Hands-on Machine Learning with R

Contents

4 Linear Regression

Linear regression is one of the simplest algs for supervised learning. But it's a good starting point and many more complex methods can be seen as extensions of it.

4.1 Prerequisites

Adding `vip` packages for interpretability of variable importance. Ames data set from before.

4.2 Simple Linear Regresison

SLR assumes the relationship between two continuous variables is at least approximately linear.

$$Y_i = \beta_0 + \beta_1 X_i + \epsilon_i, \qquad for \quad i = 1, 2, ..., n,$$

Where Y_i represents the response/target variable, X_i is the i^{th} feature value and teh betas are fixed but unknown constants (coefficients or parameters), representing the intercept and the slope.

The ϵ_i term represents noise or random error. Here we assume the errors have a mean of zero and constant variance σ^2. This is denoted as $\overset{iid}{\sim} N(0, \sigma^2)$. Since the errors are centered on zero - the expected value $E(\epsilon_i) = 0$, linear regression is really a problem of estimating the conditional mean:

$$E(Y_i|X_i) = \beta_0 + \beta_1 X_i$$

WHich we can shorten to just $E(Y)$. So the interpretation is in therms of *average responses*. E.g. β_0 is the average response value when $X = 0$ - sometimes referred to as the *bias term* and β_1 is the increase in the average response if X increases by one unit, aka the *rate of change*.

4.2.1 Estimation

We want the best fitting line, but what is the best fit? The most common way, called *Oridnary least squares* (OLS) is to minimise the *residual sum of squares*:

$$RSS(\beta_0, \beta_1) = \sum_{i=1}^{n} [Y_i - (\beta_0 + \beta_1 X_i)]^2 = \sum_{i=1}^{n} (Y_i - \beta_0 - \beta_1 X_i)^2.$$

We denote the OLS stimates of the coefficients as $\hat{\beta}_0$ and $\hat{\beta}_1$. Once we have the estimated regression equation, we can predict values of Y for X_{new}:

$$\hat{Y}_{new} = \hat{\beta}_0 + \hat{\beta}_1 X_{new}$$

Where $\hat{Y}_{new}$ is the estimated mean response at $X = X_{new}$.

So let's try modelling the ames data relationship between the sale price and the above ground living area. This link)has good info on visualising residuals.

```
## # A tibble: 6 x 3
##   Sale_Price predicted residuals
##        <int>     <dbl>     <dbl>
## 1     215000   198968.    16032.
## 2     105000   111662.    -6662.
## 3     172000   161403.    10597.
## 4     195500   192994.     2506.
## 5     213500   162437.    51063.
## 6     236500   194373.    42127.
```

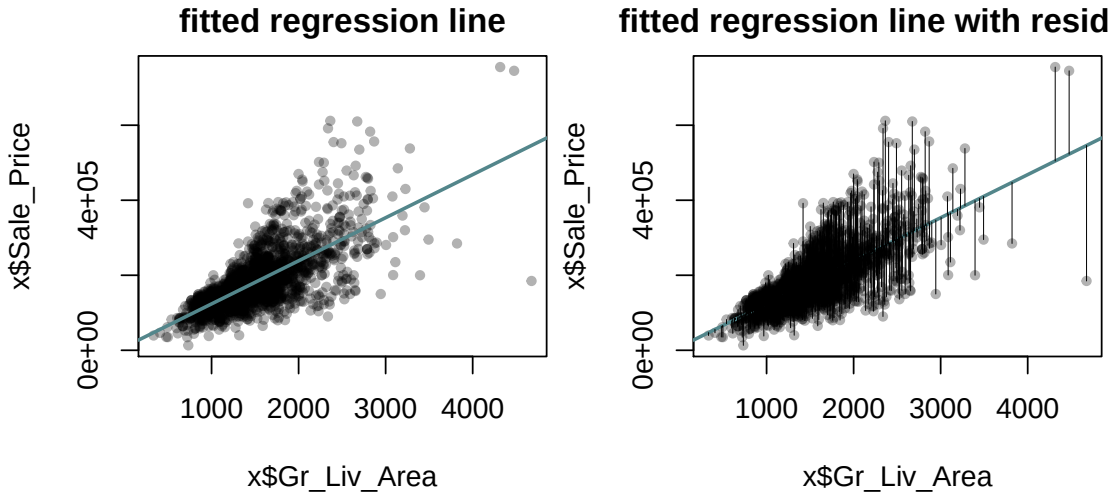


Figure 1: Regression residuals

Use `coef()` and `summary()` to have a look a the coefficients. !!! I don't get the same data, even though i have the same seed in the split?

```
coef(model1)
```

```
## (Intercept) Gr_Liv_Area
##    8732.938     114.876
```

```
summary(model1)
```

```
##
## Call:
## lm(formula = Sale_Price ~ Gr_Liv_Area, data = ames_train)
##
## Residuals:
##     Min      1Q  Median      3Q     Max
## -361143  -30668   -2449   22838  331357
##
## Coefficients:
##             Estimate Std. Error t value Pr(>|t|)
## (Intercept) 8732.938   3996.613   2.185    0.029 *
## Gr_Liv_Area  114.876      2.531  45.385   <2e-16 ***
```

```
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 56700 on 2051 degrees of freedom
## Multiple R-squared:  0.5011, Adjusted R-squared:  0.5008
## F-statistic:  2060 on 1 and 2051 DF,  p-value: < 2.2e-16
```

```
# glimpse(model1)
```

So we estimate that an increase in area by one square foot increases the selling price by 114.88$. SO nice and intuitive.

One drawback of using least squares is that we only have estimates of the coefficients, but not of the error variance σ^2. LS makes no assumptions about the random errors, so we cannot estimate σ^2.

An alternative is to use *maximum likelihood* estimation (ML) to estimate σ^2 – which we need to characterise the variability of our model. For ML we have to assume a particular distribution of the errors, most commonly that they are normally distributed. Under these assumptions the estimate of the error variance is

$$\hat{\sigma}^2 = \frac{1}{n-p}\sum_{i=1}^{n}(Y_i - \hat{Y}_i)^2 = \frac{1}{n-p}\sum_{i=1}^{n} r_i^2$$

Where r_i is the residual of the i^{th} observation. and p is the number of parameters or coefficients in the model. $\hat{\sigma}^2$ is also known as the mean squared error (MSE) and it's square root is the RMSE, and you can get it out of an `lm` object using `sigma()`

```
sigma(model1)
```

```
## [1] 56704.78
```

```
sigma(model1)^2
```

```
## [1] 3215432370
```

!!! the sigma is slightly different from the RMSE reported in the summary, not sure why, same in book.

4.2.2 Inference

The coefficients are only point estimates, so that's not super useful without a measure of variability. This is usually measured with a *standard error* (SE), the square root of it's variance. If we assume the errors are distributed $\overset{iid}{\sim} N(0, \sigma^2)$,then the SEs for the coefficients are simple and are expressed under the `Std. Error` heading in the summary for the model.

From the SE we can also do a t-test to see if the coefficients are statistically significantly different from zero. (!!! stantistically significant from zero is probably wrong).

The t-statistic is simply the estimated coefficient divided by the SE, which measures the number of standard deviations each coefficient is away from zero. The p-values are reported in the same table.

Under these same assumptions we can also derive the confidence intervals for the β coefficients. The formual is:

$$\beta_j \pm t_{1-\alpha/2, n-p} \hat{SE}(\hat{\beta}_j)$$

In R you can construct them using `confitn()`

```r
confint(model1, level = 0.95)
```

```
##                 2.5 %     97.5 %
## (Intercept) 895.0961 16570.7805
## Gr_Liv_Area 109.9121   119.8399
```

```r
# or if you wanna spell it out:
coef(model1)[2] - qt(0.975, model1$df.residual)*coef(summary(model1))[2,2]
```

```
## Gr_Liv_Area
##    109.9121
```

```r
coef(model1)[2] + qt(0.975, model1$df.residual)*coef(summary(model1))[2,2]
```

```
## Gr_Liv_Area
##    119.8399
```

So with 95% confidence we estimate that the mean sale price goes up between 109 and 119$ for each additional square foot.

Don't forget that these SEs and t-stats etc in the summary are based on the following assumptions:

1. Independent observations
2. The random errors have mean zero, and constant variance
3. The random errors are normally distributed

If your data deviate from these assumptions, there are some remedial actions you can take..

4.3 Multiple linear regression

Extend the simple linear regression with more predictors to see e.g. how are and year built are (linearly) related to the sales price using *mulitple linear regression* (MLR).

$$Y_i = \beta_0 + \beta_1 X_i + \beta_2 X_2 + \epsilon_i, \qquad for \quad i = 1, 2, ..., n,$$

Which you do in R by using + to separate predictors:

```r
(model2 <- lm(Sale_Price ~ Gr_Liv_Area + Year_Built, data = ames_train))
```

```
##
## Call:
## lm(formula = Sale_Price ~ Gr_Liv_Area + Year_Built, data = ames_train)
##
## Coefficients:
## (Intercept)  Gr_Liv_Area   Year_Built
##  -2.123e+06    9.918e+01    1.093e+03
```

```r
# or use update
(model2 <- update(model1, .~. + Year_Built))
```

```
##
## Call:
## lm(formula = Sale_Price ~ Gr_Liv_Area + Year_Built, data = ames_train)
##
## Coefficients:
## (Intercept)  Gr_Liv_Area   Year_Built
##  -2.123e+06    9.918e+01    1.093e+03
```

So holding the year constant, each additional square foot of living area increses the mean selling price by 99$. And holding the area constant, each additional year the home is newer by increases the mean price by 1093$. Here are some contour plots:

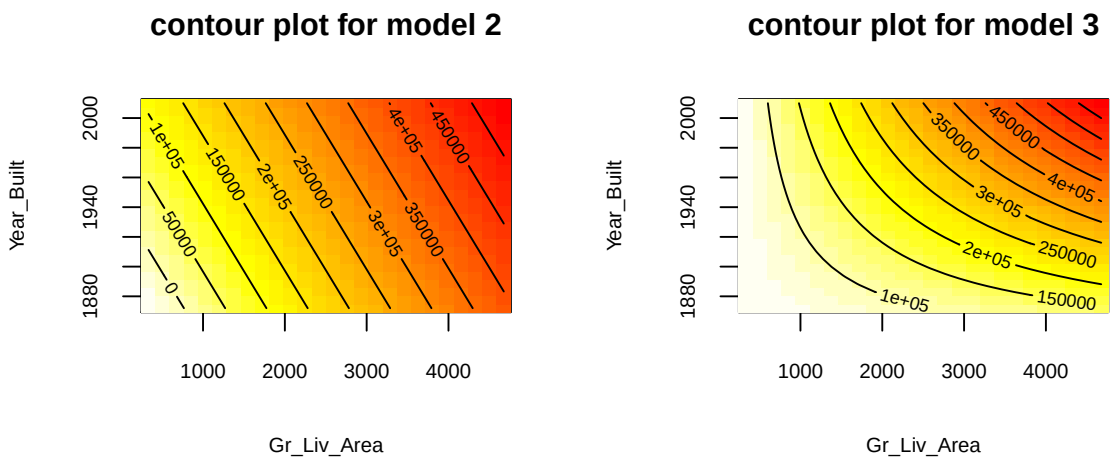


Figure 2: Contour plot of the fitted regression surface

The left one only has main effects and is therefore flat. Including interaction effects models curvature: the effect of one predictor now depend on the level of the other. So in our example this would mean including the product of both predictors:

$$Y_i = \beta_0 + \beta_1 X_i + \beta_2 X_2 + \beta_3 X_1 X_2 \epsilon_i, \qquad for \quad i = 1, 2, ..., n,$$

In R the formula is either `y ~ x1 + x2 ~ x1:x2` or `y ~ x1 * x2`.

Note the *hierarchy principle* which means that any lower order terms corresponding to the interaction term must also be included in the model.

You can include as many predictors as you like - as long as you have more observations than predictors! (So in wide tables you cannot include all of them!). These can also be interactions, or transoformations: e.g. $X_3 = X_1 X2$ or $X_4 = \sqrt(X_3)$. Of course after two dimensions visualisation becomes impractical, because we have a hyperplane of best fit.

We can try all of the predictors in the data set and clean up the output using the `broom` package: (!!! again, the results are even more different than before).

```
model3 <- lm(Sale_Price ~ ., data = ames_train)

broom::tidy(model3)
```

```
## # A tibble: 283 x 5
##    term                                estimate std.error statistic p.value
##    <chr>                                  <dbl>     <dbl>     <dbl>   <dbl>
##  1 (Intercept)                         -5.61e6 11261881.    -0.498   0.618
##  2 MS_SubClassOne_Story_1945_and_Older  3.56e3     3843.     0.926   0.355
##  3 MS_SubClassOne_Story_with_Finished~  1.28e4    12834.     0.997   0.319
##  4 MS_SubClassOne_and_Half_Story_Unfi~  8.73e3    12871.     0.678   0.498
##  5 MS_SubClassOne_and_Half_Story_Fini~  4.11e3     6226.     0.660   0.509
##  6 MS_SubClassTwo_Story_1946_and_Newer -1.09e3     5790.    -0.189   0.850
##  7 MS_SubClassTwo_Story_1945_and_Older  7.14e3     6349.     1.12    0.261
##  8 MS_SubClassTwo_and_Half_Story_All_~ -1.39e4    11003.    -1.27    0.206
##  9 MS_SubClassSplit_or_Multilevel      -1.15e4    10512.    -1.09    0.276
## 10 MS_SubClassSplit_Foyer              -4.39e3     8057.    -0.545   0.586
## # ... with 273 more rows
```

4.4 Assessing model accuracy

So now we have three main effects models, a single predictor one, one with two predictors and one with all of the features. WHich is best? Let's use RMSE and cross-validation. (this means resampling from the training dataset and validating on sub-folds, and then taking the average RMSE, instead of just the RMSE of the models as given in the summary()).

So we can use `caret::train()` to train the model using cross-validation, which is not available directly in the `lm()` funciton

```
# Train model using 10-fold cross-validation
set.seed(123)  # for reproducibility
(cv_model1 <- train(
  form = Sale_Price ~ Gr_Liv_Area,
  data = ames_train,
  method = "lm",
  trControl = trainControl(method = "cv", number = 10)
))
```

```
## Linear Regression
##
## 2053 samples
##    1 predictor
##
## No pre-processing
## Resampling: Cross-Validated (10 fold)
## Summary of sample sizes: 1846, 1848, 1848, 1848, 1848, 1848, ...
## Resampling results:
##
##   RMSE      Rsquared   MAE
##   56410.89  0.5069425  39169.09
##
## Tuning parameter 'intercept' was held constant at a value of TRUE
```

So when applied to unseen data, model1 is on average $56,600 off the mark. Let's perform cv on the other two models as well.

```
# Train model using 10-fold cross-validation
set.seed(123)  # for reproducibility
(cv_model2 <- train(
  form = Sale_Price ~ Gr_Liv_Area + Year_Built,
  data = ames_train,
  method = "lm",
  trControl = trainControl(method = "cv", number = 10)
))
```

```
## Linear Regression
##
## 2053 samples
##    2 predictor
##
## No pre-processing
## Resampling: Cross-Validated (10 fold)
## Summary of sample sizes: 1846, 1848, 1848, 1848, 1848, 1848, ...
## Resampling results:
##
##   RMSE      Rsquared   MAE
##   46292.38  0.6703298  32246.86
##
## Tuning parameter 'intercept' was held constant at a value of TRUE
```

```r
set.seed(123)  # for reproducibility
(cv_model3 <- train(
  form = Sale_Price ~ .,
  data = ames_train,
  method = "lm",
  trControl = trainControl(method = "cv", number = 10)
))
```

```
## Linear Regression
##
## 2053 samples
##   80 predictor
##
## No pre-processing
## Resampling: Cross-Validated (10 fold)
## Summary of sample sizes: 1846, 1848, 1848, 1848, 1848, 1848, ...
## Resampling results:
##
##   RMSE   Rsquared   MAE
##   26098  0.8949642  16258.84
##
## Tuning parameter 'intercept' was held constant at a value of TRUE
```

```r
# collect results from all three resamplings
summary(resamples(list(
  model1 = cv_model1,
  model2 = cv_model2,
  model3 = cv_model3
)))
```

```
##
## Call:
## summary.resamples(object = resamples(list(model1 = cv_model1, model2
##  = cv_model2, model3 = cv_model3)))
##
## Models: model1, model2, model3
## Number of resamples: 10
##
## MAE
##            Min.  1st Qu.   Median     Mean  3rd Qu.     Max. NA's
## model1 34457.58 36323.74 38943.81 39169.09 41660.81 45005.17    0
## model2 28094.79 30594.47 31959.30 32246.86 34210.70 37441.82    0
## model3 12458.27 15420.10 16484.77 16258.84 17262.39 19029.29    0
##
## RMSE
##            Min.  1st Qu.   Median     Mean  3rd Qu.     Max. NA's
## model1 47211.34 52363.41 54948.96 56410.89 60672.31 67679.05    0
## model2 37698.17 42607.11 45407.14 46292.38 49668.59 54692.06    0
## model3 20844.33 22581.04 24947.45 26098.00 27695.65 39521.49    0
##
## Rsquared
##             Min.   1st Qu.    Median      Mean   3rd Qu.      Max. NA's
## model1 0.3598237 0.4550791 0.5289068 0.5069425 0.5619841 0.5965793    0
## model2 0.5714665 0.6392504 0.6800818 0.6703298 0.7067458 0.7348562    0
## model3 0.7869022 0.9018567 0.9104351 0.8949642 0.9166564 0.9303504    0
```

!!! there is an error here, for sure. I get the same MAE results for all three models, and same Rsuared also, but RMSE is the same only for model1 and model2, not model3 though, where

mine are dramatically lower than in the book.

The function `caret::resamples()` allows you to compare the results of the resamplings. (!!! Again, my results are quite different from the book) The two predictor model has an average out of sample RMSE 46,292, and the all predictor model it's 26,098. Judging only by RMSE, modle 3 is the best.

4.5 Model concerns

There are several strong assumptions required by linear regression, that are often violated. What are they and what can you do about them?

1. **linearity of relationship**: if the relationship isn't linear, there are still transformations that could make it so. See e.g. the relatioship between the year the house was built and the price. It's not linear, but log-transforming the target variable can make it more so.

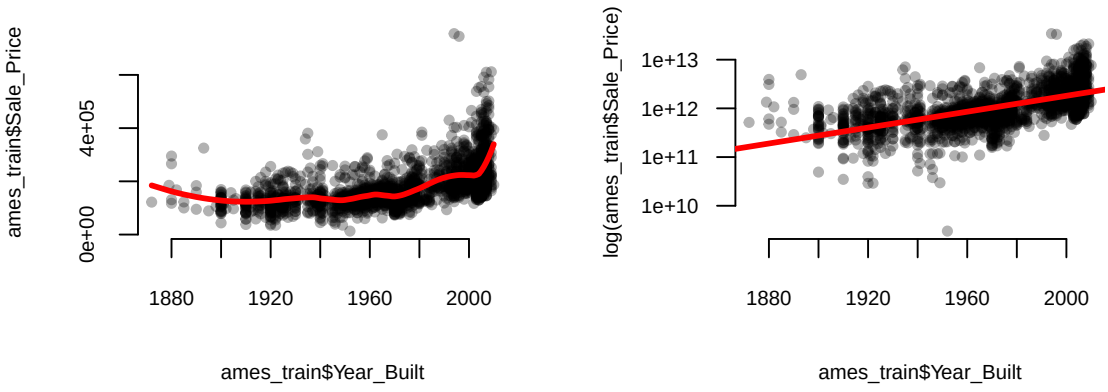


Figure 3: Transforming the target var to make the relationship more linear

2. **constant variance among residuals** aka homoscedascity, assumes that the variance among the error terms $(\epsilon_1, \epsilon_2, ..., \epsilon_n)$ is constant. If this is not the case, the p-values and confidence intervals will be wrong.

NB: use the `broom::augment()` to add information about each observation to the dataset. Usually predictions and residuals, also standard errors. You pass a model, and either the original data, or `newdata`, that was not used to fit the model.

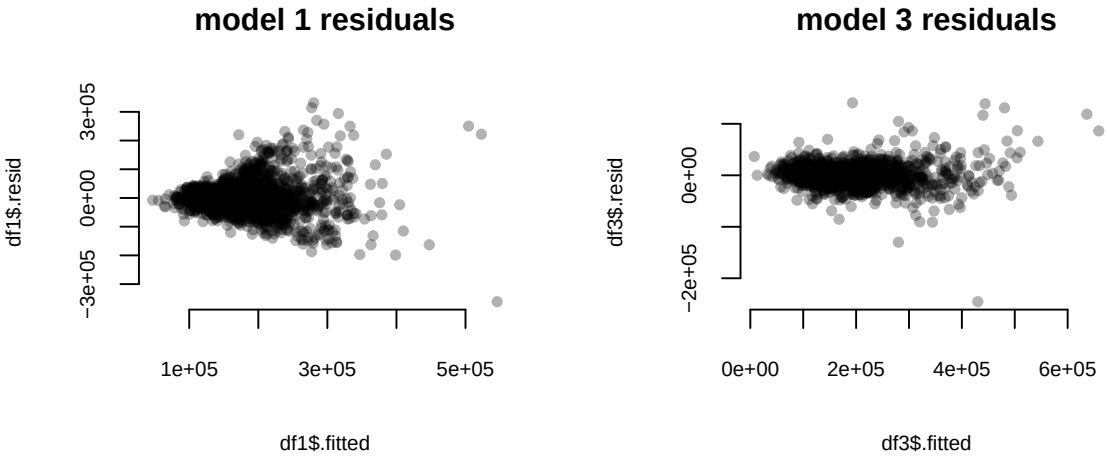


Figure 4: Residuals for models 1 and 3

You can see that adding all the predictors to the model made the residuals much more homoscedastic i.e. have constant variance.

3. **no auto-correlation** of the residuals. The residuals are supposed to be independent and uncorrelated. In Figure 5 you can see that the residuals in model 1 have a pattern, which means that ϵ_k is not independent of ϵ_{k-1}. This pattern is (probably) the result of the data being ordered by neighborhood, and of course homes in the same neighborhood are similar in a lot of ways, but our model does not account for that (it only includes living area). As soon as we add neighborhood into the model (as in model 3), the autocorrelation disappears. (but how would you notice this if they were in another order?)

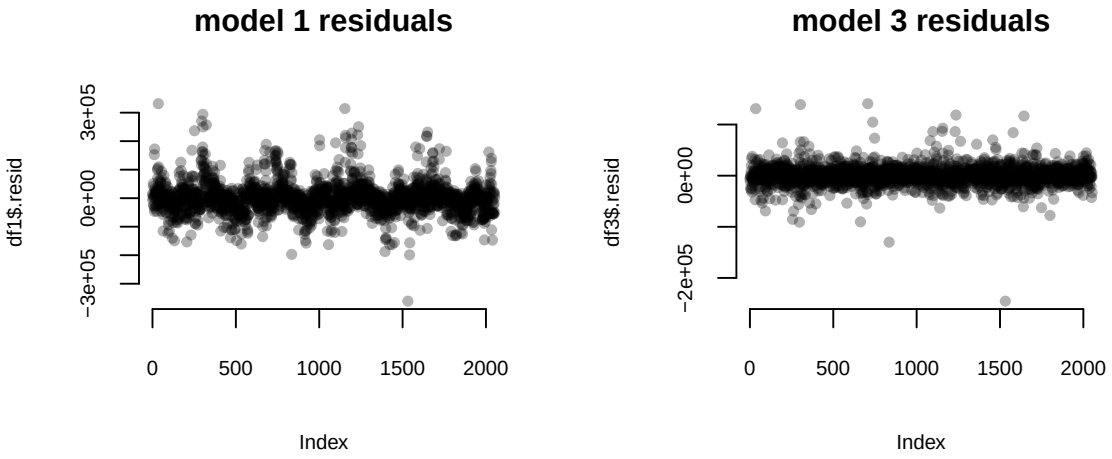


Figure 5: Auto-correlation of residuals for model 1 and much less so for model 3

4. **more observations than predictors** - if this is not the case and $p > n$ then you cannot compute an OLS estimate. You can then remove features one at a time, until $p < n$, using some pre-processing tools to guide you. But regularized regression is not limited by this, so use that instead.

5. **nu or little multi-colinearity** - *colinearity* refers to the situation where two or more predictors are closely correlated. For example the two garage variables have a correlation of 0.89 and are both correlated with Sale_Price as well. In the final model, one of them has a significant coefficient, the other not:

```
# correlation between the two variables is high
cor(ames_train$Garage_Cars, ames_train$Garage_Area)
```

```
## [1] 0.8906198
```

```
# one is significant, one not.
summary(cv_model3) %>%
  broom::tidy() %>%
  filter(term %in% c("Garage_Area", "Garage_Cars"))
```

```
## # A tibble: 2 x 5
##   term        estimate std.error statistic p.value
##   <chr>          <dbl>     <dbl>     <dbl>   <dbl>
## 1 Garage_Cars   3021.    1771.        1.71 0.0882
## 2 Garage_Area     19.7      6.03      3.26 0.00112
```

But if we remove `Garage_Area`, then Garage Cars suddenly becomes terrifically significant. (!!! again, my results are different than in the book. In fact just the inverse, I have Garage_Area significant in model3, not Garage_cars.)

```
set.seed(123)
mod_wo_Garage_Area <- train(
  Sale_Price ~ .,
  data = select(ames_train, -Garage_Area),
  method = "lm",
  trControl = trainControl(method = "cv", number = 10))

summary(mod_wo_Garage_Area ) %>%
  broom::tidy() %>%
  filter(term %in% c("Garage_Cars"))
```

```
## # A tibble: 1 x 5
##   term        estimate std.error statistic       p.value
##   <chr>          <dbl>     <dbl>     <dbl>         <dbl>
## 1 Garage_Cars    7161.     1239.      5.78 0.00000000881
```

"This reflects the instability in the linear regression model caused by between-predictor relationships; this instability also gets propagated directly to the model predictions". And since almost half of the predictors have moderate or high correlations, this is likely to be limiting the predictive accuracy of the model.

What do you do? One option is to remove predictors one by one, until the pair-wise correlations fall under a specific level. But this is tedious.

But also, multi-collinearity can arrise if one predictor is linearly related to several other features, and that is more difficutl to detect (you can use a statistic called *variance inflation factor* to figure it out), and even more difficult to remove.

So there are two main extensions of OLS, to help deal with multi-colinearity. One is **Principal component regressison**, the other **Partial least squares**, in both cases we are using dimensin reduction as pre-processing before running the regression. Alternatively *regularized regression* is introduced a few chapters dow.

4.6 Principal component regression

PCR is the two step process of dimension reduction, pre-processing the features into a smaller number of un-correlated principal components, and then running linear regression on them.

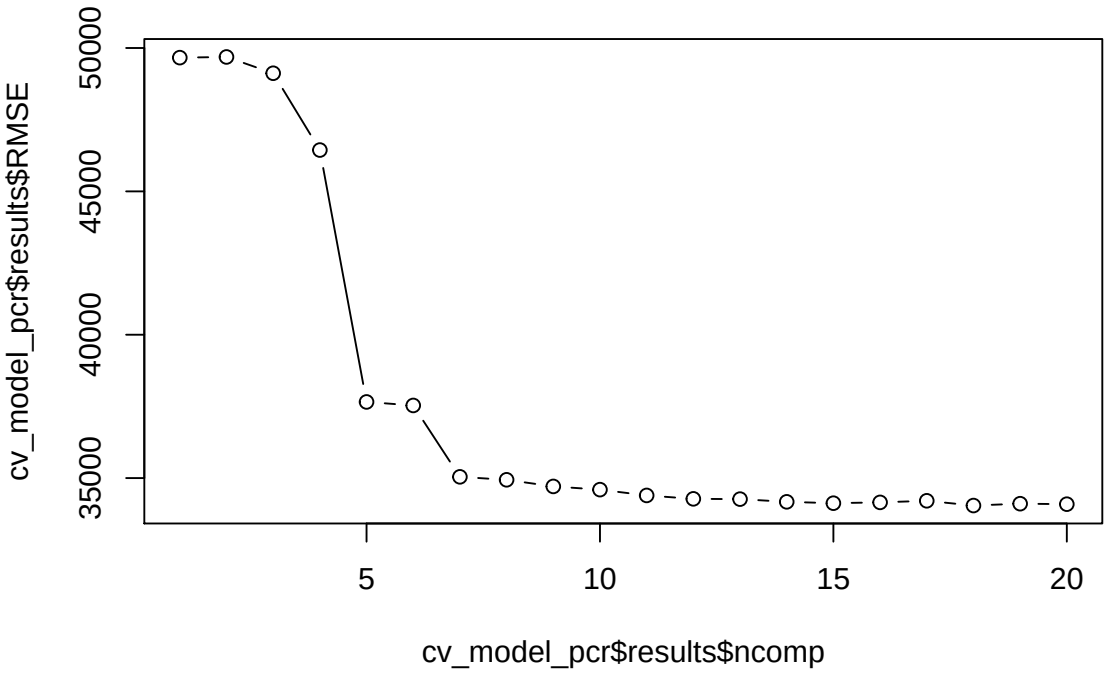
In `caret` we simply specify `method = pcr` in the `train()` call. This code also removes zero variance variables, and centers and scales the variables via the caret package instead of via recipes like we did last time. `tuneLength` controls the number of levels the parameter grid search should have. In this case we are looking at PCA, so the tuning parameter we are optimising is the number of principal components included in the model, in this case fom 1 to 20.

```
set.seed(123)
cv_model_pcr <- train(
  Sale_Price ~ .,
  data = ames_train,
  method = "pcr",
  trControl = trainControl(method = "cv", number = 10),
   preProcess = c("zv", "center", "scale"),
  tuneLength = 20)

# model with lowest RMSE
cv_model_pcr$bestTune
```

```
##    ncomp
## 18    18
```

```r
# plot cross-validated RMSE
plot(cv_model_pcr$results$ncomp, cv_model_pcr$results$RMSE, type = "b", )
```



!!! the note mistakenly says we are removing the nzv featrures, but it's only zv.

```r
## second attempt using recipes, should be the same thing.
library(recipes)
blueprint <- recipe(Sale_Price ~ ., data = ames_train) %>%
  step_zv(all_nominal()) %>%
  step_center(all_numeric(), -all_outcomes()) %>%
  step_scale(all_numeric(), -all_outcomes()) %>%
  step_pca(all_numeric(), -all_outcomes(), num_comp = 20)

# Specify resampling plan
cv <- trainControl(
  method = "cv",
  number = 10)

# preprocess and pick only PC variables:
blueprint %>%
  prep(ames_train) %>%
  bake(ames_train) %>%
  select(Sale_Price, starts_with("PC"))-> ames_pca

# train model using preprocessed data and crossvalidation.
set.seed(123)
cv_model_pcr2 <- train(
  Sale_Price ~ .,
  data = ames_pca,
  method = "lm",
  trControl = cv,
  metric = "RMSE")

# what is the average RMSE
cv_model_pcr2$results
```

```
##   intercept     RMSE  Rsquared      MAE RMSESD RsquaredSD    MAESD
## 1      TRUE 37390.75 0.7859008 26024.01 5196.7 0.04336029 2142.716
```

```
cv_model_pcr$results %>%
  filter(ncomp == 20)
```

```
##   ncomp     RMSE  Rsquared      MAE   RMSESD RsquaredSD    MAESD
## 1    20 34089.52 0.8238388 23349.98 4877.866 0.03389041 1237.785
```

!!! also, I'm not clear on how to do this same thing with the recipes/blueprings like suggested in the text. If I just copy the structure from chapter 3, the problem is two-fold, one that `step_pca` retains the original features after the pca. But you don't want to use them in the lm, you only want the PC. The second is that when running a `lm` in `train()`, I don't know how to tune on the number of variables used - i.e. the number of PCs included in the model. But I did attempt to get a second version of the model using this approach, only for 20 PCs, but the RMSE or other stats don't look similar enough really.. don't know why.

!!!Anyway, for them PCR is better than regular regression, but not for me, not at all, my PRC is absolutely shit compared to the regular one, which had a mean RMSE of 26,098$

4.7 Partial Least Squares

PLS is a type of supervised dimension reduction procedure. In PCA (or PCR rather), the components are constructed to maximally summarise the variability of the features, but ignoring the correlation with the target variable. In PLS however, the components are constructed while at the same time trying to maximise the correlation between the PCs and the outcome.

```
## try alternative to get pls as well, to get the PCs
library(recipes)
blueprint <- recipe(Sale_Price ~ ., data = ames_train) %>%
  step_zv(all_nominal()) %>%
  step_center(all_numeric(), -all_outcomes()) %>%
  step_scale(all_numeric(), -all_outcomes()) %>%
  step_pls(all_numeric(), outcome = "Sale_Price")

# Specify resampling plan
cv <- trainControl(
  method = "cv",
  number = 10)

# preprocess and pick only PC variables:
blueprint %>%
  prep(ames_train) %>%
  bake(ames_train) %>%
  select(Sale_Price, starts_with("PL"))-> ames_pls
```

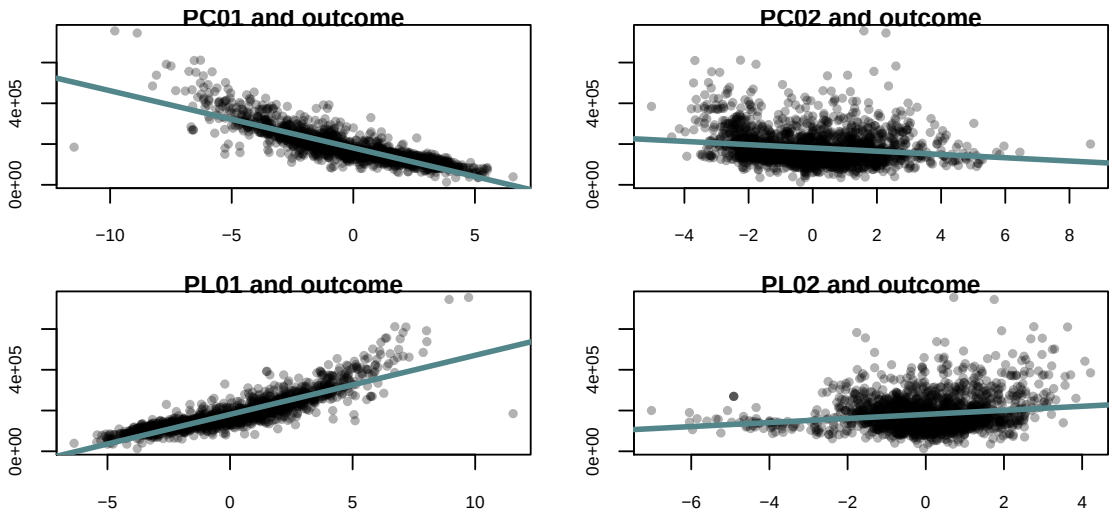


Figure 6: correlation between main PCs and outcome for PCR (top) and PLS components and outcome (bottom)

!!! The "exemplar data" illustration seems rather disingenuous.. it's not reproducible, because you're using an unspecified dataset. but at the same time if i attempt to reproduce it using the ames dta at hand, i get a really nice strong correlation, at least with PC01. So my reproduction in Figure 6 doesn't seem to show any difference in how correlated the components are between the two approaches..

!!! Also, the x-axis isn't the eignenvalue - each component has a single eigenvalue, this is the actual PC values, no?

OK, then there is an overly complicated explanation of how it works, by refering to an equation in chapter 17... But essentially the components are calculated by giving the highest weight to the variables most related to the response.

```
##    ncomp
## 20    20
```

```
## [1] 25459.51
```

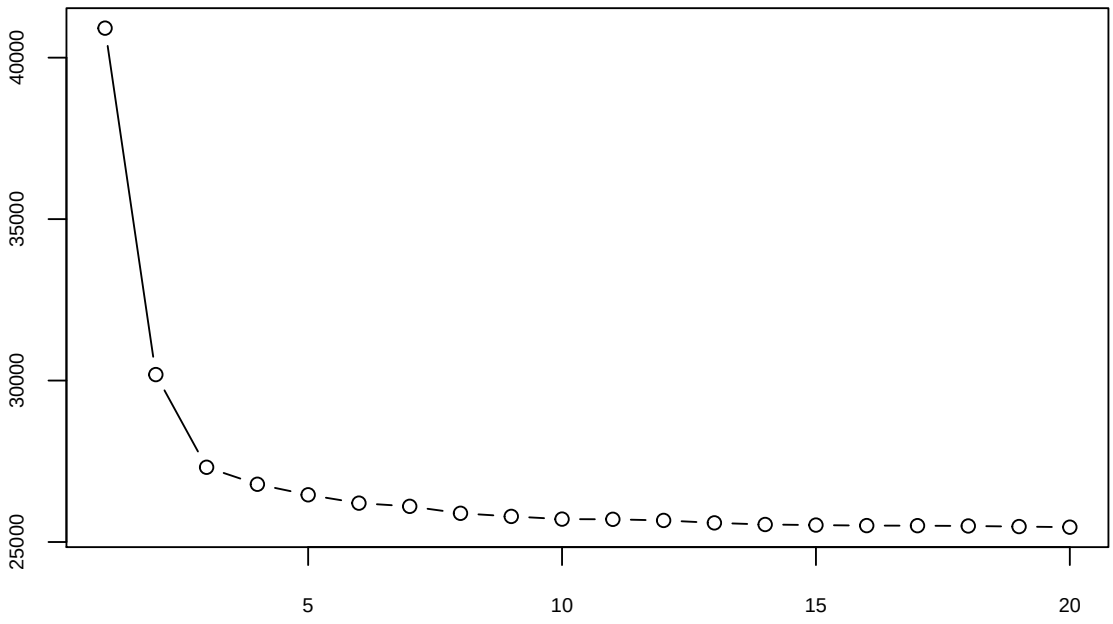


Figure 7: RMSE by number of components in PLS model

!!! the Figure 7 results are also quite different for me, I get a pretty monotonic chart, not the dip that you guys get. same seed.

4.8 Feature Interpretation

Once you've found a model maximising accuracy, the next goal is *interpretation of the model structure*. Linear regression lends itself quite nicely to this: it's untuitive that the relationship between the variables is *monotonic* and *linear*.

How do you figure out which variables are the most important? Usually just compare the $t - statistics$, although as soon as you have interactions or transformation of the variables, these become more difficult to interpret.

The `vip` package is apparently helpful in visualising the most important variables, here they are for the PLS model. !!! slightly different to the ones reported in the book thoug.

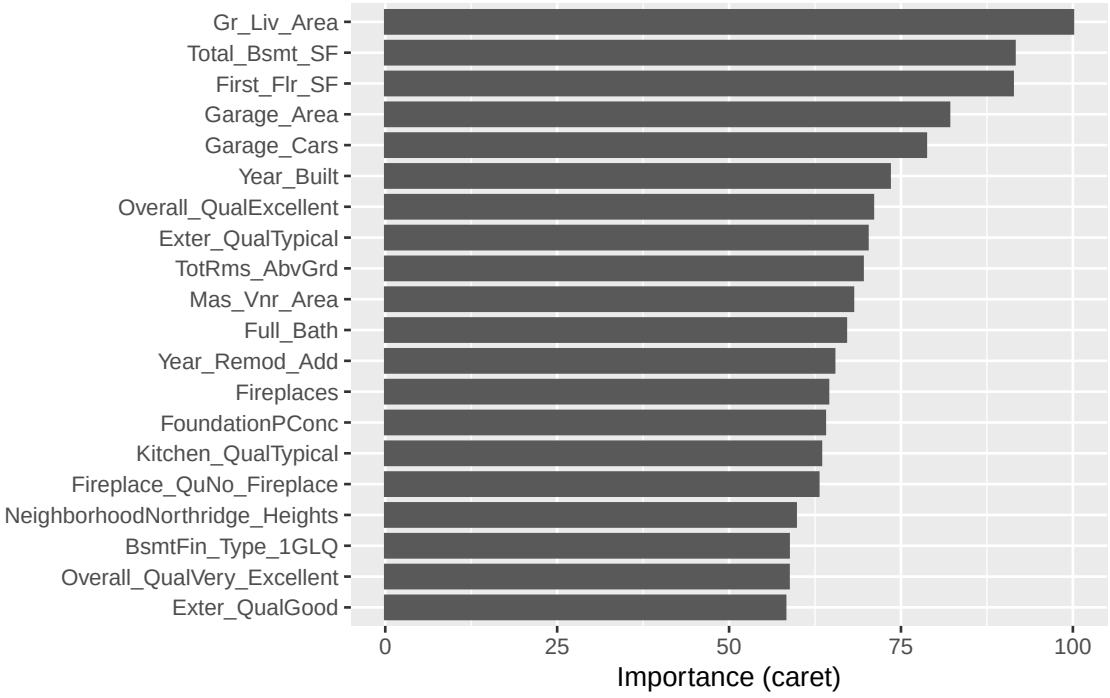


Figure 8: Top vars in PLS model

4.8.1 Partial dependence plots

PDPs plot the change in the averge predicted value of the target variable $\hat{Y}$ over the marginal distribution of individual features. This gets more interesting in other models, not so much with the linear ones, but whatevs.

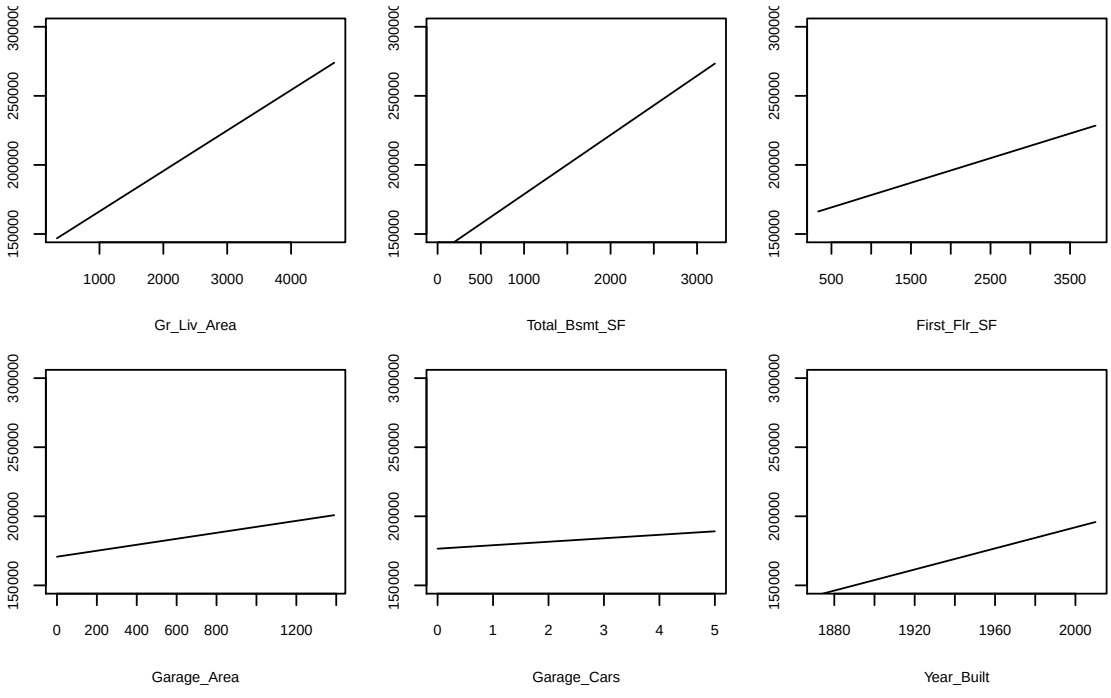


Figure 9: PDPs for some main and less main vars in PLS

!!! The text implies that less important variables have a smaller slope, but this doens't seem to be the case. At least with the sixth one.

!!! Hm, so the top features are all continuous, so you can plot them, but the categorical ones you cannot do a PDP. The foact that the top 4 are cont. is just a coincidence, but it means you don't explain what to do with the other features.

4.9 Final thoughts

Linear regression is a basic supervised learning alg, but has some assumptions that can be problematic. These can be addressed with extensions and dimension reduction steps, but other algs are better in the end..