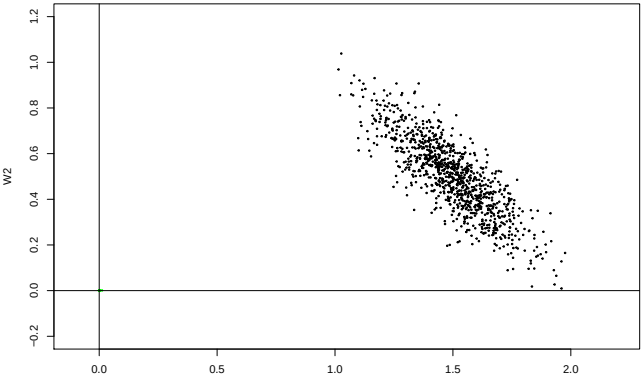
L1 Penalized Estimates (OLS - Black, $\lambda = 250$ green)

If we increase lambda futher $\lambda = 250$ in green, the cloud has shifted diagonally even more. Even more estimates of w2 are set to 0.

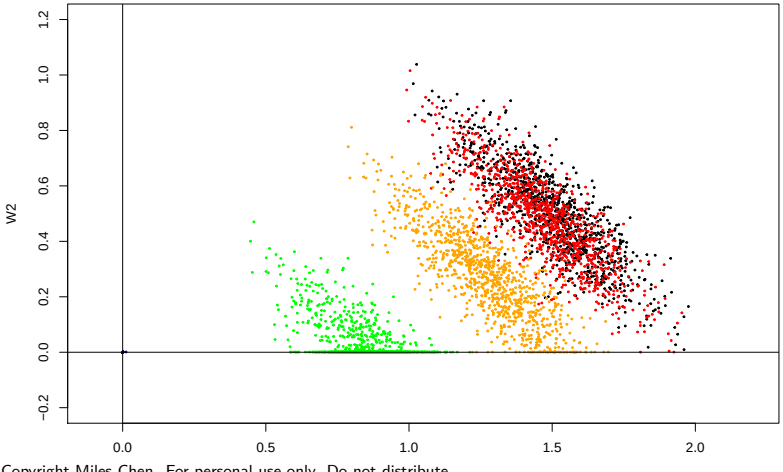


L1 Penalized Estimates (OLS - Black, $\lambda = 500$ blue)

If we increase lambda futher $\lambda = 500$ in blue, nearly the entire cloud has moved to the origin. At this location, any value of w_1 or w_2 increases the penalty factor so much, that it is better to leave w_1 and w_2 equal to 0, even if it means that the OLS loss is large.

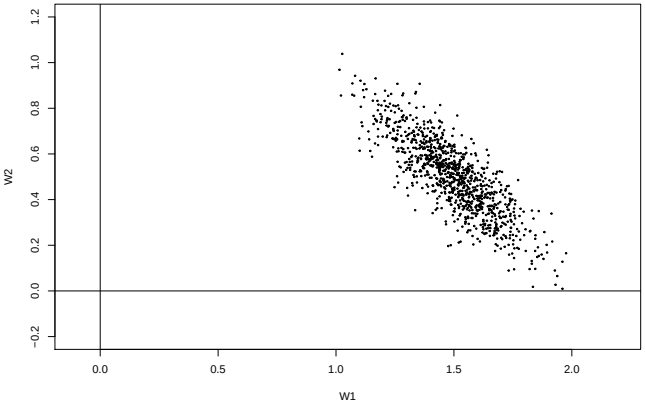


L1 Penalized Estimates (OLS - Black, $\lambda = 10$ red, $\lambda = 100$ orange,$\lambda = 250$ green, $\lambda = 500$ blue)



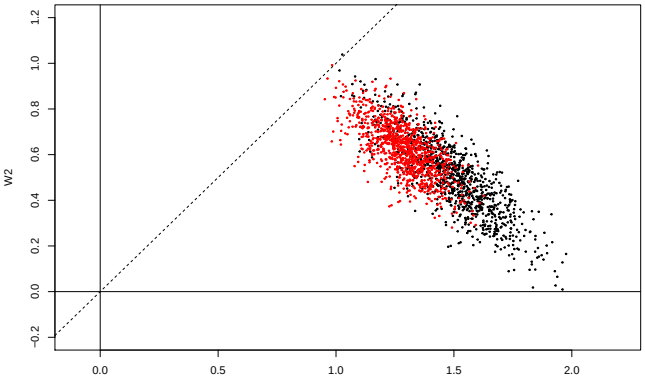
Variation in OLS Parameter Estimates

The "cloud" of OLS estimates is centered at the true values of $w_1 = 1.5$ and $w_2 = 0.5$.



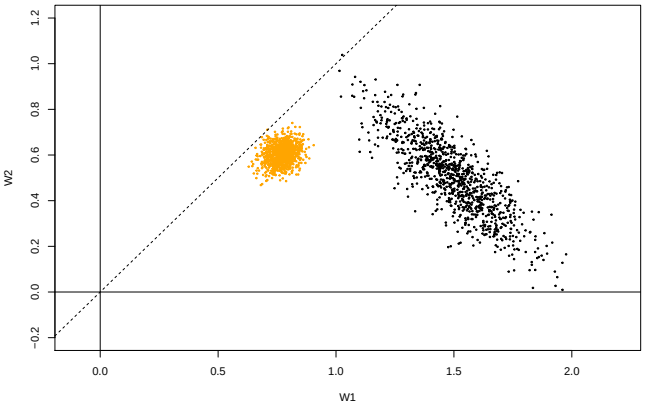
L2 Penalized Estimates (OLS - Black, $\lambda = 10$ red)

This plot shows the original cloud of OLS estimates in black and the Ridge Regression estimates with $\lambda = 10$ in orange. We can see that the cloud has moved slightly and has also shrunk, representing a decrease in variation.

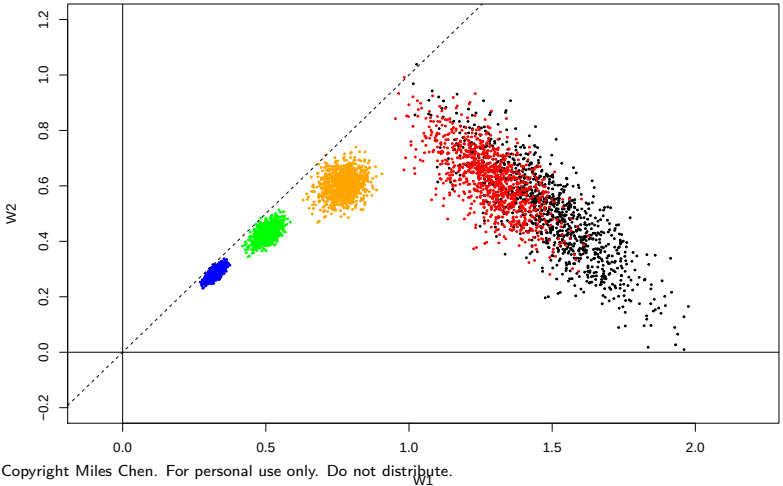


L2 Penalized Estimates (OLS - Black, $\lambda = 100$ orange)

We increase $\lambda = 100$. We can see the cloud has shifted more and the size of the cloud has shrunk even more.



L2 Penalized Estimates (OLS - Black, $\lambda = 10$ red, $\lambda = 100$ orange,$\lambda = 250$ green, $\lambda = 500$ blue)



Observations

The L1 penalty seems to move the 'cloud' of estimates closer to one of the axes. When the 'cloud' bumps into the axis, the values 'zero-out'. As lambda increases, the cloud moves closer to the origin, until all estimates are (0,0)

The L2 penalty seems to move the 'cloud' of estimates closer to the line $w1 = w2$. As the 'cloud' of w estimates moves, it displays smaller variation. bumps into the axis, the values 'zero-out'. As lambda increases, the cloud moves closer to the origin, until all estimates are (0,0)