Key Concepts in Statistical Machine Learning

Notes

SML algorithms can be:

- **Programmed** on a computer (so can be repeated).
- **Evaluated** quantitatively (using a computer) where prediction accuracy is our main measure of performance.
- **Tuned** - optimized automatically. Every machine learning model can be adjusted and improved.

Two main goals of SML

- **Inference** - learn about the data, fit models, test significance
- **Prediction** - make statements about something that is not observed, is unknown, or uncertain.
- Some combination of Inference or Prediction

Basics of learning from data

- 1. We have an ***outcome*** (output/DV) that we set to learn based on a set of features (predictors/IV/inputs)
  2. We have a **training** set of data in which we view the outcome and features for a set of objects (e.g., people)
     1. Note: we always have to split our data between training data and test data.
  3. From this data, we build a prediction model, aka a learner. Good learner = accurately predicts outcome.
  4. This is an example of **supervised learning**. This varies from **unsupervised learning** in that we in the latter we do NOT know the outcome variable and describe how data is organized and clustered. Unsupervised learning This book mainly focuses on the former.

5. Semi-supervised Learning combines a small amount of labeled data with a large amount of unlabeled data during training. The algorithms exploit one or more assumptions about the relationships betwen the labelled and unlabelled data to make choices.
6. Reinforcement Learning is a dynamic form of machine learning where the algorithm tries to determine an optimal set of decisions based on multiple choices over time.
7. **Regression** has a numerical/quantitative response or output.
8. **Classification** has a categorical output Y that takes values from a discrete set of unordered categories.
9. **Parametric:** making assumption about relationship between X an Y. Less flexible.
10. **Non-Parametric:** not making assumption about relationship between X and Y. More flexible.

Flexible vs restrictive models

- **Flexible models** have more parameters (or degrees of freedom) so they can more closely match the input data.
  - Highly flexible models, with hundreds of parameters, can be complicated to interpret.
  - They may be known as "**blackbox**" models - since we can't explain what is happening inside the model to explain the prediction, e.g., why was someone turned down for a loan.
  - **Over-fitting the data** occurs when a model is too flexible so it matches sample data so closely its predictions on new data have high variance.
- **Restrictive models** have fewer parameters (or degrees of freedom).
  - They can be easier to interpret but usually do not match the data as well.
  - **Under-fitting the data** occurs when a model is too restrictive so it misses key features of the function f and its predictions on new data have high bias.

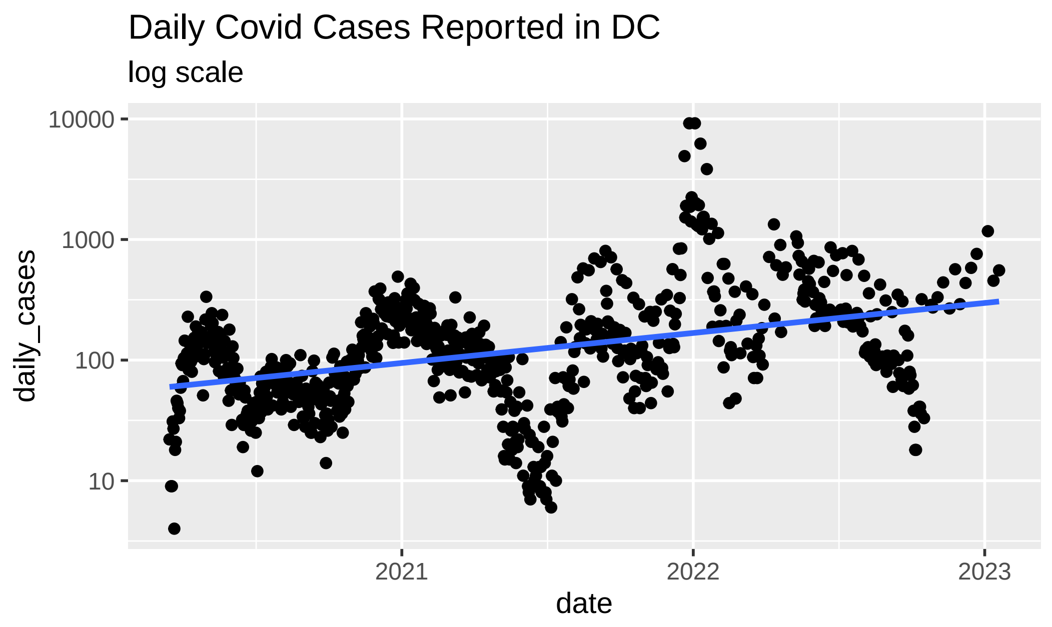
Degrees of freedom

- - Number of dimensions of space used to evaluate the variance
  - Variance places constraint on residuals
  - The reason we have n-1 degrees of freedom is to calculate sample mean
  - In OLS models, there are p+1 parameters (+1 is intercept, which represents sample mean), and hence we are losing p+1 degrees of freedom
  - More degrees of freedom means more flexible method or model
    - · Flexible models follow data more closely

- What is E(y - E(y))^2 for any random variable?
-
    * The expected value of our outcome minus the expected value of our outcome: is the variance of our response variable.
    * This is irreducible error. IT's randomness inherent in a variables distribution.

Example of degrees of freedom

```
url <- c("https://raw.githubusercontent.com/CSSEGISandData/COVID-19/master/csse_covid_19_data/csse_covid_1
readr::read_csv(url) |>
  dplyr::filter(FIPS == 11001) |>
  dplyr::select(tidyselect::contains("/")) |>
  tidyr::pivot_longer(cols = tidyselect::contains("/"), names_to = "date", values_to = "cases", names_trans
  dplyr::mutate(daily_cases = cases - dplyr::lag(cases)) |>
  dplyr::filter(daily_cases != 0) ->
  covid_19_dc
rm(url)
covid_19_dc |>
  ggplot2::ggplot(ggplot2::aes(date, daily_cases)) +
  ggplot2::geom_point() +
  ggplot2::geom_smooth(se = FALSE, method = lm, formula = y ~ x) +
  ggplot2::scale_y_log10() +
  ggplot2::labs(title = "Daily Covid Cases Reported in DC",
                subtitle = "log scale",
       caption = "COVID-19 Data Repository by the Center for Systems Science and Engineering (CSSE) at Johr
  covid_19_dc_plot
covid_19_dc_plot
```



epository by the Center for Systems Science and Engineering (CSSE) at Johns Hopkins University

Is it good to be flexible?

- Flexible methods have more parameters (d.f.) and can follow the sample data more closely
- **However, the higher the flexibility, the higher the variance so a too-flexible model may not predict very well**.

Mean squared error

MSE is the primary measure of performance for ML models. It is estimated with data NOT used to train a model. It has three components:

- variance of $\hat{y}$
- Squared bias
- Irreducible variance of y ("Irreducible error of MSE)
  - This is variance associated with our response variable's distribution and cannot be removed

Prediction error is equivalent to prediction accuracy – How well can a model predict the unknown. If we develop a method that has high variance it means there is a lot of noise in the predictions; change the data a little and the results or predictions can change quite a bit. So, in Regression problems when we have a numerical response variable, performance will be measured by ***prediction mean squared error** (MSE) defined as the expected value of the squared residuals

Tuning is the optimization of MSE

MSE can be hard to interpret, so we may want to take the square root of it to calculate the **root mean squared error**

$$MSE = E(\hat{y} - y)^2$$

Trade-off between bias and variance ("The bias variance trade-off")

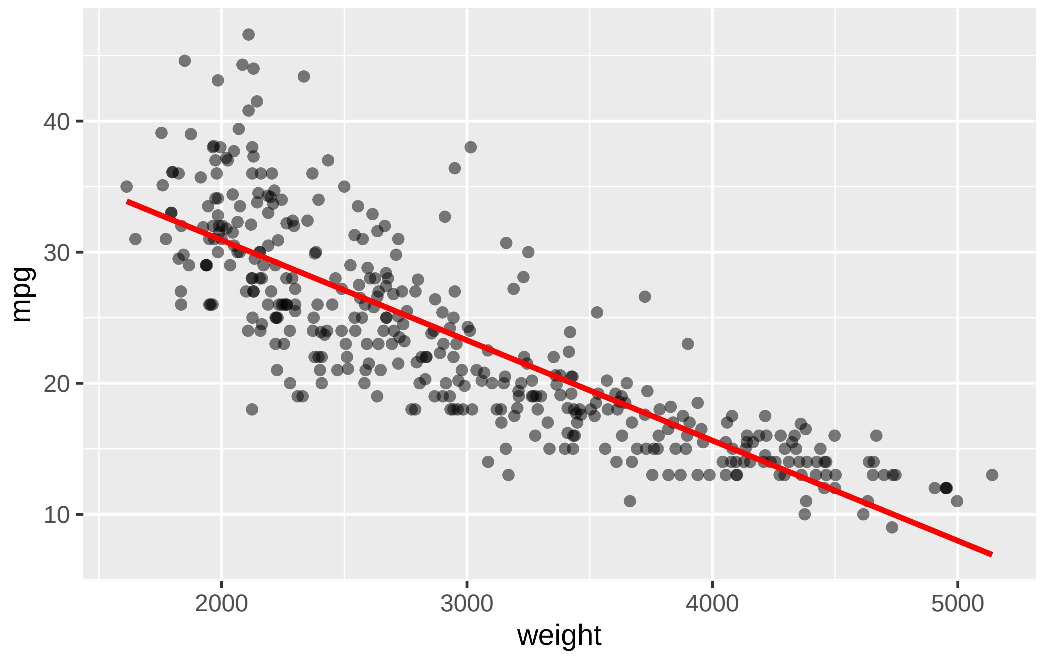
- Flexible Methods have low bias as they can follow the data but can have high variance
- Restrictive Methods have low variance and high bias as they may not match all the data well.
- This need to balance bias and variance in predictions is why SML is not just "plug and chug"– we have to think.

Example: Flexibility

Glancing at your data

When plotting Y ~ X, the relationship does not appear perfectly linear. More points fall below the best fit linear line than above.

```r
library(ISLR2)
attach(Auto)
library(tidyverse)
ggplot(Auto, aes(x = weight, y = mpg)) +
  geom_point(alpha = .5) +
  geom_smooth(method = lm, se = FALSE, col = "red", linewidth  = 2) ->
base_plot
base_plot
```



Calculating MSE

To calculate the Prediction MSE, we will split the data in half into training and testing data sets.

- To create a 50/50 split, we will get a random sample of size n/2 where n is the number of observations.
- We can use that sample as a vector of indexes to subset the data set.
- We will also set the seed for the pseudo-random number generator so our split of the observations is repeatable.

```
# extract training sample from full data
n <-  length(Auto$mpg)
set.seed(1234)
z <-  sample(n, n/2)

# Calculate the regression using the training subset of data selected by the indices in z.
reg <-  lm(mpg ~ weight, subset = z, data = Auto)

# Identifying test data by removing training data from original data set
Y <-  Auto$mpg[-z] # the True Y (can't use obs from training sample)

# predictions for test data w/ model generated from training data
Yhat <-  predict(reg, newdata = Auto[-z,])
```

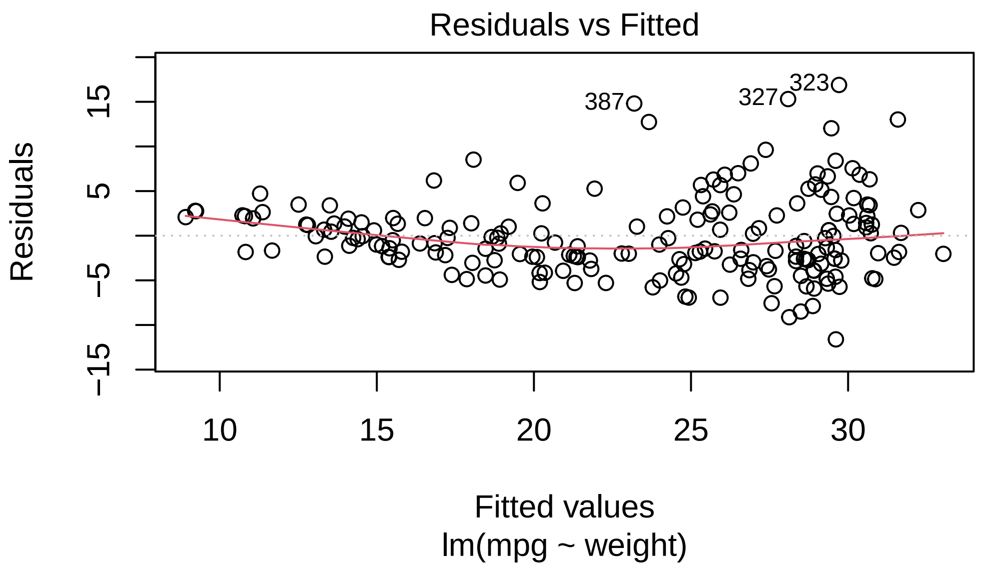
```r
# calculating MSE
MSE <- mean((Y - Yhat)^2)
MSE
```

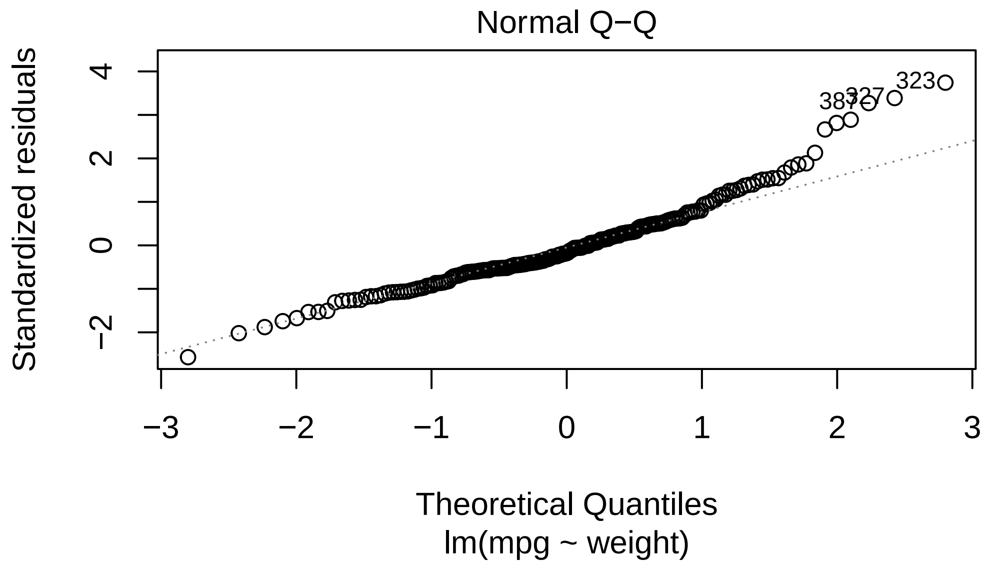
```
[1] 17.25805
```

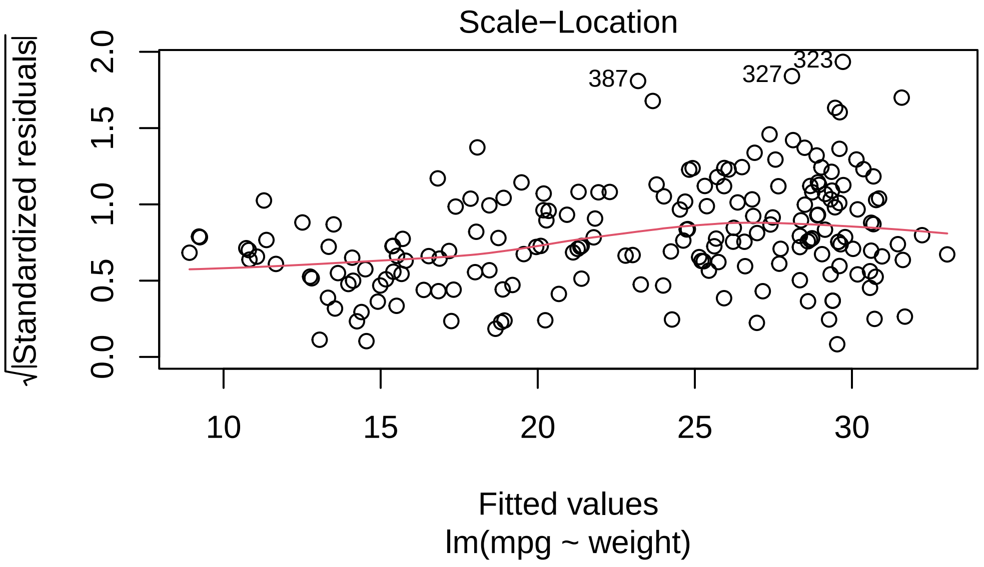
Problems with linear model

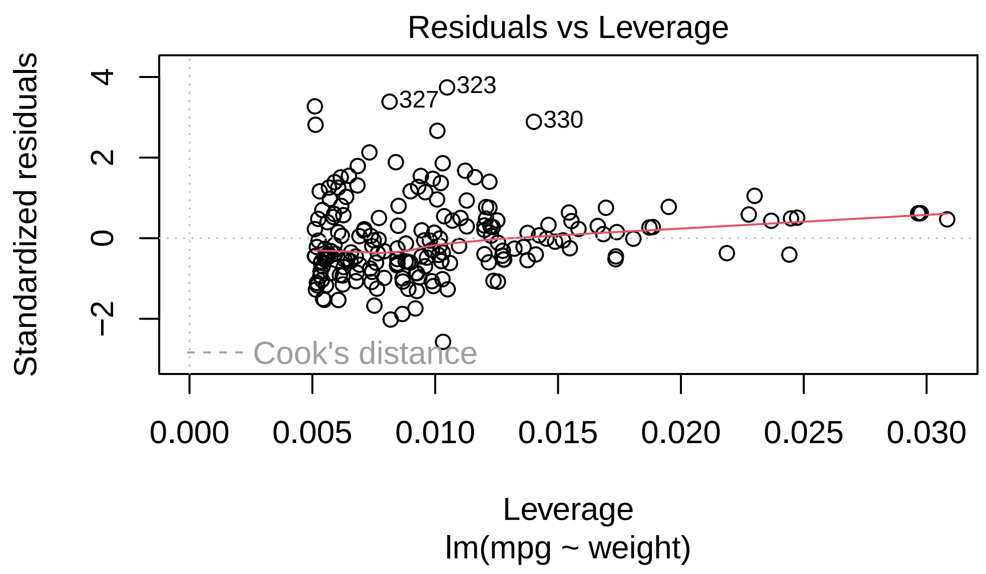
Linear regression is very restrictive/inflexible and hence is probably not representing the data very well. This can be seen in the model diagnostics.

```r
plot(reg)
```




Normal Q−Q
Standardized residuals
Theoretical Quantiles
lm(mpg ~ weight)
387
327
323


Scale−Location
√|Standardized residuals|
Fitted values
lm(mpg ~ weight)
387
327
323



- **Residuals vs fitted:** Used to identify if residuals indicate non-linear pattern. If line is straight = linear.
- **QQ plot**: Tests if data is normally distributed by mapping quantiles from our distribution against quantiles from a theoretical normally distributed distribution.
- **Scale-Location plot**: Tests for homoscedasticity, i.e., if there is equal variance in residuals across the range of predicted values.
- **Residuals vs leverage**: Helps identify influential points that could impact the predicted line.

From these tests, it looks like a non-linear model might better fit the data.

Using non-linear models

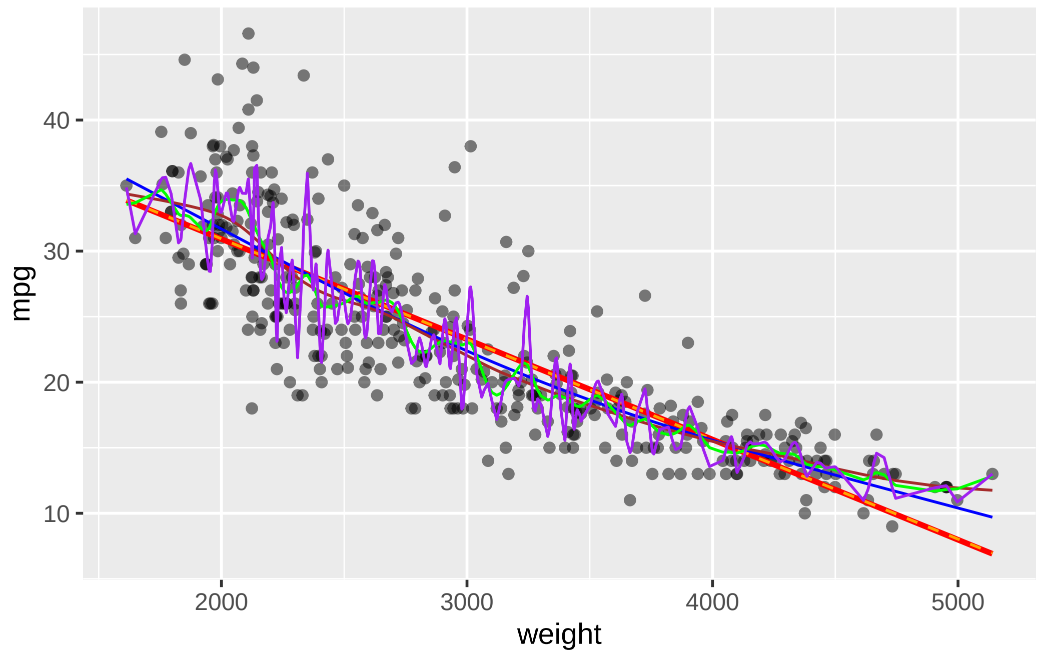
Let's try a flexible spline model.

- Spline models allow us to adjust the number of degrees of freedom or parameters in the model to provide more flexibility.
- This makes `df` a **hyper-parameter** we can use later to **tune** our model.

Let's compare different amounts of degrees of freedom.

You can see as the degrees of freedom increase, the line is more flexible and matches the data better, but the model might be less accurate for prediction.

```
library(ggformula) # the package for geom_spline
base_plot +
  geom_spline(df = 2, color = "orange", lty = 2) +
  geom_spline(df = 3, color = "blue",  linewidth = 2) +
  geom_spline(df = 10, color = "brown", linewidth = 2) +
  geom_spline(df = 40, color = "green", linewidth = 2) +
  geom_spline(df = 140, color = "purple", linewidth = 1)
```



Using MSE to compare the predictive power of models with different degrees of flexibility.

We start with splitting the data into Training and Test data sets.

- Previously we split the data evenly in half - 50% Training and 50% Test.
- What are the implications for this kind of split?
  - How well will the model fit the data using only n/2 points in the training data?
  - Is there a "better split" than 50%?
  - It appears we have another possible "hyper-parameter" for tuning our model.

Formula:

$$\hat{MSE}_{Test} = \frac{1}{n_{Test}} + \sum(y_i - \hat{y}_i)^2$$

Let's run an initial model.

Step one is create training and test data. Instead of splitting 50-50 let's pick 200 cars for our training set - out of 392 observations.

```r
# identify training and test data, and then run model on the training data
set.seed(123)
z <-  sample(nrow(Auto),200)
training <-  Auto[z,]
reg <- lm(mpg ~ weight, data = training)

# MSE comparing training and test data
Yhat <-  predict(reg, newdata = Auto)
MSE_test <-  mean((Yhat[-z] - Auto$mpg[-z])^2)
MSE_test
```

```
[1] 17.44414
```

This MSE is hard to interpret. Let's now calculate the root mean squared error.

```r
RMSE_reg <- sqrt(MSE_test)
RMSE_reg
```

```
[1] 4.176619
```

Let's try fitting a more flexible spline model instead. Let's start off with a highly flexible model. This model will predict values much closer to the data, although the accuracy of the predictions will be highly dependent on the extremity of the predictors.

```r
ss <-  smooth.spline(Auto$weight[z], Auto$mpg[z], df = 100)

# Make the predictions. Note predict() returns a list of x and y. All we want is the y element.
Yhat <- predict(ss, Auto$weight[-z])$y

# Get the RSME and compare to RMSE_Reg
RMSE_ss <- sqrt(mean((Yhat - Auto$mpg[-z])^2))
RMSE_ss
```

```
[1] 6.205467
```

```
RMSE_reg
```

```
[1] 4.176619
```

Looks like the initial RMSE which came from a regression model with 198 degrees of freedom (to see this, run a summary on the original “reg” output) has a lower (and therefore better) RMSE compared to the version in which we specified 100 degrees of freedom.

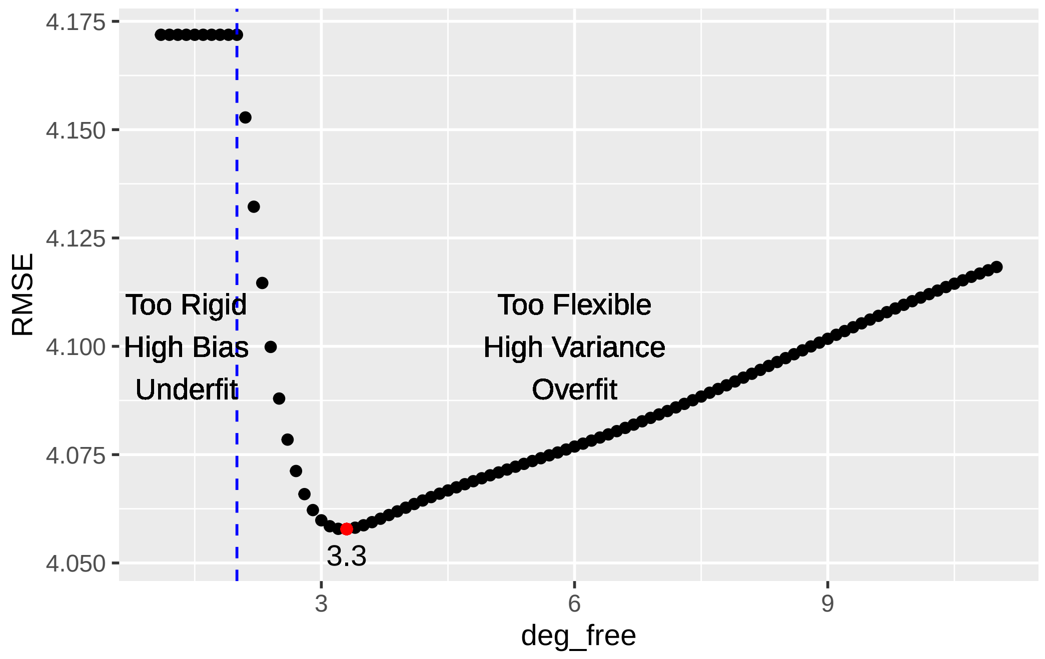
Fitting many models

Let’s fit 100 spline models with different degrees of freedom and plot how the RMSE changes with degrees of freedom.

```
# Calculating RMSE of different models
set.seed(123)
z <-  sample(nrow(Auto),200)
deg_free <- seq(1.1, 11, .1)
calc_rmse_spline <- function(x, y, index, df){
  ss <-  smooth.spline(x = x[index], y = y[index], df = df)
  Yhat <-  predict(ss, x = x[-index])$y
  RMSE <- sqrt(mean((Yhat - y[-index])^2))
}

RMSE <- map_dbl(deg_free, ~ calc_rmse_spline(Auto$weight, Auto$mpg, z, .))

# vizualizing the result
ggplot(bind_cols(deg_free = deg_free, RMSE = RMSE), aes(deg_free, RMSE)) +
  geom_point() +
  geom_point(aes(x = deg_free[(RMSE == min(RMSE))], y = min(RMSE)), color = ”red”) +
  geom_text(aes(x = deg_free[(RMSE == min(RMSE))], y = min(RMSE)), label = (deg_free[(RMSE == min(RMSE))])
  geom_vline(xintercept = 2, color = ”blue”, lty = 2) +
  geom_text(aes(x = 1.4, y = 4.1), label = ”Too Rigid\nHigh Bias\nUnderfit”) +
  geom_text(aes(x = 6, y = 4.1), label = ”Too Flexible\nHigh Variance\nOverfit”)
```


RMSE
4.175
4.150
4.125
4.100
4.075
4.050
Too Rigid
High Bias
Underfit
Too Flexible
High Variance
Overfit
3.3
3
6
9
deg_free