Hands-on Machine Learning with R

Contents

6 Regularized Regression

The problem with linear models and the sort of datasets we are dealing with today is that the assumptions quickly break down and the models overfit the data and our *out of sample error* increases. So the models appear to work well on the training data, but when you apply it out of sample i.e. on unseen data, it performs much worse.

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

6.1 Prerequisites

```
# load ames housing data
ames <- AmesHousing::make_ames()

# Stratified sampling with the rsample package
set.seed(123)
split <- initial_split(ames, prop = 0.7,
                       strata = "Sale_Price")
ames_train  <- training(split)
ames_test   <- testing(split)
# Create training (70%) and test (30%) sets for the
# rsample::attrition data.
df <- attrition %>% mutate_if(is.ordered, factor, ordered = FALSE)
set.seed(123)  # for reproducibility
churn_split <- initial_split(df, prop = .7, strata = "Attrition")
churn_train <- training(churn_split)
churn_test  <- testing(churn_split)
```

6.2 Why Regularize

So for example in OLS your objective is to minimise the sum of squared errors between the observed and predicted values. !!! you're not minimising the gray lines as the text says, but their squares!

So we're minimizing the following function:

$$SSE = \sum_{n}^{i=1}(y_i - y_i)^2$$

This objective function is fine if our data has a linear relatioship, if there are more observations than there are features (n»p) and there is no multi-colinearity. (Additionally if you want to do classical tests of inference you also have to assume normality and homoscedasticity of errors).

!!! misleading wording "As p increases, we're more likely to violate some of the OLS assumptions and alternative approaches should be considered." does not make clear that if $p > n$ there is no solution to OLS anymore. Ah, ok, you explain it later on.

Anyway, so the *betting on sparisty* principle is invoked here. This says that in high dimensional situations it is reasonalble to assume that most effects are not significant. You can justify this because it is often true, and because if it wasn't, you wouldn't know what to do anyway. So basically you want a type of **feature selection**, to reduce p, the number of features.

One type of feature selection is *hard thresholding*, this is the kind of elimination that is used in forward selection and backward elimination. But here a feature either is or isn't in the model. An alternative more modern approach is *soft thresholding* where effects are slowly pushed out, and sometimes reduced to zero also, but not necessarily. And apparently this can be more accurate as well as easier to interpret.

Another term for regularized regression is also *penalized* models or *shrinkage* methods as a way to constrain the total size of the coefficient estimates. Which apparently helps reduce the magnitude and the fuluctuations of our coefficient estimates and will reduce the vairance of the model - at the expense of no longer being unbiased. (!?). SO now we are also minimizing a penalty term:

$$min(SSE + P)$$

The way this P works is that the only way the coefficients can increase is if there is a concomitant reduction in SSE. SO thi is for OLS where SSE is the loss function. But this generalizes to other GLM models as well, that just have different loss functions. The three common penalty parameters are:

1. Ridge,
2. Lasso,
3. Elastic net or ENET, which is a combination of Ridge and Lasso.

6.2.1 Ridge regression

The ridge parameter is added to the loss function:

$$min(SSE + \lambda \sum_{j=1}^{p} \beta_j^2)$$

The penalty is also known as L^2, or the *Euclidean norm*. This is the sum of all the squared coefficients multiplied by the controlling or tuning parameter λ. If λ is zero, then this is OLS, since the penalty has no effect. But as $\lambda -> \infty$, the coefficients get forced towards zero - but not all the way.

There's an example in the book of how the individual ridge regression coefficients change as you increase λ. They all tend towards zero over time, but not necessarily monotonically. Fluctuations of individual coefficients likely indicates multicolinearity. SO you want to restrict λ to a range where this doesn't happen, which should decrease variance and therefore the error of our prediciton.

So what it's doing is pushing correlated features towards eachother, and less important features toward zero. This also crystalises the focus on the important features. But it does keep all the features. So it's good if you have reason to believe you need to retain all the variables but just reduce the noise. Like in a small dataset with a lot of multicolineariety. Otherwise if some features might be redundant then lasso is preferable or ENET.

6.2.2 Lasso–least absolute shrinkage and selection operator

The lasso penalty instead of L^2 you have the L^1 norm. This means:

$$min(SSE + \lambda \sum_{j=1}^{p} |\beta_j|)$$

The lasso pentalty pushes coefficeints to zero. So this not only improves the model but also performs autmated feature selection.

6.2.3 Elastic nets

ENets are a generalization of ridge and lasso regression, which combines the two options:

$$min(SSE + \lambda_1 \sum_{j=1}^{p} \beta_j^2 + \lambda_2 \sum_{j=1}^{p} |\beta_j|)$$

So this apparently is the best of both worlds: ridge penalty performs effective regularizatin and lasso is better at feature selection.

6.3 Implementation

We will use the direct engine `glmnet` but there are other options as well. It's very fast, but doesn't take XY formula inputs, so you have to set up the matrices instead. So we use `model.matrix` to create the feature matrices. And remove the intercept with dropping the first column.

```
# Create training  feature matrices
# we use model.matrix(...)[, -1] to discard the intercept
X <- model.matrix(Sale_Price ~ ., ames_train)[, -1]

# transform y with log transformation
Y <- log(ames_train$Sale_Price)
```

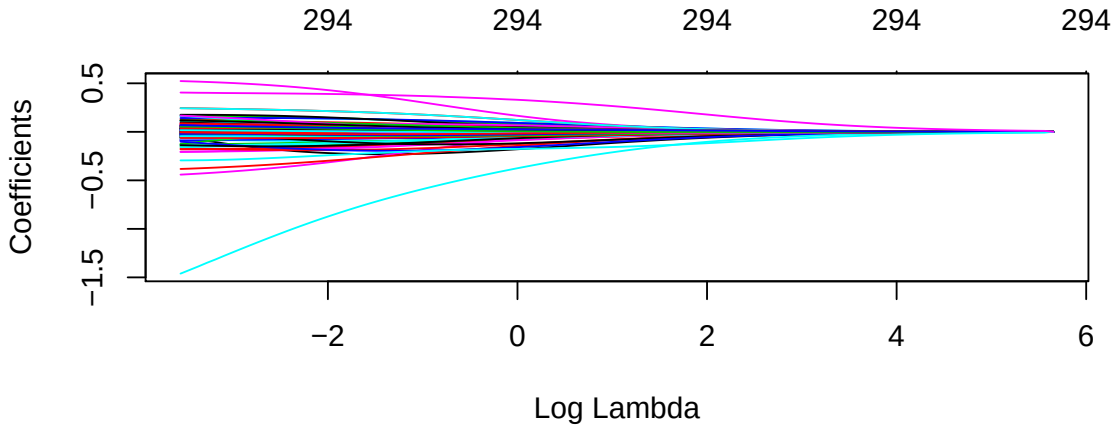


Figure 1: Ridge regression

!!! Again this plot looks different to the one in the book.. Maybe it's just sth to do with this ames data. I mean for sure. But not sure what. I mean i also have 294 features and they have 301. How did this happen?

Anyway, ridge automatically picked a reasonable range of 100 lambdas from the data, so usually no need to adjust this. It also saves all the coefficients of all the models. So we can check e.g. the largest lanbda and the smallest to see what the coefficients are like.

```
# largest labdas
ridge$lambda %>% head()
```

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

```
# smallest lambdas
ridge$lambda %>% tail()
```

```
## [1] 0.04550831 0.04146548 0.03778180 0.03442537 0.03136712 0.02858055
```

```
# two of the coefficients
tail(sort(coef(ridge)[, 100]))
```

```
##   Overall_QualExcellent          Roof_StyleShed     Exterior_2ndPreCast
##               0.1675813               0.1758649               0.2432053
##     Exterior_1stPreCast                Latitude NeighborhoodGreen_Hills
##               0.2442353               0.4048216               0.5230376
```

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

```
##                   Latitude Overall_QualVery_Excellent
##                  0.4048216                  0.1423770
```

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

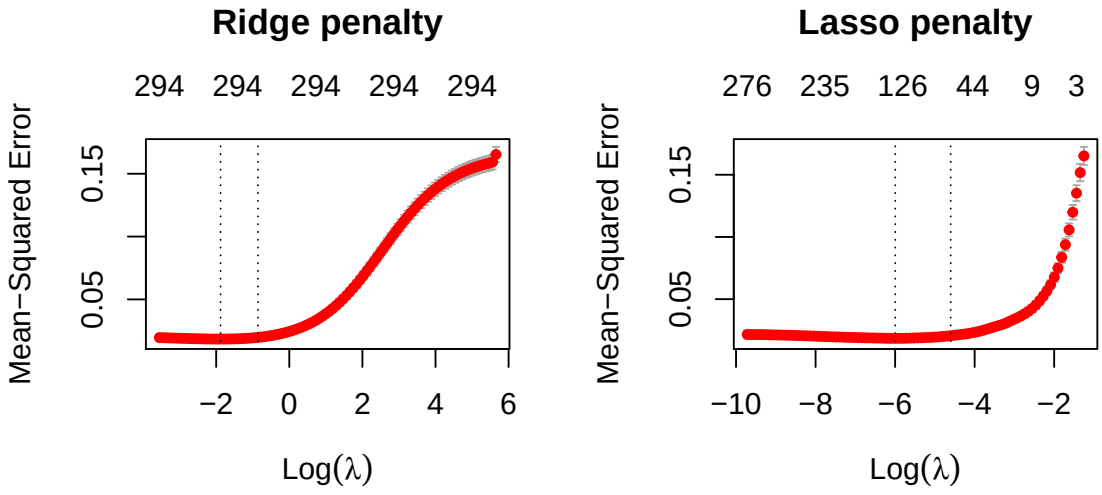
```
##                   Latitude Overall_QualVery_Excellent
##                6.382385e-36               9.838114e-37
```

!!! Divfferent values of coefficients, also top two coefficients are Latitude and ElectricalMix in my resutls

But so here we have not actually validated anything or measured how much improvement we are getting by increasing lambda.

6.3.1 Tuning

So λ is a tuning parameter, one that stops our model from overfitting. In order to find the optimal one we can perform k-fold cross validation. The function `cv.glmnet()` performs CV, by default 10-fold.



The default loss function in `cv.glmnet` is mean square error, but you can change that if you like by changing `type.measure` to e.g. mean absolute error.

Looking at the figures we can see that in both models the error falls as the penalty $log(\lambda)$ gets larger, which suggests that the normal OLS was overfitting the training data. But as we increase

the penalty further - constrain the coefficents more, the error starts to increase (!!!not decrease like it says in the book).

As the penalty increases the lasso model removes features (you can see the number at the top of the plot). But with the ridge model you keep all of them. The **first vertical line** is the λ with the lowest MSE. The second one is the largest λ value within one standard error of the minimum, which is a more restricted model to prevent overfitting even more. [Is this just a rule of thumb?]

In the `cv.glmnet` object, `cvm` is the mean cross-validated error - one for each lambda. So the minimum for our ridge model was 0.0184027, and the lambda value that gives the minimum error was 0.1525258. The largest lambda within one standard error of this one is '`ridge$lambda.1se`, and the error there is 0.0197359, You can see the lasso model values below as well.

```
# Ridge model
min(ridge$cvm)       # minimum MSE
```

```
## [1] 0.01840266
```

```
## [1] 0.01899152
ridge$lambda.min     # lambda for this min MSE
```

```
## [1] 0.1525258
```

```
## [1] 0.09686166

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

```
## [1] 0.01973589
```

```
## [1] 0.02120642
ridge$lambda.1se  # lambda for this MSE
```

```
## [1] 0.4244121
```

```
## [1] 0.4709997

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

```
## [1] 0.01849366
```

```
## [1] 0.02090188
lasso$lambda.min     # lambda for this min MSE
```

```
## [1] 0.00248579
```

```
## [1] 0.004002336

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

```
## [1] 0.02062315
```

```
## [1] 0.02410281
lasso$lambda.1se  # lambda for this MSE
```

```
## [1] 0.01003518
```

```
## [1] 0.0161575
```

So we can visualise where these lambdas are on an individual example. The red line here is the lambda where teh error is minimised and the blue one is within one standard error of it, showing how much we can constrain the coefficients while *still maximising predictive accuracy* !!! not sure what this means, why are we still maximising it? Aah, here: "The main point of the 1 SE rule, with which we agree, is to choose the simplest model whose accuracy is comparable with

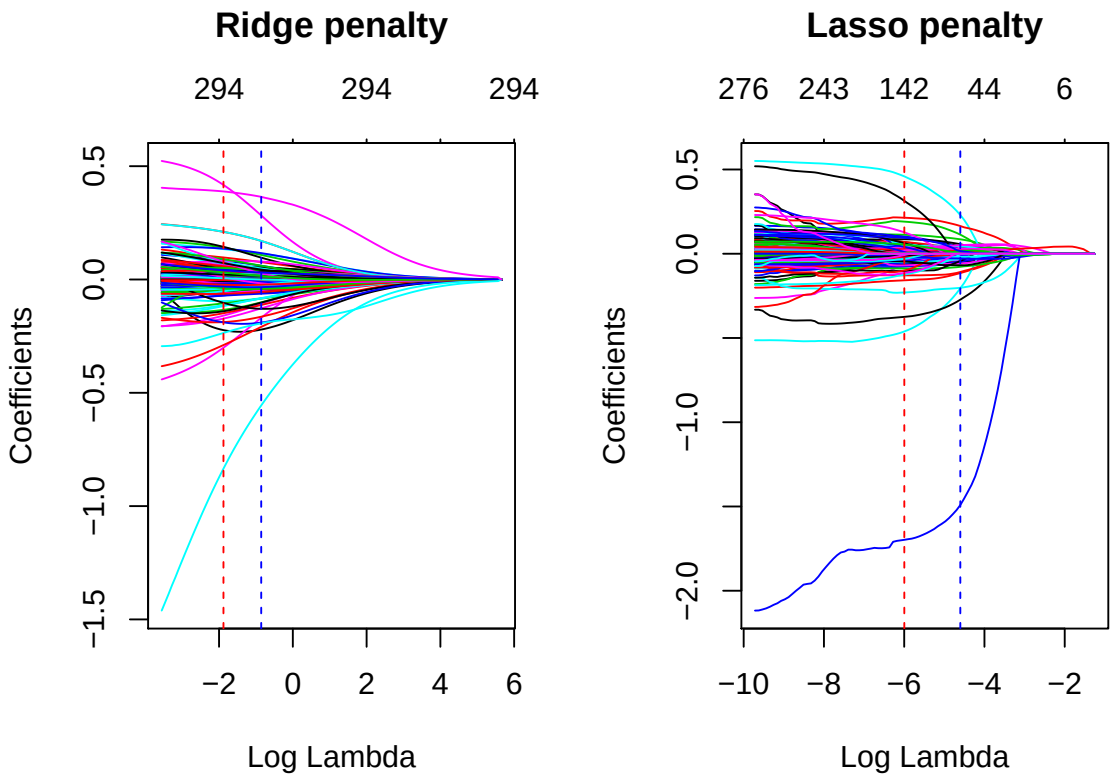
the best model." The se is the vertical one around each error. So teh idea is that the model with the lowest cross-validation error generally overfits a bit (still!?). The 1SE rule is a heuristic, for sure. So the error is 1 SE above the error of the best model, while gaining parsimony.

```r
# Ridge model
ridge_min <- glmnet(
  x = X,
  y = Y,
  alpha = 0
)

# Lasso model
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$lambda.min), col = "red", lty = "dashed")
abline(v = log(lasso$lambda.1se), col = "blue", lty = "dashed")
```



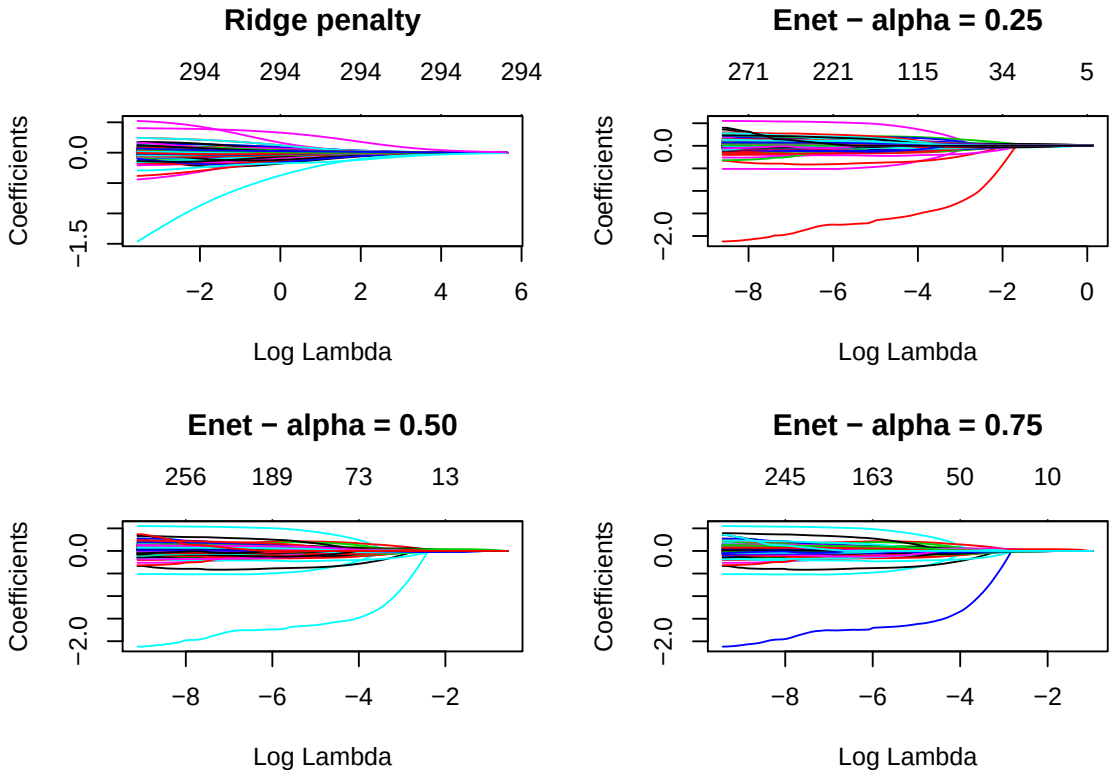
Alpha is the parameter that switches between ridge and lasso, 0.5 is equally weighted both, otherwise one penalty is more importand than the other:

```r
# Ridge model
enet_25 <- glmnet(
  x = X,
  y = Y,
  alpha = 0.25
)

# Lasso model
enet_50 <- glmnet(
  x = X,
  y = Y,
  alpha = 0.50
)

# Lasso model
enet_75 <- glmnet(
  x = X,
  y = Y,
  alpha = 0.75
)

par(mfrow = c(2, 2))
# plot ridge model
plot(ridge_min, xvar = "lambda", main = "Ridge penalty\n\n")
plot(enet_25, xvar = "lambda", main = "Enet - alpha = 0.25\n\n")
plot(enet_50, xvar = "lambda", main = "Enet - alpha = 0.50\n\n")
plot(enet_75, xvar = "lambda", main = "Enet - alpha = 0.75\n\n")
```



So in addition to lanbda you also want to tune alpha. We can do this with the `caret` package.

So this is searching the grid of ten different alphas and ten different lambdas. The best one with a alpha value of 0.1 and lambda of 0.02007035 has an error of

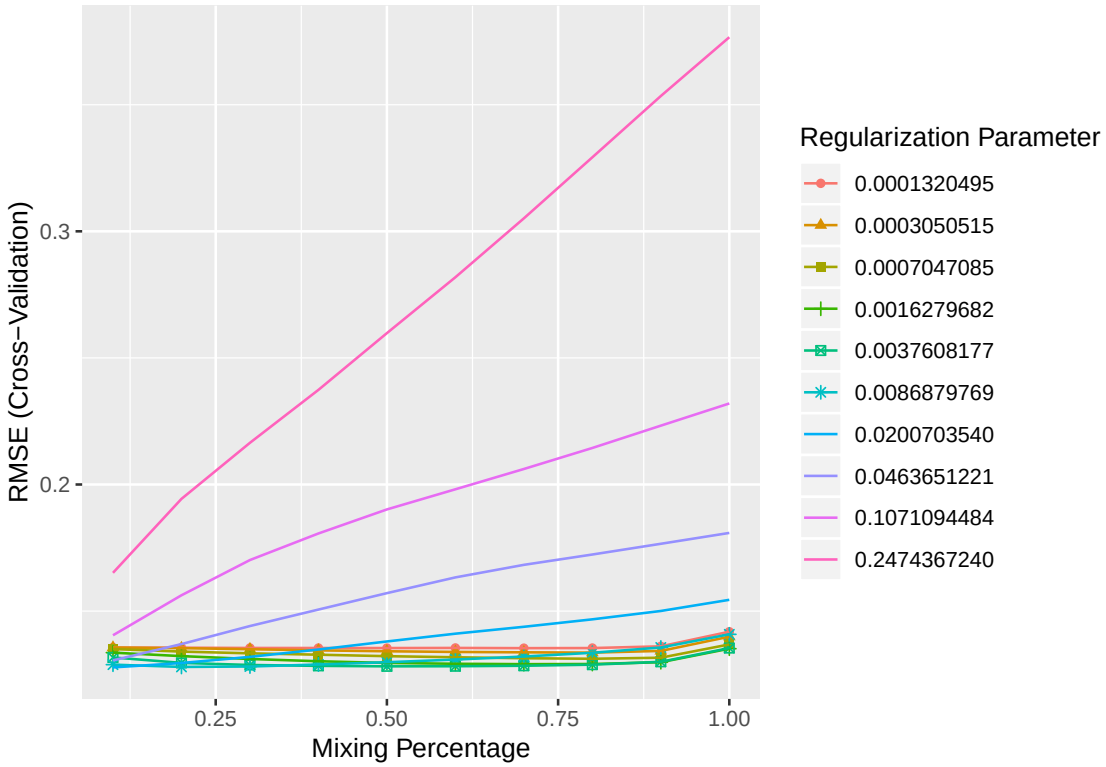
```r
set.seed(123)

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

cv_glmnet$bestTune
```

```
##   alpha     lambda
## 7   0.1 0.02007035
```

```r
# plot cross-validated RMSE
ggplot(cv_glmnet)
```



So how does this compare to the error from the previous chapter? Careful, because we logged the sales price here, we need to transform it back. So I think my RMSE error was $25,459.51 with the partial least squares OLS, and now it is $19,905. !!! These numbers are again different from the ones in the book.

```r
# predict sales price on training data
pred <- predict(cv_glmnet, X)

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

```
## [1] 19905.05
```

6.4 Feature Interpretation

This works the same was as in linear or logistic regression, the importance is determined by the relative size of standardized coefficients. At least the first two seem to be the same than the PLS model.

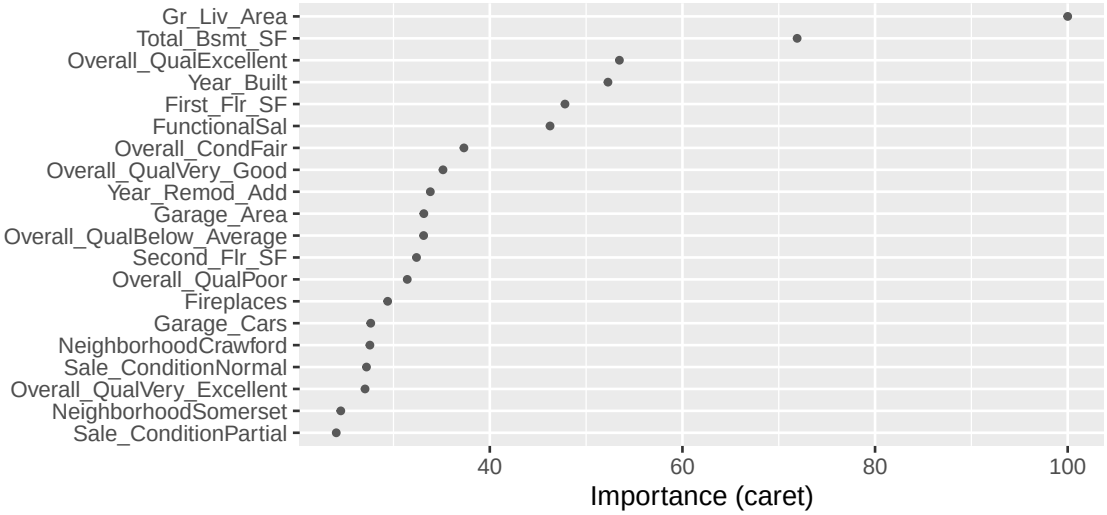
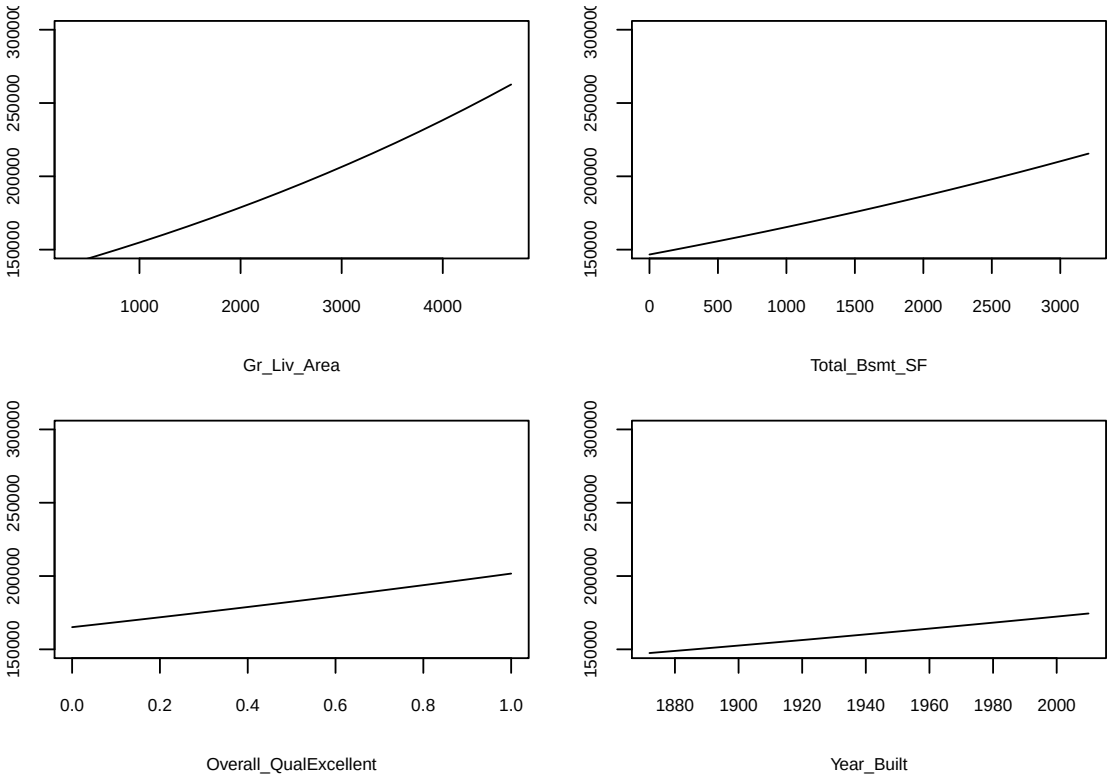


Figure 2: Most important features

Again we can look at the partial dependence plots like with the linear regression. The response increases linearly and monotonically with the features, but because the response was logged it relationship is non-linear in the original scale.

```
par(mfrow = c(2,2))
par(mar = c(5,3,.5, .5))
par(cex.axis=0.7, cex.lab=0.7, cex.main = 1)

pdp::partial(cv_glmnet, pull(top$data[1,1]), grid.resolution = 20) -> x1
plot(x1[[1]], exp(x1[[2]]), type = "l",  ylim = c(150000, 300000),
     xlab = pull(top$data[1,1]))
pdp::partial(cv_glmnet, pull(top$data[2,1]), grid.resolution = 20) -> x2
plot(x2[[1]], exp(x2[[2]]), type = "l",  ylim = c(150000, 300000),
       xlab = pull(top$data[2,1]))
pdp::partial(cv_glmnet, pull(top$data[3,1]), grid.resolution = 20) -> x3
plot(x3[[1]], exp(x3[[2]]), type = "l",  ylim = c(150000, 300000),
       xlab = pull(top$data[3,1]))
pdp::partial(cv_glmnet, pull(top$data[4,1]), grid.resolution = 20) -> x4
plot(x4[[1]], exp(x4[[2]]), type = "l",  ylim = c(150000, 300000),
       xlab = pull(top$data[4,1]))
```



!!! not sure how to get a pdp for a dummy variable. the 3rd one here is, but it's drawn like a linear one

6.5 Attrition data

```
df <- attrition %>% mutate_if(is.ordered, factor, ordered = FALSE)

# Create training (70%) and test (30%) sets for the
# rsample::attrition data. Use set.seed for reproducibility
set.seed(123)
churn_split <- initial_split(df, prop = .7, strata = "Attrition")
train <- training(churn_split)
test  <- testing(churn_split)

# train logistic regression model
set.seed(123)
glm_mod <- train(
  Attrition ~ .,
  data = 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 = train,
  method = "glmnet",
```

```
  family = "binomial",
  preProc = c("zv", "center", "scale"),
  trControl = trainControl(method = "cv", number = 10),
  tuneLength = 10
  )

# extract out of sample performance measures
summary(resamples(list(
  logistic_model = glm_mod,
  penalized_model = penalized_mod
  )))$statistics$Accuracy
```

```
##                      Min.   1st Qu.    Median      Mean   3rd Qu.
## logistic_model  0.8365385 0.8495146 0.8792476 0.8757893 0.8907767
## penalized_model 0.8446602 0.8759280 0.8834951 0.8835759 0.8915469
##                      Max. NA's
## logistic_model  0.9313725    0
## penalized_model 0.9411765    0
```

```
##                      Min.   1st Qu.    Median      Mean   3rd Qu.
## logistic_model  0.8365385 0.8495146 0.8792476 0.8757893 0.8907767
## penalized_model 0.8446602 0.8759280 0.8834951 0.8835759 0.8915469
##                      Max. NA's
## logistic_model  0.9313725    0
## penalized_model 0.9411765    0
```

The penalized model is only slightly better than the normal logistic one. !!! In chapter 5 the best model accuracy was 87.58%, not as you say in chapter 6 86.3%.

6.6 Final thoughts

Regularized reg is a great solution especially to the $p > n$ problem, especially great for large datasets with lots of features, way better than OLS in those cases. It can minimise colinearity problems and help perform automatic feature selection. Also is computationally efficient, only two tuning parameters, so easy to tune and is memory efficient.

Issues:

- all the variables need to be numeric, so requires preprocessing
- cannot handle missing data, so you need to impute or remove
- not very robust to outliers in feature or response (just like GLM)
- assume a monotonic linear relationship
- decision on including interaction effects?