Regularized Regression

Data Sets

```
attrition <- attrition %>% mutate_if(is.ordered, factor, order = F)
attrition.h2o <- as.h2o(attrition)
```

```
set.seed(123)

ames <- AmesHousing::make_ames()
ames.h2o <- as.h2o(ames)

ames.split <- initial_split(ames, prop =.7, strata = "Sale_Price")

ames.train <- training(ames.split)
ames.test <- testing(ames.split)
```

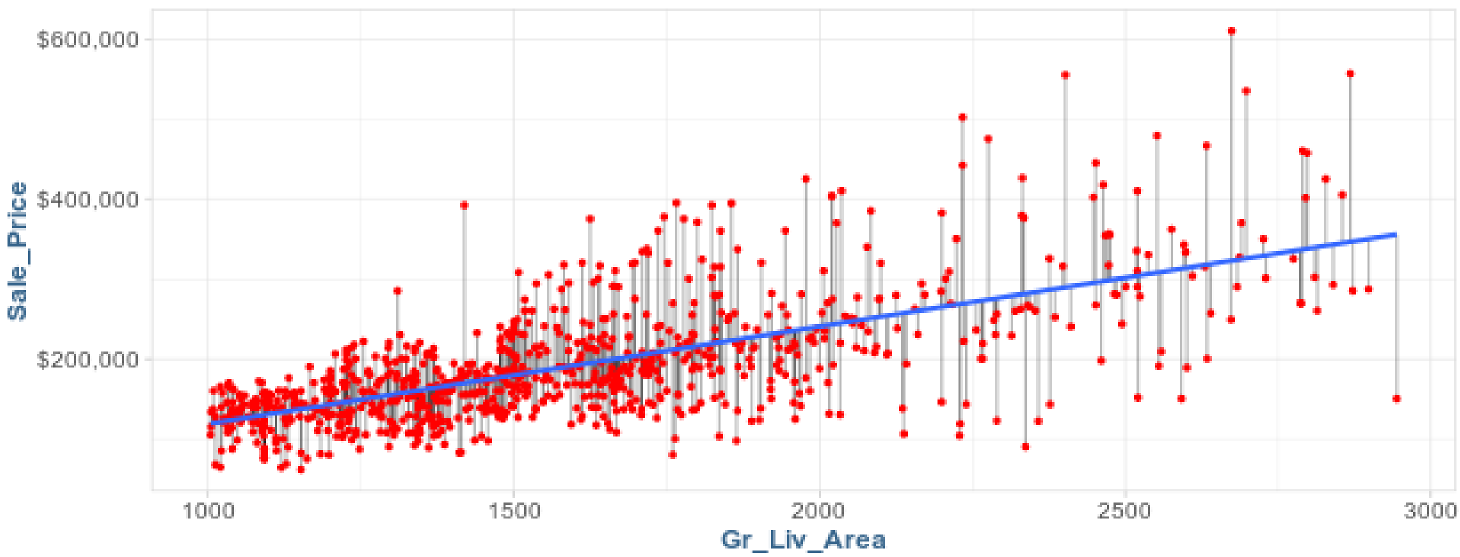
Overview

Regularization methods provide a means to constrain or regularize the estimated coefficients, which can reduce the variance and decrease out of sample error.

Why regularize?

Linear Model:

```
ames_sub <- ames.train %>%
   filter(Gr_Liv_Area > 1000 & Gr_Liv_Area < 3000) %>%
   sample_frac(.5)

model1 <- lm(Sale_Price ~ Gr_Liv_Area, data = ames_sub)

model1 %>%
   broom::augment() %>%
   ggplot(aes(Gr_Liv_Area, Sale_Price)) +
   geom_segment(aes(x = Gr_Liv_Area, y = Sale_Price,
                    xend = Gr_Liv_Area, yend = .fitted),
                alpha = 0.3) +
   geom_point(size = 1, color = "red") +
   geom_smooth(se = F, method = "lm") +
   scale_y_continuous(labels = scales::dollar)
```



Linear assumptions:

1.) Linear relationship

2.) More observations than features (n > p)

3.) Little to no multicolinearity

4.) Homoscedasticity (constant error variance)

When linear assumptions break-down, especially with large p, regularization methods are useful.

- Linear OLS: min SSE = $\sum_{i=1}^{n}(y_i - \hat{y})^2$

Feature Selection

In OLS, we have “hard threshold” methods for variable selection (forward selection, backward elimination, step-wise)

More modern approach is called “soft thresholds”, which slowly pushes the effects of irrelevant features toward zero, and in some cases will zero out entire coefficients.

With wide data sets (or ones that exhibit multicolinearity) we have regularization methods (or penalized models / shrinkage methods).

Reduces variance at expense being unbiased.

Regularization parameter:

- min SSE = $\sum_{i=1}^{n}(y_i - \bar{y})^2 + P$

Basically, we can think of the regularization parameter as a constraint to the size of the coefficients such that the only way the coefficients can increase is if we experience a comparable decrease in the model’s loss function.

Three common regularization methods:

1.) Ridge

2.) Lasso (or LASSO)

3.) Elastic net (or ENET), which is a combination of ridge and lasso.

Ridge Penalty

Ridge penalty: $\lambda \sum_{j=1}^{p} \beta_j^2$

So the minimization function becomes:

$\sum_{i=1}^{n} (y_i - \hat{y})^2 + \lambda \sum_{j=1}^{p} \beta_j^2$

The size of this penalty, referred to as L^2 (or Euclidiean) norm, can take on a wide range of values, which is controlled by the tuning parameter λ.

When L^2 is zero, there is no effect (reverts back to OLS).

However, as $\lambda \rightarrow$ inf, the penalty becomes large and forces the coefficients toward zero (but not all the way).

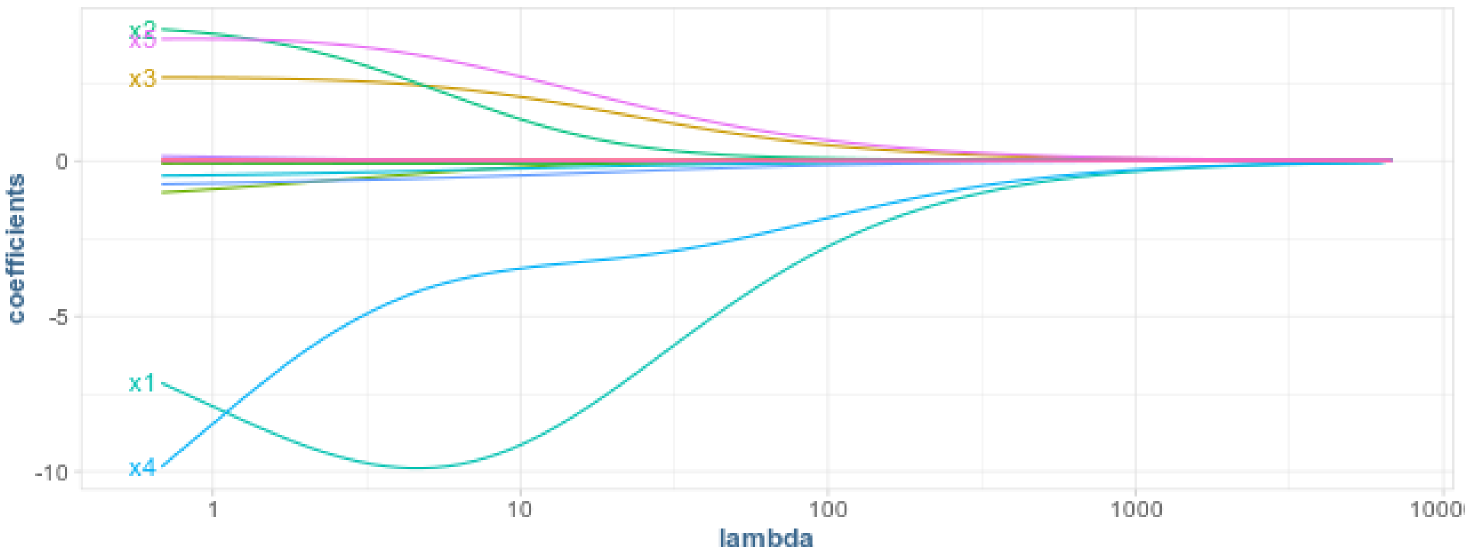
Ridge Penalty visually:

```
boston_train_x <- model.matrix(cmedv ~ ., pdp::boston)[, -1]
boston_train_y <- pdp::boston$cmedv

# model
boston_ridge <- glmnet::glmnet(
   x = boston_train_x,
   y = boston_train_y,
   alpha = 0
)

lam <- boston_ridge$lambda %>%
   as.data.frame() %>%
   mutate(penalty = boston_ridge$a0 %>% names()) %>%
   rename(lambda = ".")

results <- boston_ridge$beta %>%
   as.matrix() %>%
   as.data.frame() %>%
   rownames_to_column() %>%
   gather(penalty, coefficients, -rowname) %>%
   left_join(lam)
```

```
Joining, by = "penalty"
```

```
result_labels <- results %>%
   group_by(rowname) %>%
```

```
  filter(lambda == min(lambda)) %>%
  ungroup() %>%
  top_n(5, wt = abs(coefficients)) %>%
  mutate(var = paste0("x", 1:5))

ggplot() +
  geom_line(data = results, aes(lambda, coefficients, group = rowname, color = rowname), show.
  scale_x_log10() +
  geom_text(data = result_labels, aes(lambda, coefficients, label = var, color = rowname), nud
```



In essence, the ridge penalty pushes muticolinear features together rather than allowing one to be wildy positive and one widly negative. Additionally, many of the less-imporant features also get pushed toward zero.

What ridge regression is **NOT**:

A feature selection technique. It will retain all features in the final model. Therefore, ridge regression is appropriate if you need to retain all the features, yet reduce the noise that less influential variables may create.

Lasso penalty

Lasso stands for **L**east **A**bsolute **S**hrinkage and **S**election **O**perator.

Lasso penalty: $\lambda \sum_{j=1}^{p} |\beta_j|$

Lasso Regresion Function: $\sum_{i=1}^{n} (y_i - \hat{y})^2 + \lambda \sum_{j=1}^{p} |\beta_j|$

As $\lambda \rightarrow$ inf, the lasso penalty will actually push feature coefficents to zero.

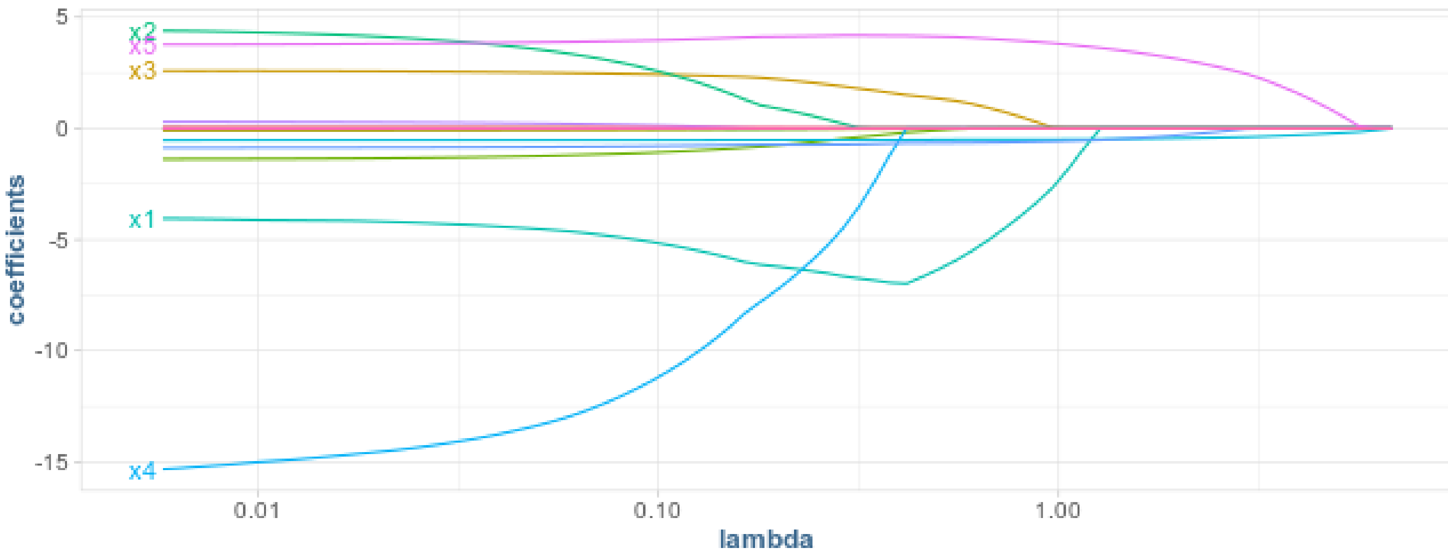
This model serves as a sort of automatic feature selection.

Visually:

```
# model
boston_lasso <- glmnet::glmnet(
   x = boston_train_x,
   y = boston_train_y,
   alpha = 1
)

lam <- boston_lasso$lambda %>%
   as.data.frame() %>%
   mutate(penalty = boston_lasso$a0 %>% names()) %>%
   rename(lambda = ".")

results <- boston_lasso$beta %>%
   as.matrix() %>%
   as.data.frame() %>%
   rownames_to_column() %>%
   gather(penalty, coefficients, -rowname) %>%
   left_join(lam)
```

```
Joining, by = "penalty"
```

```
result_labels <- results %>%
   group_by(rowname) %>%
   filter(lambda == min(lambda)) %>%
   ungroup() %>%
   top_n(5, wt = abs(coefficients)) %>%
   mutate(var = paste0("x", 1:5))

ggplot() +
   geom_line(data = results, aes(lambda, coefficients, group = rowname, color = rowname), show.
   scale_x_log10() +
   geom_text(data = result_labels, aes(lambda, coefficients, label = var, color = rowname), nud
```



We can see that as λ grows, the number of features retained decreases.

Lasso regression can be a good tool to extract the most consistent features.

Elastic nets

A generalization of the ridge and lasso penalties, called *elastic nets*, combines the two penalties.

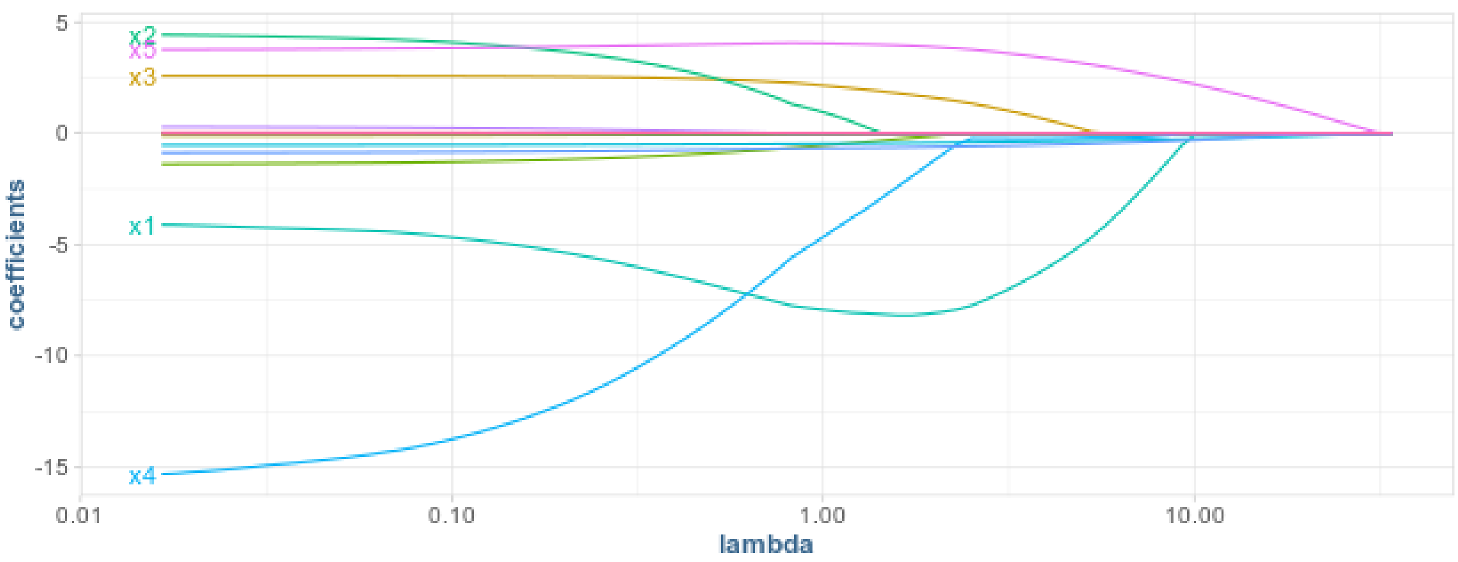
Elastic net function: min $\sum_{i=1}^{n}(y_i - \hat{y})^2 + \lambda \sum_{j=1}^{p} \beta_j^2 + \lambda \sum_{j=1}^{p} |\beta_j|$

Visually:

```
# model
boston_elastic <- glmnet::glmnet(
   x = boston_train_x,
   y = boston_train_y,
   alpha = .2
)

lam <- boston_elastic$lambda %>%
   as.data.frame() %>%
   mutate(penalty = boston_elastic$a0 %>% names()) %>%
   rename(lambda = ".")

results <- boston_elastic$beta %>%
   as.matrix() %>%
   as.data.frame() %>%
   rownames_to_column() %>%
   gather(penalty, coefficients, -rowname) %>%
   left_join(lam)
```

```
Joining, by = "penalty"
```

```
result_labels <- results %>%
  group_by(rowname) %>%
  filter(lambda == min(lambda)) %>%
  ungroup() %>%
  top_n(5, wt = abs(coefficients)) %>%
  mutate(var = paste0("x", 1:5))

ggplot() +
  geom_line(data = results, aes(lambda, coefficients, group = rowname, color = rowname), show.
  scale_x_log10() +
  geom_text(data = result_labels, aes(lambda, coefficients, label = var, color = rowname), nuc
```



Implementation

glmnet: *alpha* parameter = penalty weight

$\alpha = 0$, ridge

$\alpha = 1$, lasso

$0 < \alpha < 1$, elastic net

```
X <- model.matrix(Sale_Price ~., data = ames.train)[, -1]

Y <- log(ames.train$Sale_Price)
```

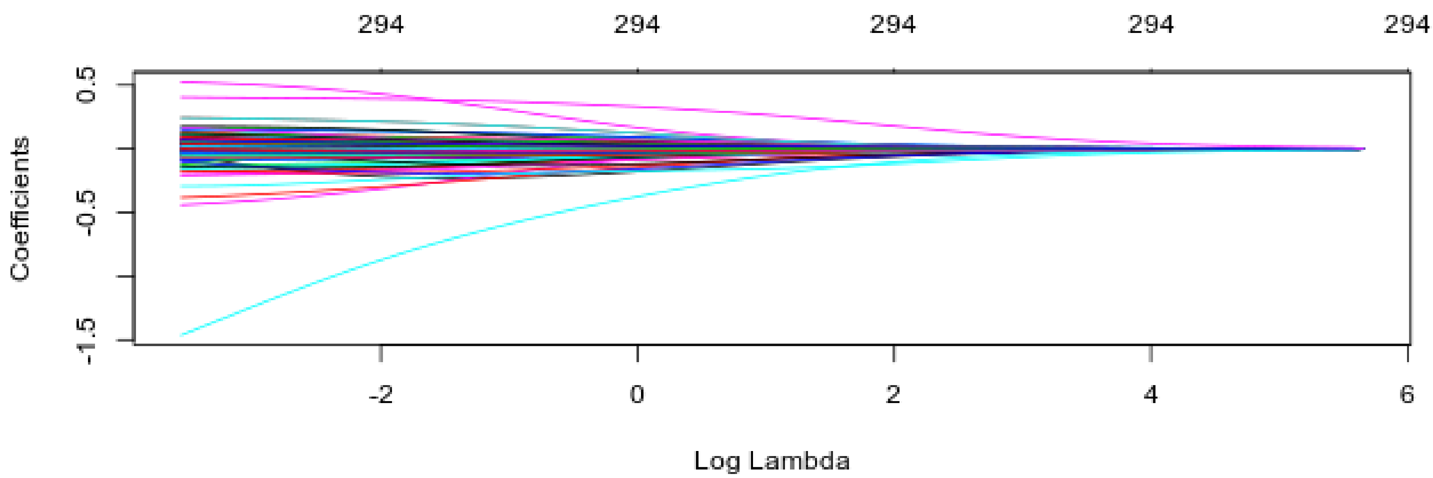
We need to ensure that all features are on a common scale (otherwise large magnitude features will have more weight).

(standardization is done automatically with *glmnet*)

```
# Apply ridge regression to ames data

ridge <- glmnet(
   x = X,
   y = Y,
   alpha = 0
)
```

glmnet automatically fits models across a wide range of λ values.

```
plot(ridge, xvar = "lambda")
```



coefficients:

```
params <- coef(ridge)

ridge$lambda %>% head()
```

```
[1] 285.8055 260.4153 237.2807 216.2014 196.9946 179.4942
```

```
coef(ridge)[c("Latitude", "Overall_QualVery_Excellent"), 100]
```

```
                 Latitude Overall_QualVery_Excellent
                0.4048216                  0.1423770
```

```
coef(ridge)[c("Latitude", "Overall_QualVery_Excellent"), 1]
```

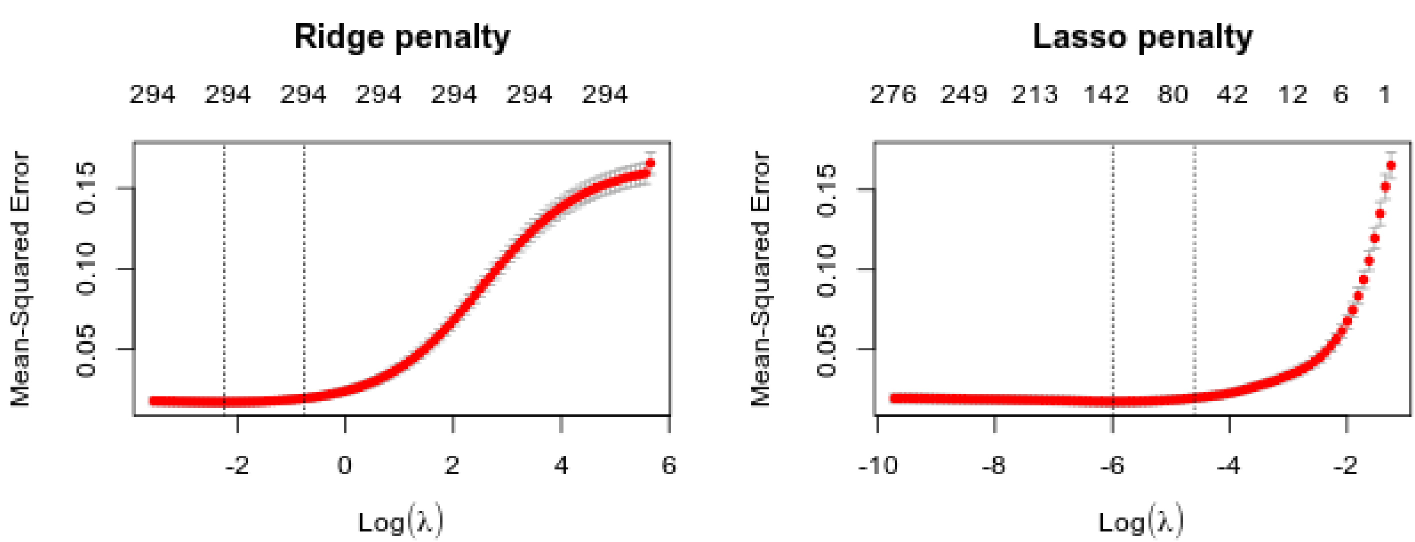
```
                                             Latitude
0.000000000000000000000000000000000000063823847
                           Overall_QualVery_Excellent
0.000000000000000000000000000000000000009838114
```

Tuning

To help find the optimal value of λ we use k-fold cross validation.

Note: by default glmnet::cv.glmnet uses MSE as the loss function, this can be changed by altering the 'type.measurement'

```r
# ?glmnet::cv.glmnet

ridge <- cv.glmnet(
   x = X,
   y = Y,
   alpha = 0
)

lasso <- cv.glmnet(
   x = X,
   y = Y,
   alpha = 1
)

# plot results

par(mfrow = c(1, 2))
plot(ridge, main = "Ridge penalty\n\n")
plot(lasso, main = "Lasso penalty\n\n")
```




```r
# Ridge model
min(ridge$cvm)
```

```
[1] 0.01748122
```

```
ridge$lambda.min # lambda for this min MSE
```

```
[1] 0.1051301
```

```
ridge$cvm[ridge$lambda == ridge$lambda.1se] # 1-SE rule
```

```
[1] 0.01975572
```

```
ridge$lambda.1se # lambda for this MSE
```

```
[1] 0.4657917
```

```
# Lasso model
min(lasso$cvm)
```

```
[1] 0.01754244
```

```
lasso$lambda.min
```

```
[1] 0.00248579
```

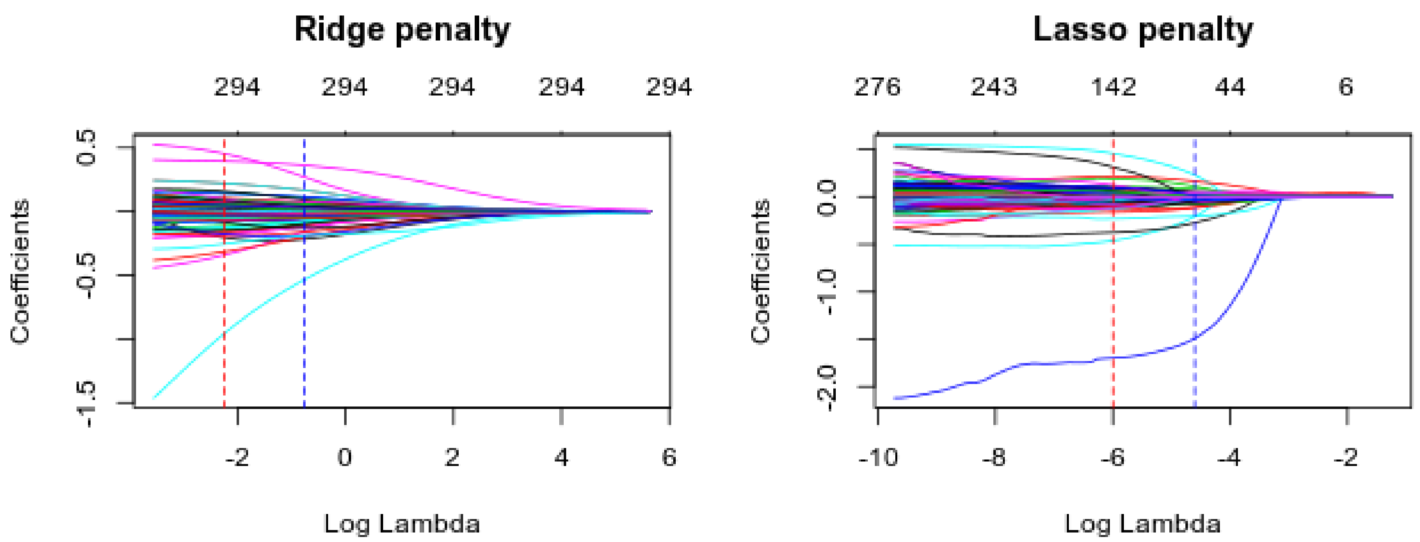
```
lasso$cvm[lasso$lambda == lasso$lambda.1se]
```

```
[1] 0.01979976
```

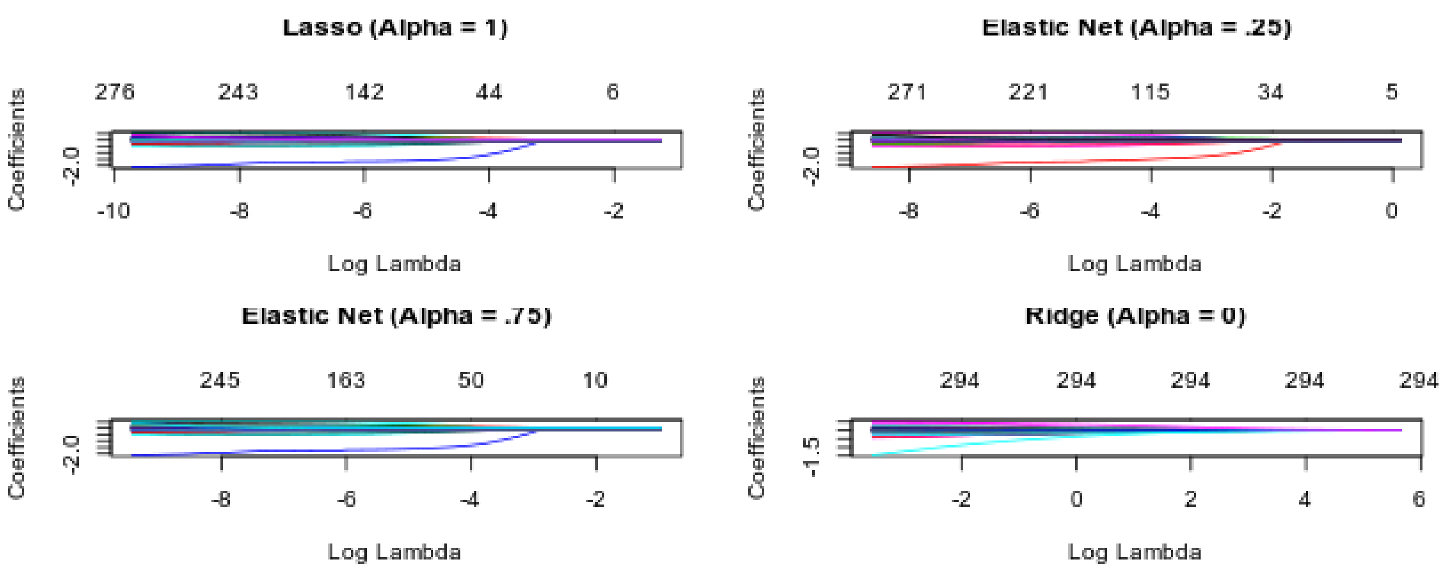
Feature Reduction

```
ridge_min <- glmnet(
   x = X,
   y = Y,
   alpha = 0
)

lasso_min <- glmnet(
   x = X,
   y = Y,
   alpha = 1
)

par(mfrow = c(1, 2))

# plot ridge model
plot(ridge_min, xvar = "lambda", main = "Ridge penalty\n\n")
abline(v = log(ridge$lambda.min), col = "red", lty = "dashed")
abline(v = log(ridge$lambda.1se), col = "blue", lty = "dashed")

# plot lasso model
plot(lasso_min, xvar = "lambda", main = "Lasso penalty\n\n")
abline(v = log(lasso$lambda.min), col = "red", lty = "dashed")
```

```r
abline(v = log(lasso$lambda.1se), col = "blue", lty = "dashed")
```




```r
lasso    <- glmnet(X, Y, alpha = 1.0)
elastic1 <- glmnet(X, Y, alpha = 0.25)
elastic2 <- glmnet(X, Y, alpha = 0.75)
ridge    <- glmnet(X, Y, alpha = 0.0)

par(mfrow = c(2, 2))
plot(lasso, xvar = "lambda", main = "Lasso (Alpha = 1)\n\n\n")
plot(elastic1, xvar = "lambda", main = "Elastic Net (Alpha = .25)\n\n\n")
plot(elastic2, xvar = "lambda", main = "Elastic Net (Alpha = .75)\n\n\n")
plot(ridge, xvar = "lambda", main = "Ridge (Alpha = 0)\n\n\n")
```



Grid Search for auto-tune

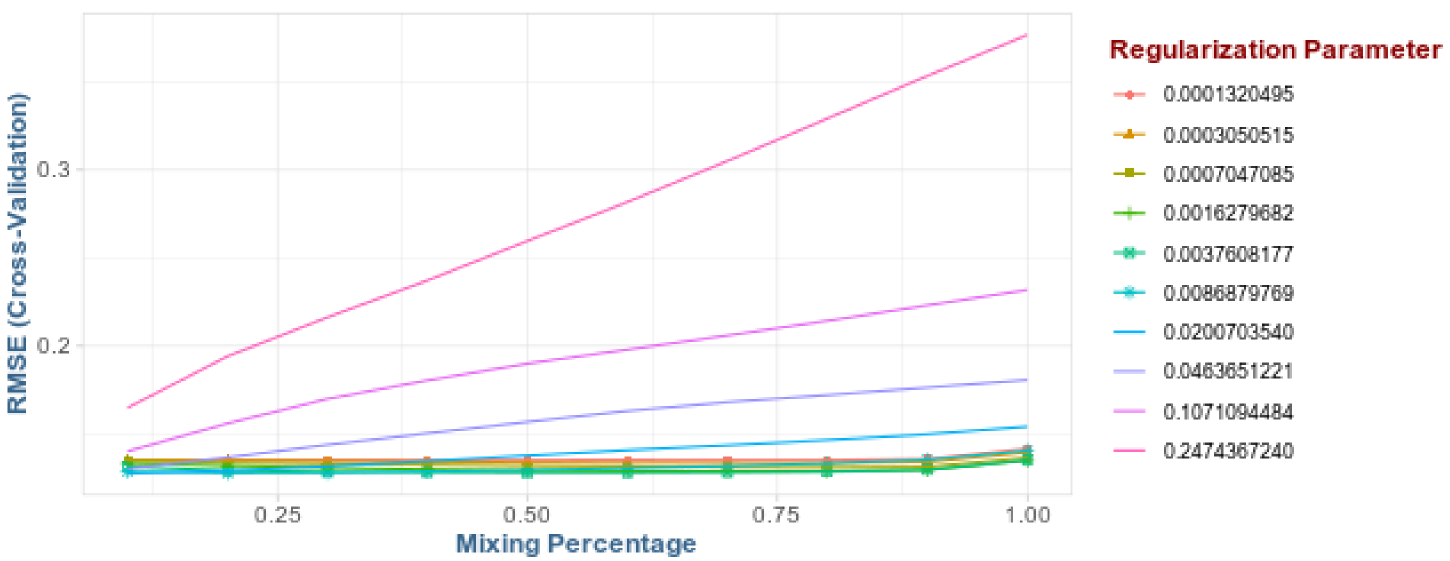
```r
set.seed(123)

cv_glmnet <- train(
   x = X,
   y = Y,
   method = "glmnet",
   preProc = c("zv", "center", "scale"),
   trControl = trainControl(method = "cv", number = 10),
   tuneLength = 10
)

# model with lowest RMSE
cv_glmnet$bestTune
```

```
  alpha     lambda
7   0.1 0.02007035
```

```r
ggplot(cv_glmnet)
```

```
Warning: The shape palette can deal with a maximum of 6 discrete values because
more than 6 becomes difficult to discriminate; you have 10. Consider
specifying shapes manually if you must have them.
```

```
Warning: Removed 40 rows containing missing values (geom_point).
```



Evaluate performance:

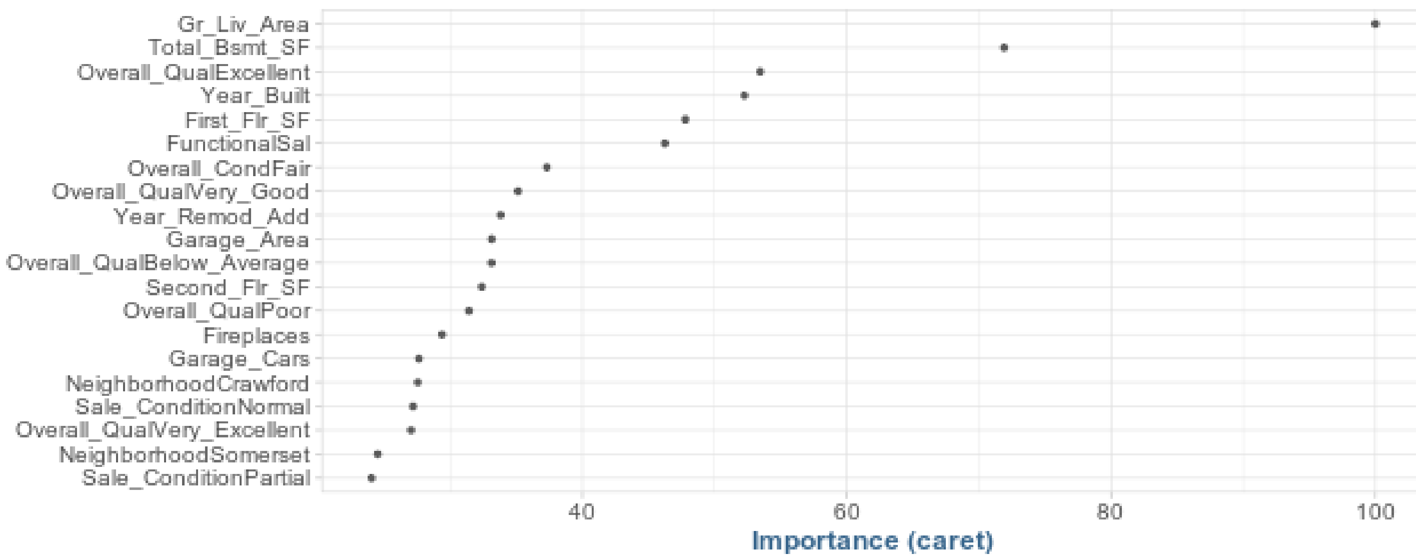
```r
pred <- predict(cv_glmnet, X)

# compute RMSE of transformed predictors
RMSE(exp(pred), exp(Y))
```

```
[1] 19905.05
```

Feature Interpretation

Feature interpretation is roughly the same as in PSL.

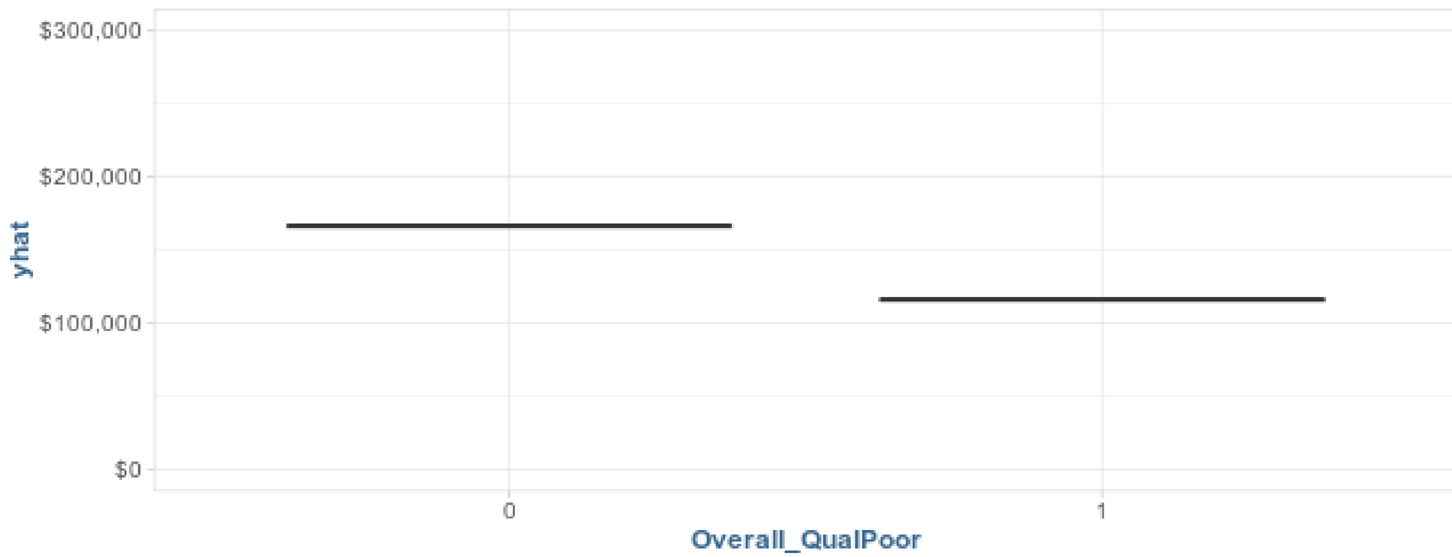
```r
vip(cv_glmnet, num_features = 20, bar = F)
```




```r
p1 <- pdp::partial(cv_glmnet, pred.var = "Gr_Liv_Area", grid.resolution = 20) %>%
  mutate(yhat = exp(yhat)) %>%
  ggplot(aes(Gr_Liv_Area, yhat)) +
  geom_line() +
  scale_y_continuous(limits = c(0, 300000), labels = scales::dollar)

p2 <- pdp::partial(cv_glmnet, pred.var = "Overall_QualExcellent") %>%
  mutate(
    yhat = exp(yhat),
    Overall_QualExcellent = factor(Overall_QualExcellent)
    ) %>%
  ggplot(aes(Overall_QualExcellent, yhat)) +
  geom_boxplot() +
  scale_y_continuous(limits = c(0, 300000), labels = scales::dollar)

p3 <- pdp::partial(cv_glmnet, pred.var = "First_Flr_SF", grid.resolution = 20) %>%
  mutate(yhat = exp(yhat)) %>%
  ggplot(aes(First_Flr_SF, yhat)) +
  geom_line() +
  scale_y_continuous(limits = c(0, 300000), labels = scales::dollar)
```

```
p4 <- pdp::partial(cv_glmnet, pred.var = "Garage_Cars") %>%
  mutate(yhat = exp(yhat)) %>%
  ggplot(aes(Garage_Cars, yhat)) +
  geom_line() +
  scale_y_continuous(limits = c(0, 300000), labels = scales::dollar)

grid.arrange(p1, p2, p3, p4, nrow = 2)
```


```
pdp::partial(cv_glmnet, pred.var = "Overall_QualPoor") %>%
  mutate(
    yhat = exp(yhat),
    Overall_QualPoor = factor(Overall_QualPoor)
    ) %>%
  ggplot(aes(Overall_QualPoor, yhat)) +
  geom_boxplot() +
  scale_y_continuous(limits = c(0, 300000), labels = scales::dollar)
```



Attrition

```
set.seed(123)

churn_split <- initial_split(attrition, prop = .7, strata = "Attrition")
churn.train <- training(churn_split)
churn.test <- testing(churn_split)

glm_mod <- train(
   Attrition ~.,
   data = churn.train,
   method = "glm",
   family = "binomial",
   preProc = c("zv", "center", "scale"),
   trControl = trainControl(method = "cv", number = 10)
)

# train regularized logistic regression model

set.seed(123)

penalized_mod <- train(
   Attrition ~.,
   data = churn.train,
   method = "glmnet",
   family = "binomial",
   preProc = c("zv", "center", "scale"),
```

```
  trControl = trainControl(method = "cv", number = 10),
  tuneLength = 10
)

summary(resamples(list(
  logistic_model = glm_mod,
  penalized_model = penalized_mod
)))$statistics$Accuracy
```

```
                     Min.   1st Qu.    Median      Mean   3rd Qu.      Max.
logistic_model  0.8235294 0.8665049 0.8737864 0.8717854 0.8819081 0.9126214
penalized_model 0.8446602 0.8759280 0.8834951 0.8835759 0.8915469 0.9411765
                NA's
logistic_model     0
penalized_model    0
```

```
# clean up
rm(list = ls())
```