A primer on Regression

Tyrel Stokes

12/02/2022

Data Cleaning

All of the data was taken from kaggle. We have team standings, team boxscore, and player box score information. I will not discuss in length the cleaning procedure here, the focus is on the regression component

```
standings <- read.csv("2012-18_standings.csv")

standings$date2 <- as.Date(standings$stDate)

season_beg <- seq(from = as.Date("2012-10-01"),by = "year", length.out = 6)

season_end <- seq(from = as.Date("2013-07-01"),by = "year", length.out = 6)


### The below maps each date to a season

standings$season_int <-  unlist(lapply(standings$date2,function(x){
 c(1:6)[(x >season_beg) &(x < season_end)]
}))

### Now we will find out how many games are in each season

games_season <- standings %>% group_by(season_int) %>% summarise(ngames = max(gamePlay))

final_date <- standings %>% group_by(season_int) %>% summarise(ldate = max(date2))

### All full seasons

################

## Now we can get the standings for each team at the end of the year

standings_final <- standings %>% filter(date2 %in% final_date$ldate)

tm_box <- read.csv("2012-18_teamBoxScore.csv")

tm_box$date2 <- as.Date(tm_box$gmDate)
```

```
tm_box$season_int <- unlist(lapply(tm_box$date2,function(x){
 c(1:6)[(x >season_beg) &(x < season_end)]
}))


#######
#######

## Now we will aggregate some key stats on the season level for each team

tm_season <- tm_box %>% group_by(teamAbbr,season_int) %>%
  summarise(av_poss = mean(poss), av_pace = mean(pace),
            av_pts_for = mean(teamPTS),av_pts_ag = mean(opptPTS))

## `summarise()` has grouped output by 'teamAbbr'. You can override using the `.groups` argument.

#################################################################
#################################################################

dt_combine <- left_join(standings_final,tm_season, by = c("teamAbbr","season_int"))
```

Okay so we have this nice combined data, pulling a bunch of metrics together with the end of season standings.

Some points about regression

First let us start with linear regression. Linear regression is perhaps the foundational model of all of statistics. It is really well understood, simple, fast, and has tonnes of nice properties which can be exploited. The more that you can understand about linear regression the better, there is pretty much always something more to understand.

Basic Structure and Some Math

Let $Y = [Y_1, Y_2, \ldots, Y_n]$ be an $n \times 1$ vector of outcomes. n is the number of data points that we have, so we have n outcomes. In basketball an example outcome might be something like wins at the end of the year.

Let X be an $n \times p$ matrix of covariates. Covariates are things that we want to use to make predictions of inferences about the outcome with. Perhaps we expect the number of wins last season to be related to wins this year or some team stats in recent years or information about the players on the team. All of these could be covariates. What to include depends on the goals of the analysis and the analyst themself.

Recall some ideas from probability. First, we can let the conditional expectation of $Y|X$ (that is Y given X) is written $E[Y|X]$. In regression, quite often we are trying to model the conditional expectation. That is we propose some model to try to recover or approximate it in some way. Why do we do this?

There are lots of potential justifications. First, conditional expectations or conditional means are really useful. They can give us average predictions at different covariate levels. So suppose wins are the outcome and let's just suppose wins last year is the only covariate. I will introduce the notation to let $Wins_t$ be the wins in some season t and $Wins_{t-1}$ be the number of wins last year.

The conditional expectation $E[Wins_t|Wins_{t-1}]$ is a function that for any value of wins last year spits out an average for how many wins I would expect this year. That's useful to know and potentially answers a question. Recall from probability that we could write:

$$E[Wins_t|Wins_{t-1} = x]$$

to be the conditional expectation at some particular number of wins last year, this is a single number not a function anymore.

Okay so conditional expectations can be great (see the next section for some extra details for why we might be interested in the conditional expectation), but the problem is that we don't know what they are. In statistics when we don't know something, we have to estimate them. Linear regression (and regression more generally) is one particular way of estimating conditional means or making predictions.

The way linear regression works is that we decompose the outcome into two parts

1. The linear predictor, we denote $X\beta$. Where β is a p dimensional vector of coefficients.

2. The error. We usual write this ϵ.

So

$$Y = X\beta + \epsilon$$

We can always write a decomposition like this. If we make a linear prediction, it won't be perfect so there will always be some error. Notice $\epsilon = Y - X\beta$.

The cool thing about linear regression is it helps us pick the "best" coefficients. The coefficients which make the squared errors as small as possible. It is nice to know that the algorithm is in some sense best! What do we mean by that we mean that our estimator $\hat{\beta}$ minimizes the following

$$\hat{\beta} = \arg\min_{\beta \in \mathcal{R}^p} \sum_{i=1}^{n} (Y_i - X_i\beta)^2$$
$$= (X^T X)^{-1} X^T Y$$

From a geometric perspective $X\hat{\beta}$ is the best projection of Y onto linear combinations of X (think pythagorus theorem). One way to recall projection is to think about shadows or "dropping perpendiculars". Let's take a look at this in 2 dimensions. Predicting wins next year with wins in the previous season.

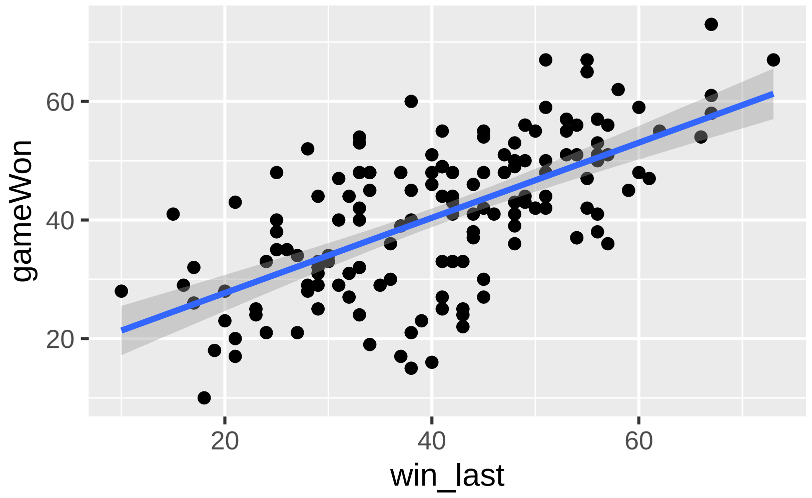
```
win_lag_df <- dt_combine %>% group_by(teamAbbr) %>% arrange(season_int) %>% mutate(win_last = dplyr::la
```

```
win_lag_df %>% ggplot(aes(x=win_last,y=gameWon)) +geom_point() + geom_smooth(method = "lm")
```

```
## 'geom_smooth()' using formula 'y ~ x'
```

```
## Warning: Removed 30 rows containing non-finite values (stat_smooth).
```

```
## Warning: Removed 30 rows containing missing values (geom_point).
```



In two dimensions we can directly plot and visualize the best linear approximation $X\hat{\beta}$ which is represented by the blue line above. The errors is the shortest distanace from each point to the line. Plotting it, we can see that the relationship seems mostly linear anyway, so likely linear regression is doing a fairly good job of capturing the true unknown relationship. The gray bands visualize the confidence bands or the uncertainty we have in the slope. Notice they get bigger on the extremeties where we have less data. The more data we have the more information we should have to be able to make our approximation good.

Here is how you run the code in R. We put the outcome on the left hand side of the tilde $\sim$ and all the predictors on the right hand side. In this case we have just one. Unless we tell it otherwise, R will also include an intercept term. Generally we want an intercept, but if for some reason you'd like not to you can add a -1 term on the predictor side of the equation.

```
mod <- lm(gameWon ~ win_last, data = win_lag_df)

summary(mod)
```

```
##
## Call:
## lm(formula = gameWon ~ win_last, data = win_lag_df)
##
## Residuals:
##      Min       1Q   Median       3Q      Max
## -24.3704  -5.8458   0.5016   6.6061  20.8974
##
## Coefficients:
##             Estimate Std. Error t value Pr(>|t|)
## (Intercept) 15.01585    2.69368   5.574 1.14e-07 ***
## win_last     0.63386    0.06279  10.095  < 2e-16 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 9.726 on 148 degrees of freedom
##   (30 observations deleted due to missingness)
## Multiple R-squared:  0.4078, Adjusted R-squared:  0.4038
## F-statistic: 101.9 on 1 and 148 DF,  p-value: < 2.2e-16
```

Here we see the table of estimates. Go to the coefficients part in the middle.

We have two coefficients. The first is the intercept of the line, estimated to be 15.015. That's our best guess estimate from the lin regression formula. And the error around that 2.7. Then we have the win_last coefficient, estimated to be about 0.63. What this tells us is that for every win we get in the previous season we expect about 0.63 wins next season.

One way again to think about linear regression is an approximation to the conditional expectation. If $E[Y|X]$ is the real condidtional expectation, then if we think the linear regression is a credible approximation to the conditional mean, we can call the regrrssion predictions $\hat{E}[Y|X]$. In the side-bar we show that from this perspective the coefficients can be interpreted as the increase in predicted outcome we get for every 1 increase in the predictor.

$$\begin{aligned} Y &= \alpha + X\beta + \epsilon \\ \implies E[Y|X = X] &= \alpha + X\beta \\ \implies E[Y|X = X+1] &= \alpha + (X+1)\beta \\ &= \alpha + X\beta + \beta \\ &= E[Y|X = X] + \beta \end{aligned}$$

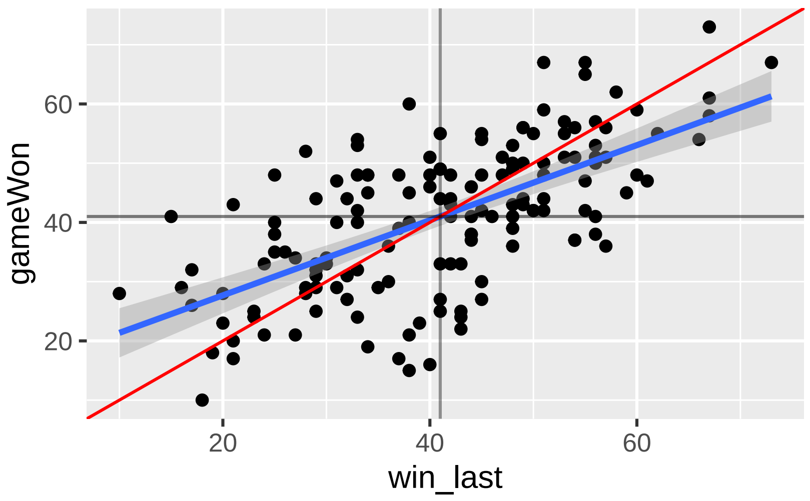
Now let's think even more carefully about the meaning of the coefficient for last seasons wins being 0.63. Notice that this is less than 1. I'm going to plot the 45 degree line (in red) and the average number of wins (41 in gray) on top of the linear regression plot to really appreciate this. The 45 degree line is what the slope would look like if the estimated coefficient was exactly 1.

```
win_lag_df %>% ggplot(aes(x=win_last,y=gameWon)) +geom_point() + geom_smooth(method = "lm") + geom_abli
```

```
## ‘geom_smooth()‘ using formula ’y ~ x’
```

```
## Warning: Removed 30 rows containing non-finite values (stat_smooth).
```

```
## Warning: Removed 30 rows containing missing values (geom_point).
```



Notice that the regression line and the 45 degree line intersect right at the average 42. When we move to the right of the average on the x-axis, the regression line is now lower than the 45 degree line and vice versa to the left. Notice as we move farther from the intercept the distance between the 45 degree line and our prediction gets larger

This is related to the concept of regression to the mean. Our linear regression prediction predicts that teams with more than above average seasons are going to be pulled back towards the overall average of 41. Similarly, we expect improvements from the worst performing teams. That's a prediction which makes sense in a competitive league with a salary cap and a draft structure.

That's the kind of thing we want to be thinking about when we use models. Does this make sense? How would I test if the predictions for the model are any good? We want to be very critical of our models and maintain healthy skepticism. That does not mean, however, that we throw out results that we can't make sense of right away either.\

Measures of goodness of fit

How good is my model you ask? One simple measure of performance in linear regression is $\mathcal{R}^2$. This can be interpreted as the amount of variance of the outcome linearly explained by the predictors in the model.

```
summary(mod)
```

```
##
## Call:
## lm(formula = gameWon ~ win_last, data = win_lag_df)
##
## Residuals:
##      Min       1Q   Median       3Q      Max
## -24.3704  -5.8458   0.5016   6.6061  20.8974
##
## Coefficients:
##             Estimate Std. Error t value Pr(>|t|)
## (Intercept) 15.01585    2.69368   5.574 1.14e-07 ***
## win_last     0.63386    0.06279  10.095  < 2e-16 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 9.726 on 148 degrees of freedom
##   (30 observations deleted due to missingness)
## Multiple R-squared:  0.4078, Adjusted R-squared:  0.4038
## F-statistic: 101.9 on 1 and 148 DF,  p-value: < 2.2e-16
```

In the summary again we can see the multiple R-squared and adjusted R-squared are 0.408 and 0.404 respectively. The adjustment corrects for the number of variables. In either case, we see the number of wins "explain" about 40% of the variance we see year to year in wins. That's pretty reasonably high. It is a little hard to in general say what high is though because it depends on the context. Things that are really noisy and hard to explain will have lower values in general. When R-squared is close to 1 we would see the prediction values and the actual values really tightly bound to near the prediction line. But when we compare models against each other on the same data large differences in R-squared tell us something about the signal we have found in the data. In other words, we want to try to mostly compare apples to apples, that is similar models on similar data sets to each other.

Another useful idea in regression and modeling is the idea of out of sample prediction. In the previous graph we were looking at predictions made on the same data used to build our model. Same with R-squared, it's a performance measure on the data we used to make the model. In fact the whole point of linear regression is that we pick the linear model which explains as much of the variance as possible. There is an idea that this is cheating a bit and we might be getting fooled by randomness or random patterns which seem to emerge in the data. One way to hedge against this is to test performance on data outside of your data. In our case our data is 2012-2018. We could take find the data from other seasons, make predictions and evaluate performance on that.

Another way without getting new data is to hide some of your data from yourself. Here I will hide the data from the last season, make predictions and test performance there, plotting the results.

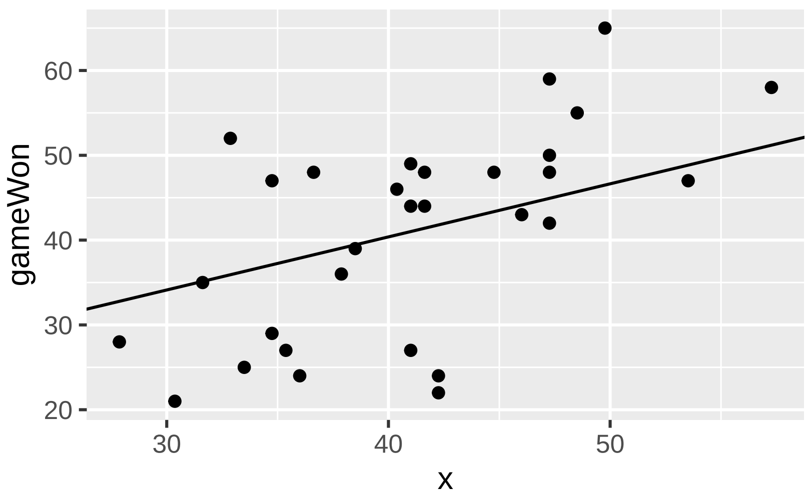
The adjusted R-squared "adjusts" for the number of covariates that are included. It can be shown that you can never decrease unadjusted (centered) R-square by adding a variable. Intuitively this is because we can always set the coefficient of the new variable to 0 which would make the prediction the same as before, so if we are optimizing we can't do worse. Here's one way to look at the formulas

$$R^2 = 1 - \frac{Var(\hat{\epsilon})}{Var(y)}$$

$$R^2_{adj} = 1 - \frac{\frac{n-1}{n-p-1} Var(\hat{\epsilon})}{Var(y)} \quad (1)$$

where p is the number of covariates. When the number of covariates is small, the two metrics will be very close to one another.

```
mod_2 <- lm(gameWon ~ win_last, data = (win_lag_df %>% filter(season_int!= 6)))
pred_season_6 <- predict(mod_2,(win_lag_df %>% filter(season_int== 6)))
real_data <- (win_lag_df %>% filter(season_int== 6)%>%select(gameWon))

## Adding missing grouping variables: 'teamAbbr'

pred_data <- data.frame(x = pred_season_6,real_data)

pred_data %>% ggplot(aes(x = x, y = gameWon)) + geom_point() + geom_abline(slope = coef(mod_2)[2], inte
```



The above is the fit on the data we didn't use, it doesn't look as convincing at a glance but it is hard to tell overall because the graph is on a slighlty different scale (we could modify to make this not the case). We can calculate the R-squared on this held out data.

```
rss <- sum((pred_data$gameWon - pred_data$x)^2)
tss <- sum((pred_data$gameWon - mean(pred_data$gameWon))^2)
R2 <- 1 - rss/tss
R2
```

```
## [1] 0.3840244
```

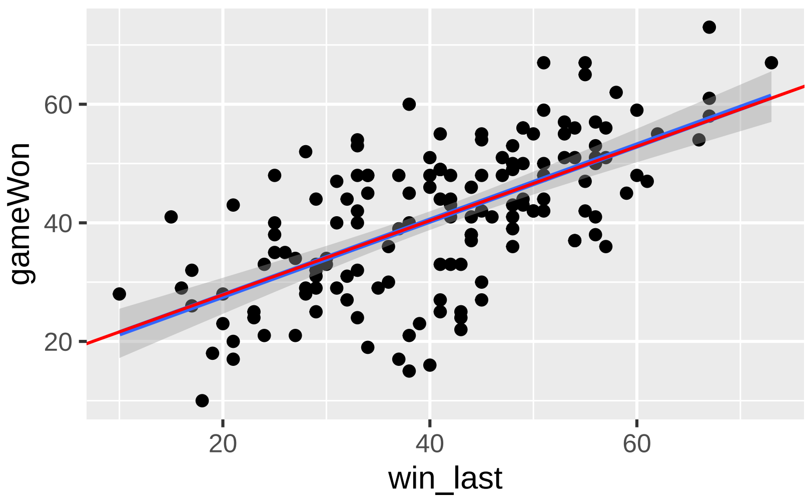
This isn't really that much lower, especially considering we used 1/6th less data to produce it compared to before. We can plot the impact of holding out a season on our data has as well.

```
win_lag_df %>% ggplot(aes(x=win_last,y=gameWon)) +geom_point() + geom_smooth(method = "lm") + geom_ablin
```

```
## ‘geom_smooth()‘ using formula ’y ~ x’
```

```
## Warning: Removed 30 rows containing non-finite values (stat_smooth).
```

```
## Warning: Removed 30 rows containing missing values (geom_point).
```



The blue line (the original slope) and the red line (the slope estimated without the last season of data) sit directly on top of each other. The actual coefficient in the second regression was 0.626 compared to 0.634 in the original.

That's great it tells us that the help out season was not shockingly different that the others, it didn't have a lot of what we call leverage changing the estimate by exerting lot's of influence. We could do an exercise like this and check every year if we wanted, it could help us to understand if these relationships are changing significantly over time, which is something we would want to know if we were going to use it to make predictions in the future.

Cross Validation

Cross-Validation is a technique that formalizes the idea of checking our predictions on data that we haven't used.

To recap in the previous section, the first goodness of fit test we did was checking the r-squared. We noted, however, that in some sense "re-used" the data we used to make our estimates of β and then using that same data to check how well we did. Remember that we chose our beta's to be the once that *minimize* the remaining noise, this is equivalent to choosing the β's which *maximize* the amount of variance of the outcome we explain. It is a little disingenuous then to pick the maximizing β's for this data set (and for the specific predictors we include in the model) and then evaluate how good of a fit it is with that *same* data.

How much of a problem this becomes is related to what our goal is. Especially in the world of prediction, this can be a large problem. The goal of a prediction model is to use it on unseen data to make predictions of the outcome. If that is our goal then we should probably evaluate it on how well the model predicts outcomes for unseen data. Intuitively hidden data should be more similar to other data we haven't seen or so we hope.

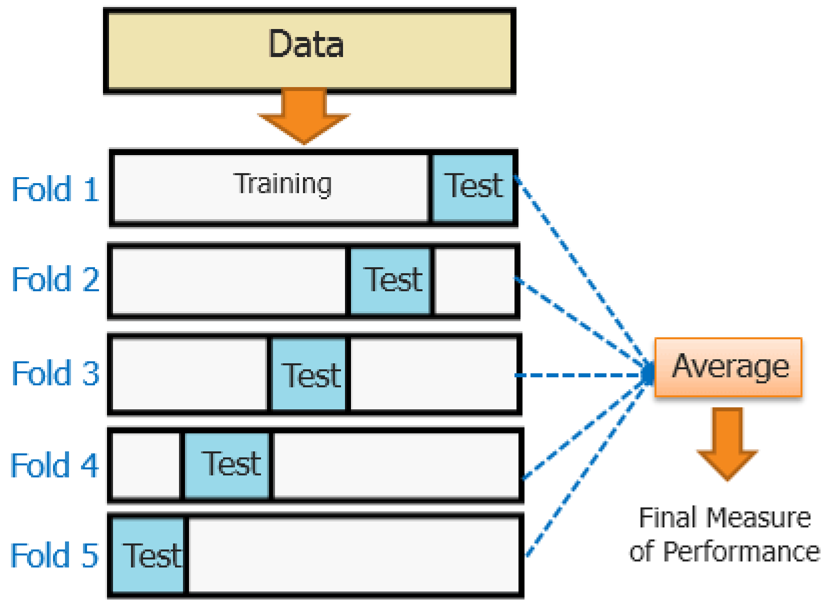
Cross-Validation is a way of estimating what we call expected generalized error. We can think of this as a number which measures the distance we are away from the data generating process which actually produced the data average over the distribution of the data.

Leave-on-out (LOO) cross validation is when we do the leave one out procedure for every data point. LOO using log-loss converges to the expected generalization error. Watanabe showed that this holds for a really large class of models including even neural networks and bayesian heirarchical models.

The idea is to randomly jumble our data into K groups. We try to make the groups roughly equal-sized.

If $K = 5$ and we have 500 rows of data, the we randomly put 100 data points into each fold. Now what we are going to do is take the first group and set it aside, it's called the test data set.

```
knitr::include_graphics("kfold.png")
```



The above graph gives you the diagram. We use the remaining 4 groups of data to run the model, then we check how well it fits the test data that we didn't use. We can use any metric we want really, but we should choose the metric carefully depending on the models we are using and the properties we want. In the linear regression case we might use something like R^2 or estimated squared error. We repeat this process for all 5 groups, setting one group aside, using the other 4 to estimate and we average our test metric across the 5 groups. We might use this to choose our predictors. For example in each iteration of the erection we might take a couple competing models (different regression types or just different variables) and compare how they do on the unseen data. We might take the best model as the one we actually use. There are of course many other things we can do but that is a reasonable option.

Cross-Validation Code

Here I am re-using some simulation code to demonstrate CV. A useful exerise to the reader would be to try and recreate this for the wins data set we were playing with earlier.

The wins data set has a slightly more complicated structure than the example below. The win data set was organized by season. In such a case we could choose to either randomly hold out observations like the code below or hold out full seasons. The two procedures may have slightly different properties. This is sometimes called block cross-validation.

```
N <- 500 # number of observations in each sample

p <- 5 # number of predictors

sigma <- 2 # how noisy is this data. The bigger this value gets the more importnat noise or luck will p
sigma_x <- c(1,2,3,1,1) # this is how noisy our predictors are
mean_x <- c(5,0,20,-6,2) # this is the average value of the predictor
```

```
beta_mat <- matrix(nrow = N, ncol =(p+1)) # this just captures the two estimated parameters from each s

alpha <- 10 # This is the true intercept
beta <- c(2,-1,.2,1.5,4) # our true beta


## Simulate a single run

  X <- rmvnorm(N,mean_x,sigma = diag(sigma_x)) #This generates the predictors

  Y <- alpha + X%*%beta + rnorm(N,0,sigma)

  X <- data.frame(X)
```

So now we pretend that we were just handed this data set. Imagine that we aren't sure if we should just include the first 4 variables or all 5. We feel very confident theoretically that the first 4 should be important (maybe from other people's work or what we understand about the process), but aren't sure about the last one.

First step is to split the data into groups. There is a package carret for this, but I will do it by hand so it's easy to see what's going on and so you can play around with the code easier.

One validation metric will be Root Mean Squared Error[RMSE], the other will be R^2

$$RMSE = \frac{1}{K}\sum_{k=1}^{K}\sqrt{\frac{1}{n_k}\sum_{i=1}^{n_k}(y_{test_i} - X_{test_i}\hat{\beta_{train}})^2}$$

We take the n_k test outcomes that we put aside y_{test} and we compute our predictions using the $\hat{\beta_{train}}$ that we estimate using the training data (the other 4 groups of data not in the test set) and the predictors from the test data set X_{test}. The prediction we can call $\hat{Y_{test}} = X_{test_i}\hat{\beta_{train}}$. Now we look how far away the square of our predictions $\hat{Y_{test}}$ are from the actual values Y_{test} are on average.

```
index <- sample.int(nrow(X),size=nrow(X),replace=FALSE) # an index, one for each data point, shuffled i

folds <- vector("list", length=5) # a list to hold the indexes, on for each fold

for(i in 1:5){
  folds[[i]] <- index[(1+(i-1)*100):(100*i)] # This creates the folds we will call later.
}
```

```
# Cross-Validation Loop,

rmse1 <- vector(length = 5) # store the rmse values from each iteration from model with all the variabl
rmse2 <- vector(length = 4) # store the rmse from each iteration with the first 4 predictors

Rsqr1 <- vector(length = 5)
Rsqr2 <- vector(length = 5)
colnames(X) <- paste0("Var_",c(1:5))

for(k in 1:5){

  test_ind <- folds[[k]]

  X_test <- X[test_ind,]
  Y_test <- Y[test_ind]

  data_test <- data.frame(Y = Y_test, X = X_test)
  names(data_test)[2:6] <- paste0("Var_",c(1:5))

  X_train <- data.frame(X[-test_ind,])
  Y_train <- Y[-test_ind]

  data_train <- data.frame(Y = Y_train, X_train)
  names(data_train)[2:6] <- paste0("Var_",c(1:5))

  mod_train1 <- lm(Y ~ .,data=data_train)
  mod_train2 <- lm(Y ~ ., data=data_train[,-6])

  Y_pred1 <- predict(mod_train1,newdata = data_test)
  Y_pred2 <- predict(mod_train2,newdata = data_test)# predictions from first model

  rmse1[k] <- sqrt(mean((Y_test - Y_pred1)^2))
  rmse2[k] <- sqrt(mean((Y_test - Y_pred2)^2))

  Rsqr1[k] <- sum((mean(Y_test) - Y_pred1)^2)/sum((Y_test - mean(Y_test))^2)
  Rsqr2[k] <- sum((mean(Y_test) - Y_pred2)^2)/sum((Y_test - mean(Y_test))^2)

}

rmse1

## [1] 1.930042 1.935442 2.000978 2.009654 2.069367

rmse2
```

```
## [1] 3.881325 4.681765 4.976808 4.893482 4.544050
```

```r
mean(rmse1)
```

```
## [1] 1.989096
```

```r
mean(rmse2)
```

```
## [1] 4.595486
```

```r
Rsqr1
```

```
## [1] 0.9691290 0.9180989 0.8340924 0.8045217 0.8286522
```

```r
Rsqr2
```

```
## [1] 0.3297088 0.2797829 0.2140348 0.2622292 0.5018590
```

```r
mean(Rsqr1)
```

```
## [1] 0.8708988
```

```r
mean(Rsqr2)
```

```
## [1] 0.3175229
```

We can see very clearly that the first model is much better by both metrics. Higher R^2 is better, lower RMSE is better. So we would reasonably include the 5th variable, it seems to greatly improve our predictions.

What about regression that's not linear (in that way anyway)

There are many other ways to do regression. Here we are going to briefly talk about regression with binary outcomes (0 or 1). Wins-losses are well represented like this. If you remember from probability, conditional expectation of binary variables are also conditional probabilities, so we will model say $P(Y = 1|X)$. Here we are going to try to predict the outcome of an individual game.

To make it interesting let's create a data set with rolling averages of some key team metrics from the most recent 20 games. I use the zoo package below to get a rolling average for number of points for and against in the most recent 20 games for each team.

```r
library(zoo)
```

```
##
## Attaching package: 'zoo'
## The following objects are masked from 'package:base':
##
##     as.Date, as.Date.numeric
```

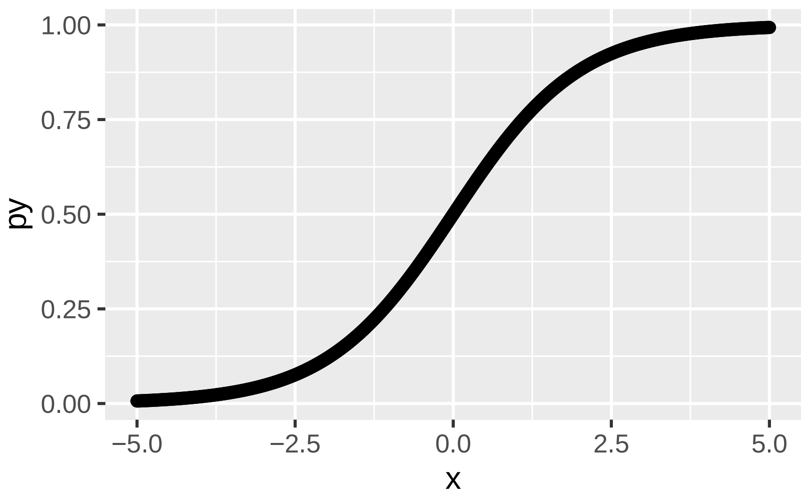
```
dt <- tm_box %>% group_by(teamAbbr,season_int) %>%
  mutate(ma_pts_for = rollmean(lag(teamPTS), k =20, fill = NA),
         ma_pts_ag = rollmean(lag(opptPTS), k =20, fill = NA),
         win = ifelse(teamPTS > opptPTS,1,0))


### We have two rows per game so the below gets rid of some of it

dt2 <- dt[!duplicated(dt[,c(1:9)]),]
```

Now we have a cool data set, we can use those pieces of information to predict a win. We will logistic regression to do this. The full treatment of logistic regression and generalized linear models is beyond this primer, but here is a tiny bit of math. Logistic regression proposes the following model for the conditional probability.

$$P(Y = 1|X;\beta) = \frac{1}{1 + \exp(-(X\beta))}$$

The function on the right is called the logistic function. It is a really convenient way to remap any real number (our predictors and coefficients $X\beta$ could be any real number) and ensure it is always between 0 and 1. In general, the function makes a nice little s-shape like the following simulation.

```
logistic_fun <- function(x){
  1/(1+exp(-x))
}

x <- seq(from = -5, to = 5, by = .02)

py <- logistic_fun(x)
dtt <- data.frame(py = py, x=x)
ggplot(data = dtt,aes(y=py,x=x)) + geom_point()
```



So this model isn't linear on the probability scale, but it is linear once we apply the inverse logit (or expit) function and we are left with $X\beta$ just like in linear regression.

Ok, let's run a real example and get predicted win probabilities

```
win_prob_reg <- glm(win ~ ma_pts_for + ma_pts_ag,data = dt2,family =binomial(link = "logit"))

dt2$win_prob1 <- rep(NA, nrow(dt2))
dt2$win_prob1[!is.na(dt2$ma_pts_for)] <- predict(win_prob_reg, type = "response")

summary(win_prob_reg)
```

```
##
## Call:
## glm(formula = win ~ ma_pts_for + ma_pts_ag, family = binomial(link = "logit"),
##     data = dt2)
##
## Deviance Residuals:
##     Min       1Q   Median       3Q      Max
## -2.0765  -0.9967  -0.6850   1.1536   2.3552
##
## Coefficients:
##              Estimate Std. Error z value Pr(>|z|)
## (Intercept) -0.049860   0.608577  -0.082    0.935
## ma_pts_for   0.133048   0.006377  20.863   <2e-16 ***
## ma_pts_ag   -0.136263   0.006818 -19.985   <2e-16 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## (Dispersion parameter for binomial family taken to be 1)
##
```

```
##     Null deviance: 7564.2  on 5576  degrees of freedom
## Residual deviance: 6950.6  on 5574  degrees of freedom
##   (1802 observations deleted due to missingness)
## AIC: 6956.6
##
## Number of Fisher Scoring iterations: 4
```

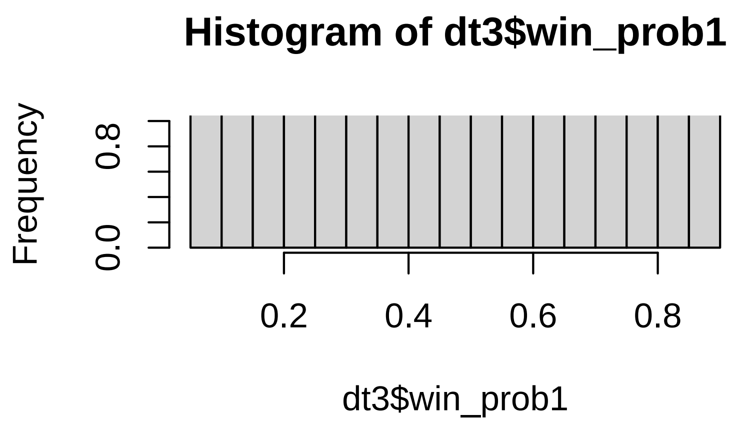
We get coefficients just like linear regression. I won't go into the interpretation here too much, but we can think about the signs. Notice the rolling points for average coefficient is positive. This tells us, all else equal, that more points in the recent past are associate with an increased chance of winning. Similarly, more points against recently is associated with a decreased chance of winning. This makes sense. There are many things to think about when diagnosing a binary model like this, for now we will focus on calibration plots.

A calibration plot asks "when our model says the home team has a 40% percent chance of winning" do they win 40% of the time?

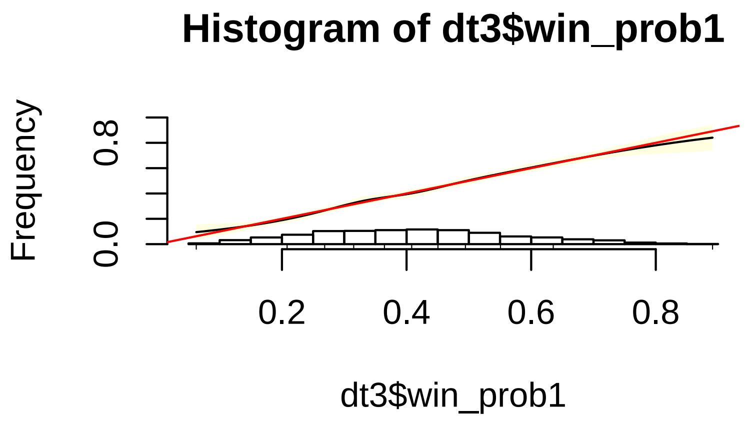
```
library(gbm)
```

```
## Loaded gbm 2.1.8
```

```
dt3 <- dt2 %>% filter(!is.na(ma_pts_for))


h <- hist(dt3$win_prob1, ylim = c(0,1))
```



```
h$counts <- h$counts/sum(h$counts)
plot(h, ylim = c(0,1))
calibrate.plot(y = dt3$win, p = dt3$win_prob1, replace = F)
```



Above is the calibration plot. Ideally we want the fitted black line as close to the 45 degree red line as possible. This looks pretty good, no huge systematic deviations. notice however at the ends the black line is a little farther away. I added a histogram on top of the plot of our predictions. Notice that at the ends we don't have so many predictions. We would hope at the very least that we are well-calibrated over the areas we have lot's of predictions. If not we have problems to address.

A good exercise for the reader would be to play with the win model. How can we make it better? Are there important covariates we are missing? How optimal is a 20 game rolling average compared to other choices?

A tiny bit more about loss functions and conditional mean

In statistics, we often find ourself in a situation where we need to make a prediction. When you want to make a prediction you need some way of determining how good that prediction is. What counts as a close prediction? What counts as a poor prediction? It is often really useful to write down a function which does this for us, that is we define a distance to penalize how wrong our predictions are. This is sometimes called a loss function.

One way to make a prediction is to optimize a loss function. Once we pick a loss function, we choose a model in some way that makes our loss or error as small as possible.

In general, we can represent a loss function as $L(Y, X)$. Where again we want to use the covariates or predictors X to get as close as possible to the actual outcome Y.

We can then write the average or expected loss as $E[L(Y, X)]$. It's just some function of X and Y so we can take expectations! Here are a few examples of loss functions.

1. Quadratic loss (or means squared error or L2 loss)

This is likely the most important or most used loss function. It is defined as follows:

$$L_{quad}(Y, X) = (Y - f(X))^2$$

So give me some function of our predictions $f(X)$ and the loss is a vector of squared distances from the actual outcome. If we want the average of this loss function we get the mean squared error. In practice the different functions X will be defined by the model we propose.

2. L1 loss or Absolute error

$$L_{abs}(Y, X) = |Y - f(X)|$$

3. Log-Loss

$$L_{log}(Y, X) = -log(P(Y|X))$$

This one is related to likelihood functions and the KL Divergence
When we estimate the expected loss, we use what is called the empirical average. Let's demonstrate with the log-loss. We want to know $E[L_{log}(Y, X)] = E[-log(P(Y|X))]$ so we take the following estimate:

$$-\frac{1}{n}\sum_{i=1}^{n} log(P(Y_i|X_i))$$

Where the $P(\cdot)$ is also replaced by an estimate from our model.
Now it turns out that when we take quadratic loss as our loss function, it turns out that the function which minimizes it is the conditional expectation!
i.e

$$\arg\min_{f(X)\in\mathcal{L}_2} E[(Y - f(X)^2)] = E[Y|X]$$

So we can think of it either way, if we care about quadratic loss we should target the conditional mean or implictly when we model the conditional mean we are minimizes quadratic loss. This is also related to the idea of projection. Remember that we can think of linear regression as a projection of Y onto linear combinations of the covariates X. In what we call $\mathcal{L}_2(P)$, a hilbert space equipped with covariance as the inner product, conditional expectation is actually the projection in this space. When the conditional expectation happens to be linear in parameters β, the linear regression becomes both the linear projection and the L2 projection, i.e the conditional mean. In general, however, we have this decomposition:

$$\begin{aligned} Y &= E[Y|X] + (Y - E[Y|X]) \\ &= E[Y|X] + \varepsilon' \\ &= X\hat{\beta} + (E[Y|X] - X\hat{\beta}) + \varepsilon' \end{aligned}$$

This can be thought of as breaking into 3 parts. The linear projection $X\hat{\beta}$, the non-linearity term $(E[Y|X] - X\hat{\beta})$ which tells us how different the actual conditional mean function is to the linear projection and the irredicible error ε'. Below I perform a few simulations to see what happens when the actual conditional mean function cannot be recovered by the linear regression.

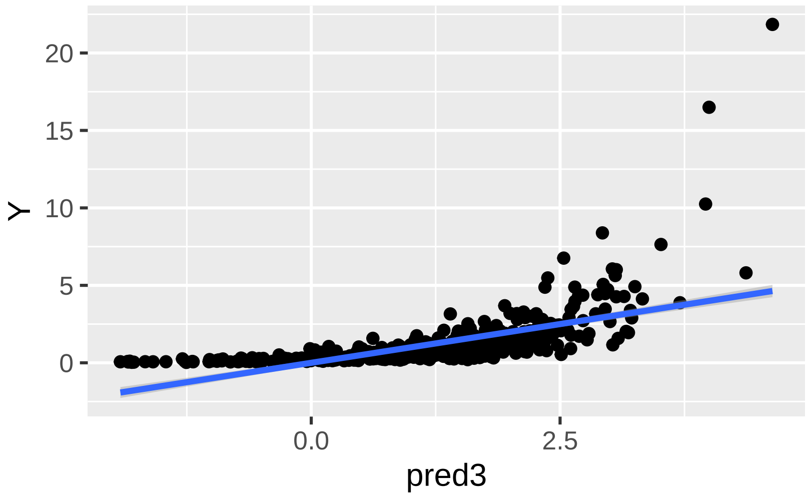
Now let's see what happens if we simulate from something that is not linear but try to fit this model anyway. Let's say that the true model is the following:

$$Y = exp(\alpha + X\beta + \epsilon)$$

In other words if we took the log of the data it would be linear again

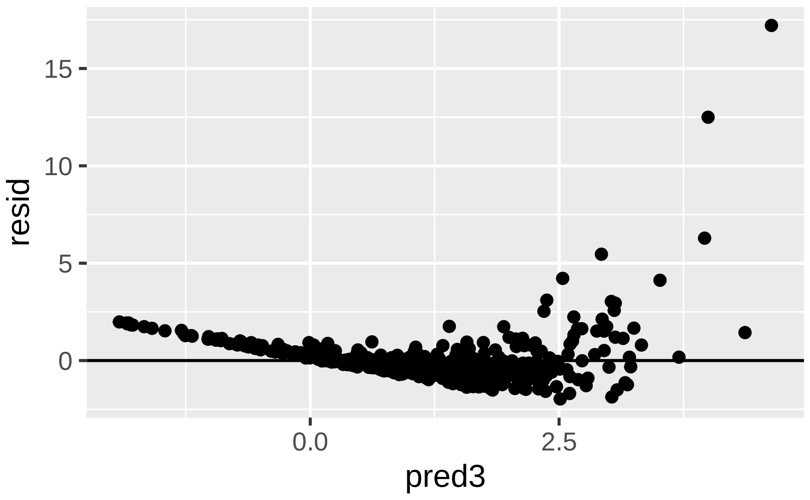
```
n <- 500 # number of observations in each sample


p <- 5 # number of predictors

sigma <- 0.5 # how noisy is this data. The bigger this value gets the more importnat noise or luck will
sigma_x <- c(1,2,1,1,1) # this is how noisy our predictors are
mean_x <- c(1,0,1,-1,1) # this is the average value of the predictor


alpha <- 0# This is the true intercept
beta <- c(.2,-.5,.2,.4,-.4) # our true beta
```

```
## Simulate a single run

  X <- rmvnorm(n,mean_x,sigma = diag(sigma_x)) #This generates the predictors

  Y <- exp(alpha + X%*%beta + rnorm(n,0,sigma)) # This generates the outcome variables

  mod3 <- lm(Y~X) # estimates beta

pred3 <- predict(mod3,data.frame(int=1,X))# predicted values using the prediction function

df3 <- data.frame(pred3,Y)
ggplot(df3,aes(x=pred3,Y)) + geom_point() + geom_smooth(method="lm")

## ‘geom_smooth()‘ using formula ’y ~ x’
```



This doesn’t look great, we can see the data does not form a straight line and there are parts where we systematically over and under-fit. A common and useful diagnostic plot is a residual plot. We plot the residuals against the predicted values

```
df3$resid <- df3$Y - df3$pred

ggplot(df3, aes(x = pred3, y =resid)) + geom_point() + geom_hline(yintercept =0)
```



In a well specified regression we should see random scatter across the entire range of the predictions, but instead here we see systematic patterns. When we see this it's a good sign there is a better model out there. If we insist on using a linear regression, we can. It still is the linear projection which may be meaningful, but notice that the accuracy depends greatly on the covariates. We would only want to trust such a projection over the areas we have lots of data for. Extrapolating these kinds of predictions will get us in serious trouble.

There is a really good paper by Buja et al that talks about how to interpret linear regression when we use the wrong model if you are interested in more.

This is really just the beginning of linear regression, but many of the key ideas are here. As theory and as a tool it is likely something you will come back to over and over again throughout your lifecycle as a data researcher, each time hopefully learning something new or appreciating it in a new light. I know that I have.