We’ve made our predictions about the past (estimating the relative strengths of teams based on past results), now we need to predict the future. I think it also nicely captures that even our predictions about the past are noisy- we can not ever truly know the exact strengths of football teams; the job of analytics is to estimate these are accurately as possible. But any noise in those past predictions will be carried forward and amplified when predicting the future.

Onward to the code, first as always, loading libraries and setting a seed for reproducibility:

library(tidyverse)

library(ggrepel)

set.seed(3459)

We’re then going to load all the stuff we prepped and predicted in the last post. Remember the α parameter below refers to a teams attacking strength (the relative number of goals they are expected to score), and the β parameter refers to the attacking strength (the inverse of the relative number of goals they are expected to concede). Finally, γ refers to the extra advantage of playing at home.

fixtures <- readRDS("../../static/files/dc\_2/dc\_fixtures.rds")

results <- readRDS("../../static/files/dc\_2/dc\_results.rds")

model <- readRDS("../../static/files/dc\_2/dc\_model.rds")

model

## $alpha

## Arsenal Blackburn\_Rovers Coventry\_City Dover\_Athletic

## 1.1106558 0.6370160 0.3023048 -0.2875353

## Enfield\_Town Frimley\_Green

## -0.3767038 -1.3857376

##

## $beta

## Arsenal Blackburn\_Rovers Coventry\_City Dover\_Athletic

## 0.6457175 0.4289270 0.3647815 -0.1362931

## Enfield\_Town Frimley\_Green

## -0.3852812 -0.9178517

##

## $gamma

## gamma

## 0.189462

We’ll define a quick function to do our prediction. For a quick explanation of exactly why it’s coded as presented, see below:’.

First, let’s load libraries and also set a seed for the reproducibility of this document

**#** munging

library(tidyverse)

**#** seed **for** reproducibility

set.seed(3459)

##Set up

In reality, we’d probably want to model a whole league or cup. However, these can generally contain 20+ teams, many of which will have similar abilities. For simplicity here, lets instead imagine a summer league between 6 English football clubs where each team plays each other twice (once at home and once away)

teams <- c("Arsenal", *# 5th in the 1st tier*

"Blackburn\_Rovers", *# 15th in 2nd tier*

"Coventry\_City", *# 8th in 3rd tier*

"Dover\_Athletic", *# 14th 5th tier*

"Enfield\_Town", *# 10th in 7th tier*

"Frimley\_Green") *# 2nd in 9th tier*

We’ve managed to arrange a league that has a nice stratification between teams, so we’d expect each to be comfortably better than the next best (which will make sanity checking our results easier). Lucky for us, the teams are also in alphabetical order of strength so in case you don’t have any prior on a team, take the first letter of it’s name (A-F).

Each week each team play one game, so we’ll have a fixture list that looks like:

head(fixtures, 8)

**#***# home away gameweek*

**#***# 1 Frimley\_Green Arsenal 1*

**#***# 2 Enfield\_Town Blackburn\_Rovers 1*

**#***# 3 Dover\_Athletic Coventry\_City 1*

**#***# 4 Arsenal Enfield\_Town 2*

**#***# 5 Frimley\_Green Dover\_Athletic 2*

**#***# 6 Blackburn\_Rovers Coventry\_City 2*

**#***# 7 Dover\_Athletic Arsenal 3*

**#***# 8 Coventry\_City Enfield\_Town 3*

Obviously for this we’re going to have to make up our data. For the code used to generate it, see the bottom of the post.

Let’s say that we’ve had 8 weeks of games played so far, and the results have been as follows

head(results,8)

**#***# home away hgoal agoal gameweek*

**#***# 1 Dover\_Athletic Coventry\_City 0 3 1*

**#***# 2 Enfield\_Town Blackburn\_Rovers 0 3 1*

**#***# 3 Frimley\_Green Arsenal 0 8 1*

**#***# 4 Arsenal Enfield\_Town 5 0 2*

**#***# 5 Blackburn\_Rovers Coventry\_City 1 1 2*

**#***# 6 Frimley\_Green Dover\_Athletic 1 2 2*

**#***# 7 Blackburn\_Rovers Frimley\_Green 6 0 3*

**#***# 8 Coventry\_City Enfield\_Town 2 1 3*

A better way to show this is to generate a matrix of home (y axis) vs. away (x axis) and show the goals scored in each match between them:

p1 <- results %>%

*# remove unplayed games*

filter(!is.na(hgoal)) %>%

ggplot(., aes(x = away, y = home, fill = hgoal-agoal)) +

geom\_tile() +

*# add the scorelines*

geom\_label(aes(label = paste(hgoal, agoal, sep = "-")), fill = "white") +

*# colour where green shows home win and red an away win*

scale\_fill\_gradient2(low = "darkred", high = "green", midpoint = 0, guide = FALSE) +

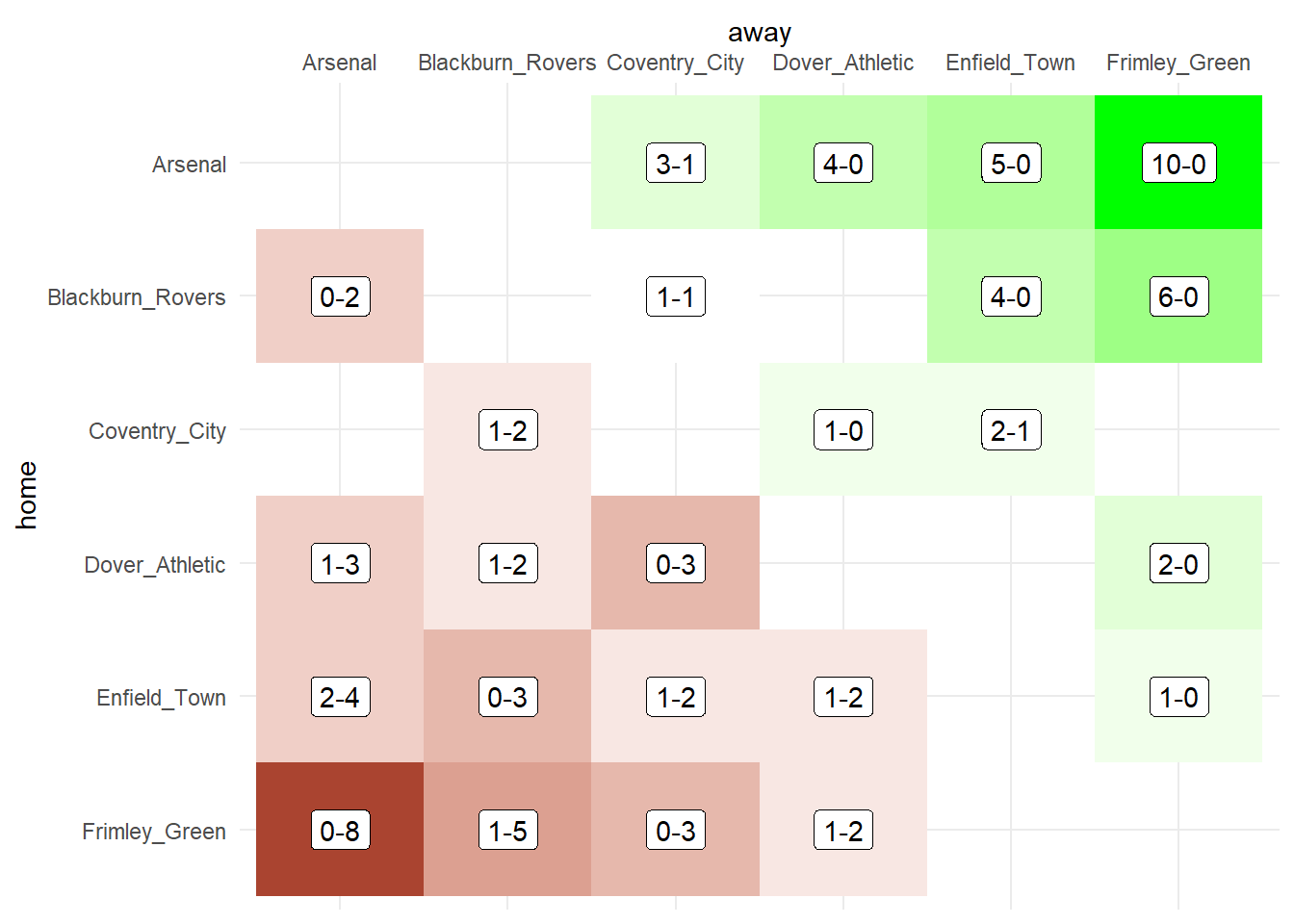
scale\_x\_discrete(limits = levels(results$home), position = "top") +

scale\_y\_discrete(limits = rev(levels(results$away))) +

theme\_minimal()

*# plot*

p1



As the colour gradient (from bottom right to top left) shows, the teams we’d expect to do better are. Given the stochastic nature of football though, there are some surprises. E.g. Blackburn only managing to draw at home to Coventry.

A good sense of teams relative abilities can be seen in the league table of results so far (assuming 3 points for a win, and 1 for a draw):

*# function to melt results*

*# returns df with team and goals for and against for each match*

melt\_results <- function(results\_df) {

results\_df %>%

*# select only relevant columns*

select(home, away, hgoal, agoal) %>%

gather(location, team, -hgoal, -agoal) %>%

*# calculate goals for/against the team*

mutate(g\_for = case\_when(

location == "home" ~ hgoal,

location == "away" ~ agoal

)) %>%

mutate(g\_ag = case\_when(

location == "home" ~ agoal,

location == "away" ~ hgoal

))

}

*# function to calculate points won and gd for each team*

results\_to\_table <- function(results\_df) {

results\_df %>%

*# use above melting function*

melt\_results(.) %>%

*# 3 points for a win, 1 for a draw*

mutate(points = case\_when(

g\_for > g\_ag ~ 3,

g\_ag > g\_for ~ 0,

g\_for == g\_ag ~ 1

)) %>%

*# calculate goal difference for each match*

mutate(gd = g\_for - g\_ag) %>%

group\_by(team) %>%

*# get the final statistics per team*

summarise(games\_played = n(),

gf = sum(g\_for),

ga = sum(g\_ag),

gd = sum(gd),

points = sum(points)) %>%

arrange(-points, -gd, -gf)

}

*# calculate league table for our played fixtures*

league\_table <- results %>%

filter(!is.na(hgoal)) %>%

select(-gameweek) %>%

results\_to\_table(.) %>%

print()

**#***# # A tibble: 6 x 6*

**#***# team games\_played gf ga gd points*

**#***# <chr> <int> <dbl> <dbl> <dbl> <dbl>*

**#***# 1 Arsenal 8 39 4 35 24*

**#***# 2 Blackburn\_Rovers 8 23 6 17 19*

**#***# 3 Coventry\_City 8 14 8 6 16*

**#***# 4 Dover\_Athletic 8 8 15 -7 9*

**#***# 5 Enfield\_Town 8 6 22 -16 3*

**#***# 6 Frimley\_Green 8 2 37 -35 0*

Where teams positions are nicely rank ordered (the data for this example is fairly curated so it’s not that surprising).

##Predictions

With two rounds to go, there’s still 6 fixtures we might want to predict (to try and judge which team will end up where, or just to bet on the remaining games).

This are:

*# get the yet to be played matches*

unplayed\_games <- fixtures %>%

filter(gameweek > 8) %>%

print()

**#***# home away gameweek*

**#***# 1 Coventry\_City Arsenal 9*

**#***# 2 Blackburn\_Rovers Dover\_Athletic 9*

**#***# 3 Frimley\_Green Enfield\_Town 9*

**#***# 4 Arsenal Blackburn\_Rovers 10*

**#***# 5 Coventry\_City Frimley\_Green 10*

**#***# 6 Dover\_Athletic Enfield\_Town 10*

If we want to predict these results, we need to have data on the strength of the teams above, but also, a good prior on what sort of scores we should expect.

Using real data from the engsoccerdata package we can get the results of all 48840 English football league games between August 1992 and May 2016. If we melt this to get the goals scored by each team by their location we get a data.frame of 97680 records of a teams performance in a game:

*# load real data from the english league*

real\_data <- engsoccerdata::england %>%

*# filter out 'premier league era' matches*

filter(Season > 1991) %>%

*# select only relevant columns*

select(home, away = visitor, hgoal, agoal = vgoal) %>%

*# munge*

melt\_results() %>%

select(-hgoal, -agoal) %>%

mutate(data = "real")

head(real\_data)

**#***# location team g\_for g\_ag data*

**#***# 1 home Arsenal 2 4 real*

**#***# 2 home Chelsea 1 1 real*

**#***# 3 home Coventry City 2 1 real*

**#***# 4 home Crystal Palace 3 3 real*

**#***# 5 home Everton 1 1 real*

**#***# 6 home Ipswich Town 1 1 real*

Here every row shows a team that played a match (as it’s sorted by league then alphabetically, the first 6 records are all for Arsenal). It also shows if the team played home or away. The data also shows the goals scored by (e.g.) Arsenal in g\_for, and the goals they conceded in g\_ag.

If we plot the goals scored for each game, we get a nice humped distribution with slightly offset peaks for home and away. That is to say, in most games teams will score 0, 1, or 2 goals, and that scoring more than 6 goals in a match is incredibly rare. The difference between the home and away distributions mean that teams are slightly more likely to score more if playing at home, compared to play away from home.

**# plot goals scored home/away for real english football matches**

p2 <- real\_data %>%

ggplot(., aes(x = g\_for, fill = location)) +

**# smooth densities**

geom\_density(adjust = 8, alpha = 0.5) +

scale\_fill\_manual(values = c("red", "blue")) +

scale\_x\_continuous(breaks = 0:6) +

labs(title = "Goals scored at home and away in English football",

subtitle = "data from 48.8k matches 1992-2016",

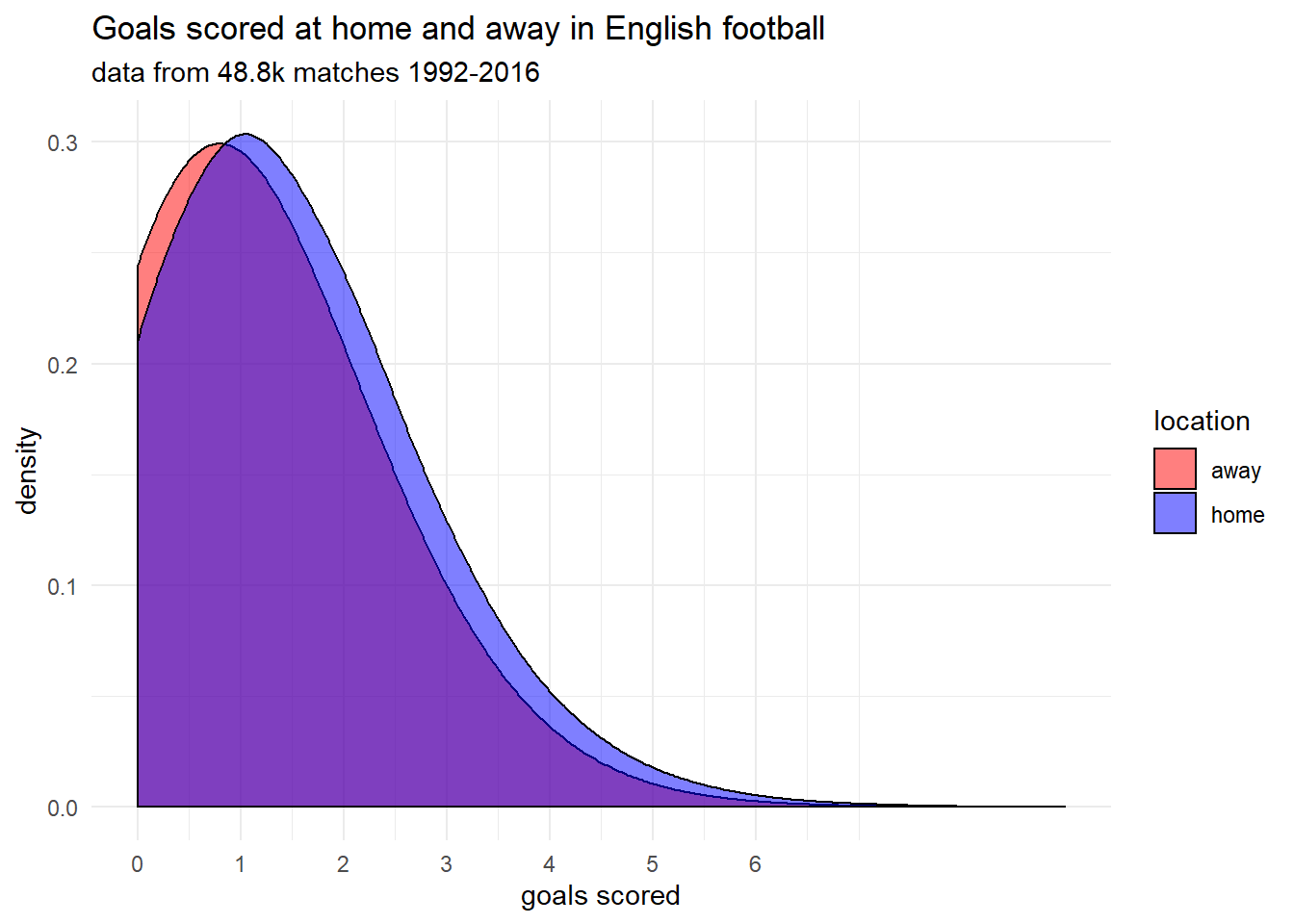
x = "goals scored",

y = "density") +

theme\_minimal()

**# plot**

p2



We can work out what the average difference between playing at home and away is by taking the means of goals scored at home, and when playing away:

*# calculate mean home and away goals*

real\_data\_means <- real\_data %>%

group\_by(location) %>%

summarise(mean\_scored = mean(g\_for)) %>%

print()

**#***# # A tibble: 2 x 2*

**#***# location mean\_scored*

**#***# <chr> <dbl>*

**#***# 1 away 1.12*

**#***# 2 home 1.47*

Goals in games are both relatively sparse, and relatively stochastic; football is a low scoring game where goals are evenly distributed throughout the game. In theory any attack made by a team i has a probability of being scored dependent upon the strength of team i’s attack (αi) which is independent of all the other attacks that team has made.

(there is some reason to doubt this may be the case3, but for now this is a fine generalisation)

By grouping all teams together into “home” and “away” categories (in a league setting each team will play each other home and away so this should average out) and taking the average number of goals scored per match as the Poisson mean (λ) we can see how well our above graph fits a simulated Poisson process.

*# generate Poisson distributed vector with mean = real world mean*

simulated\_poisson <- real\_data\_means %>%

**split**(f = .$location) %>%

lapply(., function(**x**) df = data.frame(dist = rpois(100000, **x**$mean\_scored),

location = **x**$location)) %>%

*# map it all together and label*

map\_df(I) %>%

mutate(data = "simulated")

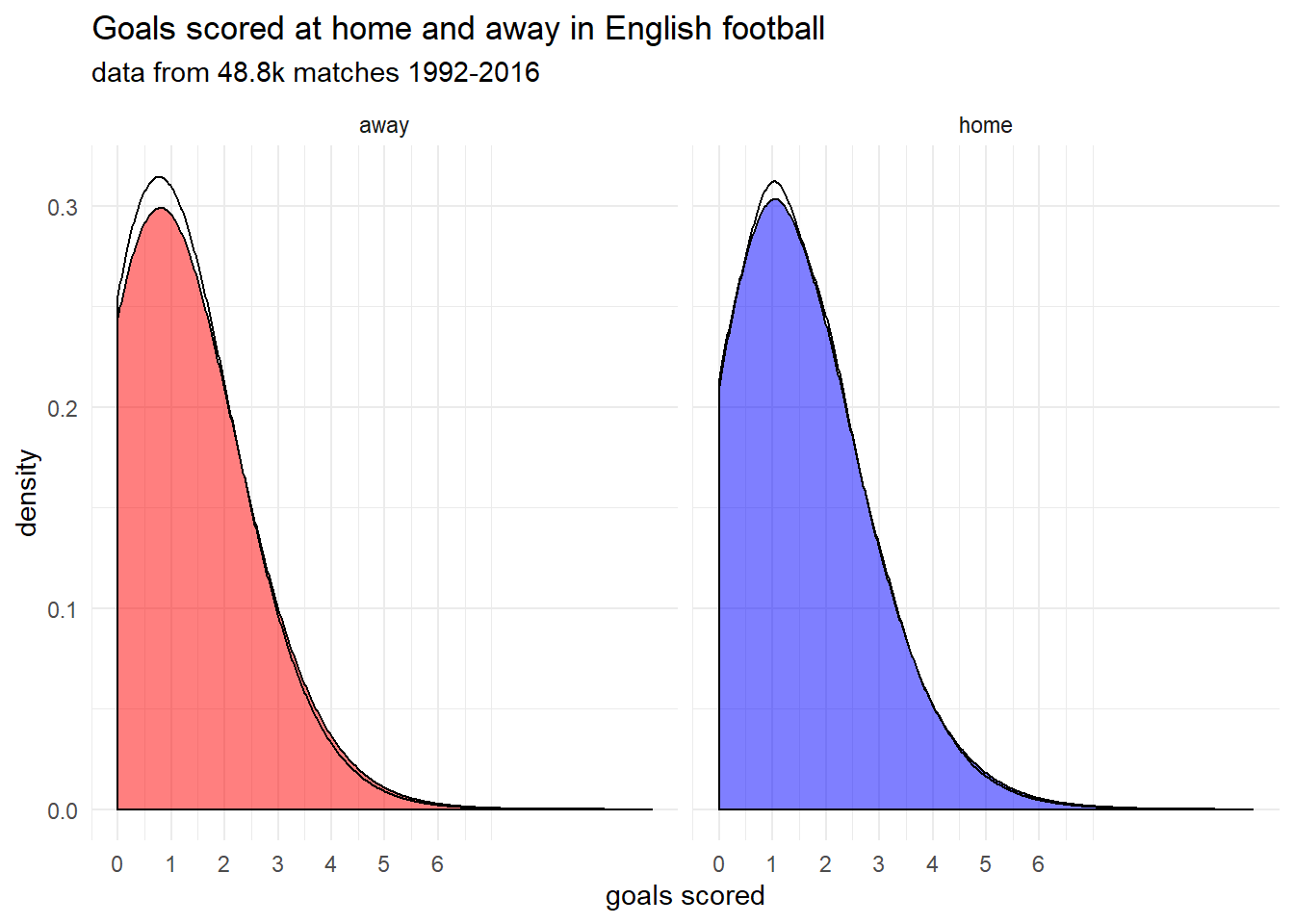
*# add these distributions to the plot*

p2 + geom\_density(data = simulated\_poisson, aes(**x** = dist),

fill = NA, adjust = 8, alpha = 0.2) +

scale\_fill\_manual(**values** = c("red", "blue"), guide = FALSE) +

facet\_wrap(~location)



It’s not perfect, but it’s not a bad fit either. In actuality, a Chi-squared test will show that goals scored *does not* follow a Poisson distribution given the number of matches we have as data. But for the sake of this post, put that out of mind.

If we think that goals scored represents some Poisson process, it can be modeled using the equation which underlies the Poisson distribution. For a given interval (one match), the probability of x events (goals scored) in that interval will be:

P(x)=λxe−λx!P(x)=λxe−λx!

The simplest model we can produce is to estimate λ as each team’s attack rating (henceforth αi) which is equal to observed mean rate of goals for that team.

That is the say the probability of team i scoring x goals against team j is:

P(xi,j=x)=αxie−αix!P(xi,j=x)=αixe−αix!

where αi is the sum of all goals scored divided by the total number of matches:

αi=1NN∑n=1xαi=1N∑n=1Nx

grouping by teams makes this easy to calculate:

basic\_model <- results %>%

melt\_results() %>%

group\_by(team) %>%

*# we'll use the goals scored to model the attack*

*# and goals conceeded to measure defence rating*

summarise(alpha = mean(g\_for),

beta = mean(g\_ag)) %>%

print()

**#***# # A tibble: 6 x 3*

**#***# team alpha beta*

**#***# <chr> <dbl> <dbl>*

**#***# 1 Arsenal 4.88 0.5*

**#***# 2 Blackburn\_Rovers 2.88 0.75*

**#***# 3 Coventry\_City 1.75 1*

**#***# 4 Dover\_Athletic 1 1.88*

**#***# 5 Enfield\_Town 0.75 2.75*

**#***# 6 Frimley\_Green 0.25 4.62*

(we’ll come on to the beta parameter in a bit- where alpha is the average scoring rate, beta is the average conceding rate).

If we take Coventry’s remaining two games as examples we can see that they are yet to play Arsenal and Frimley Green at home

coventry\_games <- unplayed\_games %>%

*# filter out Coventry City's remaining fixtures*

filter(grepl("Coventry\_City", home)) %>%

print()

**#***# home away gameweek*

**#***# 1 Coventry\_City Arsenal 9*

**#***# 2 Coventry\_City Frimley\_Green 10*

And we can take the attack rating (α) of each team and use it to estimate the results

*# get the attack ratings of all teams*

team\_alphas <- basic\_model$alpha %>% `names<-`(basic\_model$team)

*# assume goals scored for each team will be it's attack rating*

e\_results <- paste(team\_alphas[coventry\_games$home],

team\_alphas[coventry\_games$away],

sep = "-") %>%

*# name each match with the teams competing*

`names<-`(c(paste(coventry\_games$home, coventry\_games$away, sep = "-"))) %>%

**print**()

**#***# Coventry\_City-Arsenal Coventry\_City-Frimley\_Green*

**#***# "1.75-4.875" "1.75-0.25"*

These aren’t ridiculous estimates by any stretch but it’s clear something is up. It’s pretty intuitive that Coventry City would be expected to score more goals at home to Frimley Green than at home to Arsenal.

We can account for this by introducing an opposing team defence parameter βj. In our very simple model this will be estimating by taking the average rate a team concedes goals. As with the attack rating, this is the calculated as the sum of all goals conceded divided by number of matches. We’ll then multiply αi and βj together to get the score estimate:

*# get and name the defence rating for each team*

team\_betas <- basic\_model$beta %>% `names<-`(basic\_model$team)

*# assume the goals scored will be the attack rating of the team times*

*# the defence rating of it's opponent*

e\_results <- paste(round(team\_alphas[coventry\_games$home]\*

team\_betas[coventry\_games$away], 3),

round(team\_alphas[coventry\_games$away]\*

team\_betas[coventry\_games$home], 3),

sep = "-") %>%

`names<-`(c(paste(coventry\_games$home, coventry\_games$away, sep = "-"))) %>%

**print**()

**#***# Coventry\_City-Arsenal Coventry\_City-Frimley\_Green*

**#***# "0.875-4.875" "8.094-0.25"*

The opposition scores remain the same because Coventry have on average conceded 1 goal per game.

Coventry’s predicted goals though has diverged with them now predicted to score less than a goal against Arsenal and to score 8(!) against Frimley Green, both of which sound reasonable (when you consider that Frimley Green are a team of amateurs).

However, we’re also missing one final piece of the model we’ll finish with today. Recall modelling the English football data from 1992 onwards, we were left with a difference between the home scoring rate and the away scoring rate.

**#** reprint what we calculated earlier

real\_data\_means

**#***# # A tibble: 2 x 2*

**#***# location mean\_scored*

**#***# <chr> <dbl>*

**#***# 1 away 1.12*

**#***# 2 home 1.47*

It’s pretty common knowledge that football teams do better at home, so we’ll want to factor that in. A simple estimate is to divide the mean home goals/game by the mean away goals/game.

We’ll call this parameter γ and can be formalised as the sum of home goals (which we’ll refer to as x from now on) divided by the sum of away goals (y)

γ=∑x∑yγ=∑x∑y

*# the home advantage is how much easier it is to score at home*

home\_advantage\_gamma <- sum(results$hgoal) / sum(results$agoal)

e\_results <- paste(round(team\_alphas[coventry\_games$home]\*

team\_betas[coventry\_games$away] \*

*# add in home advantage for home team*

home\_advantage\_gamma, 3),

round(team\_alphas[coventry\_games$away]\*

team\_betas[coventry\_games$home], 3),

sep = "-") %>%

`names<-`(c(paste(coventry\_games$home, coventry\_games$away, sep = "-"))) %>%

**print**()

**#***# Coventry\_City-Arsenal Coventry\_City-Frimley\_Green*

**#***# "0.955-4.875" "8.83-0.25"*

Which tilts the scales a little towards Coventry’s favour but (as we’d expect- home advantage can only go so far) doesn’t affect the results too much.

Now we have a method to predict matches, we can use this on the remaining 6 nice and easily:

*# simplify to just gamma*

gamma <- home\_advantage\_gamma

*# wrap the above into a function for home and away teams*

predict\_results <- **function**(home, away, parameters) {

e\_goals\_home <- parameters$alpha[home]\*parameters$beta[away] \* gamma

e\_goals\_away <- parameters$alpha[away]\*parameters$beta[home]

*# output a df of expected goals for home and away teams*

df <- data.frame(home = home, away = away,

e\_hgoal = e\_goals\_home, e\_agoal = e\_goals\_away)

**return**(df)

}

*# convert the basic\_model df into a list with $attack and $defence parameters*

*# for each team*

basic\_parameters <- basic\_model %>%

*# rename scored/conceeded to attack/defence*

select(-team) %>%

*# convert to a list and name each element*

**as**.**list**() %>%

lapply(., **function**(x){names(x) <- teams;**return**(x)})

*# predict results using the function defined above and the list of parameters*

*# could use e.g. mapply here but I prefer the map2 grammar*

*# run the predict results function over each game consisting of $home and $away*

predicted\_fixtures <- map2\_df(unplayed\_games$home, unplayed\_games$away,

predict\_results,

*# parameters forms an extra argument that does not vary*

basic\_parameters) %>%

*# round the outputs*

mutate\_if(is.numeric, round, digits = 2) %>%

**print**()

**#***# home away e\_hgoal e\_agoal*

**#***# 1 Coventry\_City Arsenal 0.95 4.88*

**#***# 2 Blackburn\_Rovers Dover\_Athletic 5.88 0.75*

**#***# 3 Frimley\_Green Enfield\_Town 0.75 3.47*

**#***# 4 Arsenal Blackburn\_Rovers 3.99 1.44*

**#***# 5 Coventry\_City Frimley\_Green 8.83 0.25*

**#***# 6 Dover\_Athletic Enfield\_Town 3.00 1.41*

All of which look reasonable, if maybe a little bullish on the ‘better’ teams prospects.

However, while this is good for back of the envelope predictions, we know that this is a very basic model. If we want to improve it, first we must quantify how good it is.

In order to do this we can use the results we have from the first 8 weeks of matches as training data. We know what the ‘correct’ scores are for these matches, so if our model is good, it will predict similar scores to those observed.

Remember that for the Poisson distribution, the probability of x goals in one match is

P(x)=λxe−λx!P(x)=λxe−λx!

The expected value of the Poisson distribution is equal to λ, so we can plug λ as our predicted goals, and x as the actual goals, and calculate the probability of that results occurring *given* the attack/defence/home advantage parameters that we think are correct.

We then do this for all the matches played and get the likelihood for the home and away teams scores given the model:

*# 'predict' the already played matches using our function*

predicted\_results <- map2\_df(results$home, results$away,

predict\_results,

basic\_parameters) %>%

mutate\_if(is.numeric, round, digits = 2) %>%

print()

**#***# home away e\_hgoal e\_agoal*

**#***# 1 Dover\_Athletic Coventry\_City 1.09 3.28*

**#***# 2 Enfield\_Town Blackburn\_Rovers 0.61 7.91*

**#***# 3 Frimley\_Green Arsenal 0.14 22.55*

**#***# 4 Arsenal Enfield\_Town 14.62 0.38*

**#***# 5 Blackburn\_Rovers Coventry\_City 3.14 1.31*

**#***# 6 Frimley\_Green Dover\_Athletic 0.51 4.62*

**#***# 7 Blackburn\_Rovers Frimley\_Green 14.51 0.19*

**#***# 8 Coventry\_City Enfield\_Town 5.25 0.75*

**#***# 9 Dover\_Athletic Arsenal 0.55 9.14*

**#***# 10 Arsenal Coventry\_City 5.32 0.88*

**#***# 11 Dover\_Athletic Blackburn\_Rovers 0.82 5.39*

**#***# 12 Enfield\_Town Frimley\_Green 3.78 0.69*

**#***# 13 Blackburn\_Rovers Arsenal 1.57 3.66*

**#***# 14 Enfield\_Town Dover\_Athletic 1.53 2.75*

**#***# 15 Frimley\_Green Coventry\_City 0.27 8.09*

**#***# 16 Arsenal Frimley\_Green 24.60 0.12*

**#***# 17 Blackburn\_Rovers Enfield\_Town 8.62 0.56*

**#***# 18 Coventry\_City Dover\_Athletic 3.58 1.00*

**#***# 19 Coventry\_City Blackburn\_Rovers 1.43 2.88*

**#***# 20 Dover\_Athletic Frimley\_Green 5.05 0.47*

**#***# 21 Enfield\_Town Arsenal 0.41 13.41*

**#***# 22 Arsenal Dover\_Athletic 9.97 0.50*

**#***# 23 Enfield\_Town Coventry\_City 0.82 4.81*

**#***# 24 Frimley\_Green Blackburn\_Rovers 0.20 13.30*

*# calculate the likelihood of each home/away team actually scoring that many goals*

*# given the parameters for attack/defence supplied*

likelihoods <- data.frame(lik\_hgoal = dpois(results$hgoal,

predicted\_results$e\_hgoal),

lik\_agoal = dpois(results$agoal,

predicted\_results$e\_agoal)) %>%

*# round the probabilities*

mutate\_all(round, 4) %>%

*# bind likelihoods to results*

cbind(results, . ) %>%

*# bind in predictions*

left\_join(., predicted\_results, by = c("home", "away")) %>%

*# select useful parameters*

select(home, away, hgoal, e\_hgoal, lik\_hgoal, agoal, e\_agoal, lik\_agoal) %>%

print()

**#***# home away hgoal e\_hgoal lik\_hgoal agoal e\_agoal*

**#***# 1 Dover\_Athletic Coventry\_City 0 1.09 0.3362 3 3.28*

**#***# 2 Enfield\_Town Blackburn\_Rovers 0 0.61 0.5434 3 7.91*

**#***# 3 Frimley\_Green Arsenal 0 0.14 0.8694 8 22.55*

**#***# 4 Arsenal Enfield\_Town 5 14.62 0.0025 0 0.38*

**#***# 5 Blackburn\_Rovers Coventry\_City 1 3.14 0.1359 1 1.31*

**#***# 6 Frimley\_Green Dover\_Athletic 1 0.51 0.3063 2 4.62*

**#***# 7 Blackburn\_Rovers Frimley\_Green 6 14.51 0.0065 0 0.19*

**#***# 8 Coventry\_City Enfield\_Town 2 5.25 0.0723 1 0.75*

**#***# 9 Dover\_Athletic Arsenal 1 0.55 0.3173 3 9.14*

**#***# 10 Arsenal Coventry\_City 3 5.32 0.1228 1 0.88*

**#***# 11 Dover\_Athletic Blackburn\_Rovers 1 0.82 0.3612 2 5.39*

**#***# 12 Enfield\_Town Frimley\_Green 1 3.78 0.0863 0 0.69*

**#***# 13 Blackburn\_Rovers Arsenal 0 1.57 0.2080 2 3.66*

**#***# 14 Enfield\_Town Dover\_Athletic 1 1.53 0.3313 2 2.75*

**#***# 15 Frimley\_Green Coventry\_City 0 0.27 0.7634 3 8.09*

**#***# 16 Arsenal Frimley\_Green 10 24.60 0.0005 0 0.12*

**#***# 17 Blackburn\_Rovers Enfield\_Town 4 8.62 0.0415 0 0.56*

**#***# 18 Coventry\_City Dover\_Athletic 1 3.58 0.0998 0 1.00*

**#***# 19 Coventry\_City Blackburn\_Rovers 1 1.43 0.3422 2 2.88*

**#***# 20 Dover\_Athletic Frimley\_Green 2 5.05 0.0817 0 0.47*

**#***# 21 Enfield\_Town Arsenal 2 0.41 0.0558 4 13.41*

**#***# 22 Arsenal Dover\_Athletic 4 9.97 0.0193 0 0.50*

**#***# 23 Enfield\_Town Coventry\_City 1 0.82 0.3612 2 4.81*

**#***# 24 Frimley\_Green Blackburn\_Rovers 1 0.20 0.1637 5 13.30*

**#***# lik\_agoal*

**#***# 1 0.2213*

**#***# 2 0.0303*

**#***# 3 0.0003*

**#***# 4 0.6839*

**#***# 5 0.3535*

**#***# 6 0.1052*

**#***# 7 0.8270*

**#***# 8 0.3543*

**#***# 9 0.0137*

**#***# 10 0.3650*

**#***# 11 0.0663*

**#***# 12 0.5016*

**#***# 13 0.1724*

**#***# 14 0.2417*

**#***# 15 0.0271*

**#***# 16 0.8869*

**#***# 17 0.5712*

**#***# 18 0.3679*

**#***# 19 0.2328*

**#***# 20 0.6250*

**#***# 21 0.0020*

**#***# 22 0.6065*

**#***# 23 0.0943*

**#***# 24 0.0058*

If we sum the log of those likelihood values we get a measure of how wrong overall our predictions are:

log\_likehood <- sum(log(likelihoods$lik\_hgoal), log(likelihoods$lik\_agoal)) \* -1

log\_likehood

**#***# [1] 105.995*

(n.b. there will be some rounding errors- especially on the pre-log probabilities, but this will suffice for now)

To get an idea of whether or not this is good, let’s quickly run the model with all the parameters set to zero. Given that we’re pretty sure that at least Arsenal will be a lot better than Frimley Green, this model should do worse than our basic model above.

If it indeed does fit the results worse we will get a greater error term- the log likelihood sum

*# do the same but set each teams attack and defence to 1*

*# expect model to be worse as assumes all teams are equal*

equal\_parameters <- list(

alpha = rep(1, **length**(teams)) %>% `names<-`(teams),

beta = rep(1, **length**(teams)) %>% `names<-`(teams)

)

*# predict results and munge through to find sum of log likelihoods*

worse\_log\_likelihood <- map2\_df(results$home, results$away,

predict\_results,

equal\_parameters) %>%

mutate\_if(is.numeric, round, digits = 2) %>%

*# take the log probability straight away this time*

mutate(lik\_hgoal = dpois(results$hgoal, e\_hgoal, **log** = TRUE),

lik\_agoal = dpois(results$agoal, e\_agoal, **log** = TRUE)) %>%

**select**(lik\_hgoal, lik\_agoal) %>%

map\_dbl(sum) %>%

sum(.) \* -1

worse\_log\_likelihood

**#***# [1] 112.618*

The worse log likelihood (112.6) is worse (only a bit though) than the 106.0 we previously. This suggests that either the teams are actually quite equal, or that our basic model wasn’t all that good.

Parameter Optimisation

There will exist some parameters (α and β for each team, and γ for the home field advantage) that will minimise this negative log likelihood. That is to say, they will predict the results of the already played games most accurately.

If we want to find those we can use the optim() function in the stats package. This will take a vector of parameters and iterate while slightly changing their values until it gets the lowest value it can find as the output for a supplied function. It also takes a data.frame of results between teams. The results of these games are predicted and then checked against this actually observed data.

At the end, I’ve also set the function to pass some information from each iteration into the global environment, namely, the iteration number (i), the parameter values the optim() function has chosen for this iteration, and the negative log likelihood of those parameters- the likelihood of the observed scores if those parameters are correct.

optimise\_params <- **function**(parameters, results) {

*# form the parameters back into a list*

*# parameters names alpha (attack), beta (defense), and gamma (hfa)*

param\_list <- relist\_params(parameters)

*# predict the expected results for the games that have been played*

e\_results <- map2\_df(results$home, results$away,

predict\_results,

param\_list)

*# calculate the negative log likelihood of those predictions*

*# given the parameters how likely are those scores*

neg\_log\_likelihood <- calculate\_log\_likelihood(results, e\_results)

*# capture the parameters and likelihood at each loop*

*# only do it if i is initialised*

**if**(exists("i")) {

i <<- i + 1

current\_parameters[[i]] <<- parameters

current\_nll[[i]] <<- neg\_log\_likelihood

}

*# return the value to be minimised*

*# in this case the negative log likelihood*

**return**(neg\_log\_likelihood)

}

The three separate functions are coded out separately so we can tinker with them shortly:

1. to predict our results we have been supplying a list of two elements: alpha and beta, each of which are numeric vectors. optim() can only take one vector to optimise over but we can trick it by supplying unlist(list\_of\_parameters). If we do this we then first want to convert this unlisted numeric vector back into our two element list\*

\*it isn’t vital to have the parameters arranged like this, but I think it leads to neater indexing when predicting the results

1. we then need to use these parameters to predict the results of past games. For each home and away team in a data.frame of results we can predict the expected home and expected away goals. These are then bound into a data.frame of home and away teams and these predicted goals for each
2. finally, we need to calculate the negative log likelihood by calculating the log probability of the observed goals given the predicted goals and summing these. We then multiply this by -1 as the sum of the log probabilities will be negative and we want to minimise this number as close to zero as possible. The transformation of prod(neg\_log\_likelihood, -1) is a quick hack for this4

Hopefully this is at least bearable to follow. Formalised, this can be written for teams i and matches k as:

L(αi,βi,γ;i=1...n)=K∏k=1λxkke−λkxk!μykke−μkyk!L(αi,βi,γ;i=1...n)=∏k=1Kλkxke−λkxk!μkyke−μkyk!

where for match k and teams i and j, home goals, x is defined by

xk∼Poisson(λk=αi(k)βj(k)γ)xk∼Poisson(λk=αi(k)βj(k)γ)and away goals, y

yk∼Poisson(μk=αj(k)βi(k))yk∼Poisson(μk=αj(k)βi(k))

which seems daunting when you write it down, but we’ve already covered everything we need to do solve it. It’s just saying we want to minimise the result of the multiplication (the sum of logs in our case above) of the probability of scoring x and y goals in a game. The probability of goals scored assumed to be Poisson distributed, controlled by parameters α, β, and γ for home and away teams.

*# optim requires parameters to be supplied as a vector*

*# we'll unlist the parameters then relist in the function*

relist\_params <- **function**(parameters) {

parameter\_list <- **list**(

*# alpha = attack rating*

alpha = parameters %>%

.[grepl("alpha", names(.))] %>%

`names<-`(teams),

*# beta = defence rating*

beta = parameters %>%

.[grepl("beta", names(.))] %>%

`names<-`(teams),

*# gamma = home field advantage*

gamma = parameters["gamma"]

)

**return**(parameter\_list)

}

*# use these parameters to predict results for supplied matches*

predict\_results <- **function**(home, away, param\_list) {

*# expected home goals*

e\_goals\_home <- param\_list$alpha[home] \* param\_list$beta[away] \* param\_list$gamma

*# expected away goals*

e\_goals\_away <- (param\_list$alpha[away] \* param\_list$beta[home])

*# bind to df*

df <- data.frame(home = home, away = away,

e\_hgoal = e\_goals\_home, e\_agoal = e\_goals\_away)

**return**(df)

}

*# calculate the log likelihood of predict results vs supplied results*

calculate\_log\_likelihood <- **function**(results, e\_results) {

home\_likelihoods = dpois(results$hgoal, lambda = e\_results$e\_hgoal, log = **TRUE**)

away\_likelihoods = dpois(results$agoal, lambda = e\_results$e\_agoal, log = **TRUE**)

*# sum log likelihood and multiply by -1 so we're minimising neg log likelihood*

likelihood\_sum <- sum(home\_likelihoods, away\_likelihoods)

neg\_log\_likelihood <- prod(likelihood\_sum, -1)

**return**(neg\_log\_likelihood)

}

We’ll supply parameters that are all equal to 1 to optim to stop it falling into local minima that might affect the ‘optimal’ parameters it finds. The unlisted parameters are then supplied to optim along with the optimise\_parameters() function.

**#** start with all parameters equal

equal\_parameters <- list(

alpha = rep(1, length(teams)) %>% `names<-`(teams),

beta = rep(1, length(teams)) %>% `names<-`(teams),

gamma = 1

)

**#** run optim over the functions with these initial parameters

optimised\_parameters <- optim(

**#** the equal initial parameters

par = unlist(equal\_parameters),

**#** run over the **function** to optimise parameters

fn = optimise\_params,

**#** extra arguments to **function**

results = results,

**#** Nelder-Mead equation with 10k iterations max

method = "Nelder-Mead",

control = list(maxit = 10000)

)

We can take the $par element of the output of this to find the parameters for which the negative log likelihood is minimised

**#** display the parameters found to minimise

**#** the negative log likelihood

**optimised\_parameters$**par

**#***# alpha.Arsenal alpha.Blackburn\_Rovers alpha.Coventry\_City*

**#***# 2.9858302 1.8014838 1.2995271*

**#***# alpha.Dover\_Athletic alpha.Enfield\_Town alpha.Frimley\_Green*

**#***# 0.8192267 0.7762002 0.2748448*

**#***# beta.Arsenal beta.Blackburn\_Rovers beta.Coventry\_City*

**#***# 0.4738011 0.6346112 0.7503864*

**#***# beta.Dover\_Athletic beta.Enfield\_Town beta.Frimley\_Green*

**#***# 1.2208768 1.5180931 2.5535961*

**#***# gamma*

**#***# 1.1663125*

As expected, alpha decreases as teams get worse, and beta increases. The found gamma (1.166) is only marginally higher than the 1.091 for our simple model.

The $value element gives the negative log likelihood calculated for these parameters

**optimised\_parameters$**value

**#***# [1] 57.5175*

Which is much smaller than the ~100 we got from our very basic model.

Tinkering

This is all very well but there’s still some small improvements we can make.

For starters, I always think it’s simpler to have both scales of α and β to increase as a teams becomes more skillful in attack or defence. In our original equation the expected home and away goals follow the formula

xij∼Poisson(αiβjγ)xij∼Poisson(αiβjγ)yij∼Poisson(αjβi)yij∼Poisson(αjβi)

if instead of multiplying by β, we divide instead, a stronger defence will reduce the value of xij/yij (reducing the number of expected goals for the opposing team).

xij∼Poisson(αiγβj)xij∼Poisson(αiγβj)yij∼Poisson(αjβi)yij∼Poisson(αjβi)

To achieve this we just have to flip two lines of the predict\_results function. Instead of multiplying α and β, we divide them instead.

*# change prediction to inverse defence parameters*

predict\_results <- **function**(home, away, param\_list) {

e\_goals\_home <- (param\_list$alpha[home] / param\_list$beta[away]) \* param\_list$gamma

e\_goals\_away <- (param\_list$alpha[away] / param\_list$beta[home])

df <- data.frame(home = home, away = away,

e\_hgoal = e\_goals\_home, e\_agoal = e\_goals\_away)

**return**(df)

}

*# re run using new subfunction*

optimised\_parameters2 <- optim(

par = unlist(equal\_parameters),

fn = optimise\_params,

results = results,

method = "Nelder-Mead",

control = **list**(maxit = 10000))

*# check this does what we want*

optimised\_parameters2$par

(n.b. I won’t print out the results of all these steps as this post is long enough, but you can run and see the gradual improvements for yourself)

Next we want to subtly change how the expected goals are calculated.

Given that

A=B⋅CDA=B⋅CDis exactly the same as

A=elog(B)+log(C)−log(D)A=elog(B)+log(C)−log(D)we can convert the parameters we are looking for into log(parameters) and take the exponent of their sum as the predicted goals. This might seem like a minor change, but prevents an important exception. Using home goals as an example, remember that

xij∼Poisson(αiγβj)xij∼Poisson(αiγβj)if any of the three parameters become negative then we’re left with a Poisson distribution with a negative mean, which is is absurd: events cannot unhappen. For instance, imagine a football game where one team scores negative goals.

If we take the log parameters instead we have

xij∼Poisson(eαi−βj+γ)xij∼Poisson(eαi−βj+γ)where no matter what values α, β, or γ take, the exponent of their sum will never be negative. When playing a very strong away teams, the mean goals will tend towards 0 (though will never actually reach it).

*# change prediction to use log parameters*

*# exp(log(x) + log(y)) = x \* y*

predict\_results <- function(home, away, param\_list) {

e\_goals\_home <- **exp**(param\_list$alpha[home] - param\_list$beta[away] + param\_list$gamma)

e\_goals\_away <- **exp**(param\_list$alpha[away] - param\_list$beta[home])

df <- data.frame(home = home, away = away,

e\_hgoal = e\_goals\_home, e\_agoal = e\_goals\_away)

**return**(df)

}

*# initialise parameters as all 0*

*# log(1) = 0*

equal\_parameters <- list(

alpha = rep(0, **length**(teams)) %>% `names<-`(teams),

beta = rep(0, **length**(teams)) %>% `names<-`(teams),

gamma = 0

)

*# re run using new subfunction*

optimised\_parameters3 <- optim(

par = unlist(equal\_parameters),

fn = optimise\_params,

results = results,

*# using log will avoid non-finite differences*

*# so can use BFGS model*

method = "BFGS",

control = list(maxit = 10000))

Finally, we want to constrain the final optimised parameters by fixing the sum of all attack parameters, and the sum of all defence parameters, to equal 0. In practice, this basically means that above average attacking/defending teams will have parameters above 0, and below average teams will have parameters below 0. This is handy, but also the main advantage is this prevents [overfitting](https://en.wikipedia.org/wiki/Overfitting) of the parameters by the optimisation algorithm.

To do this, we can simply drop the first (or last, or any, it doesn’t matter) parameter from attack or defence (the parameters for Arsenal) and then calculate Arsenal’s parameters as the sum of the remaining parameters multiplied by minus 1.

αn=−n−1∑i=1αiαn=−∑i=1n−1αiand also

βn=−n−1∑i=1βiβn=−∑i=1n−1βiIn terms of code this just requires adding one line to the relist\_params() function to append the value back. We also then need to remove this parameter that we will add back in from the initial parameters which is done below.

Our output will now be missing the parameters for Arsenal (as they will only exist within the function), but we can easily calculate it from the parameters we do get out.

*# introduce sum to zero constraint by calculating*

*# first teams parameters as minus sum of the rest*

relist\_params <- function(parameters) {

parameter\_list <- list(

alpha = parameters %>%

.[grepl("alpha", names(.))] %>%

append(prod(sum(.), -1), .) %>%

`names<-`(teams),

beta = parameters %>%

.[grepl("beta", names(.))] %>%

append(prod(sum(.), -1), .) %>%

`names<-`(teams),

gamma = parameters["gamma"]

)

**return**(parameter\_list)

}

*# remove the first team from the attack and defence ratings*

equal\_parameters <- list(

alpha = rep(0, **length**(teams)-1) %>% `names<-`(teams[2:**length**(teams)]),

beta = rep(0, **length**(teams)-1) %>% `names<-`(teams[2:**length**(teams)]),

gamma = 0

)

*# initialise i to collect data about the optimisation process at each iteration*

i <- 0

*# collect current parameter values and neg log likelihood at each iteration*

current\_parameters <- list()

current\_nll <- list()

We can then final the optim() function one final time to get our final optimised parameters

# run our final calculation

optimised\_parameters4 <- optim(

par = unlist(equal\_parameters),

fn = optimise\_params,

results = results,

method = "BFGS",

control = list(maxit = 10000))

We can plot the log likelihood at each iteration. Notice how it starts around <120, which is pretty close what our worse\_log\_likelihood returned. For these optimisations, the original parameters we are supplying are similar to the zeroed parameters for that example.

As the optim() function plays with the parameters you can see the log likelihood jumps around quite violently, but over time tend towards zero.

p3 <- data.frame(likelihood = unlist(current\_nll),

iteration = seq(length(current\_nll))) %>%

ggplot(aes(**x** = iteration, **y** = likelihood)) +

geom\_line(colour = "red") +

*# cut out some cases where optim() has been a bit ambitious*

coord\_cartesian(ylim = c(0, 250)) +

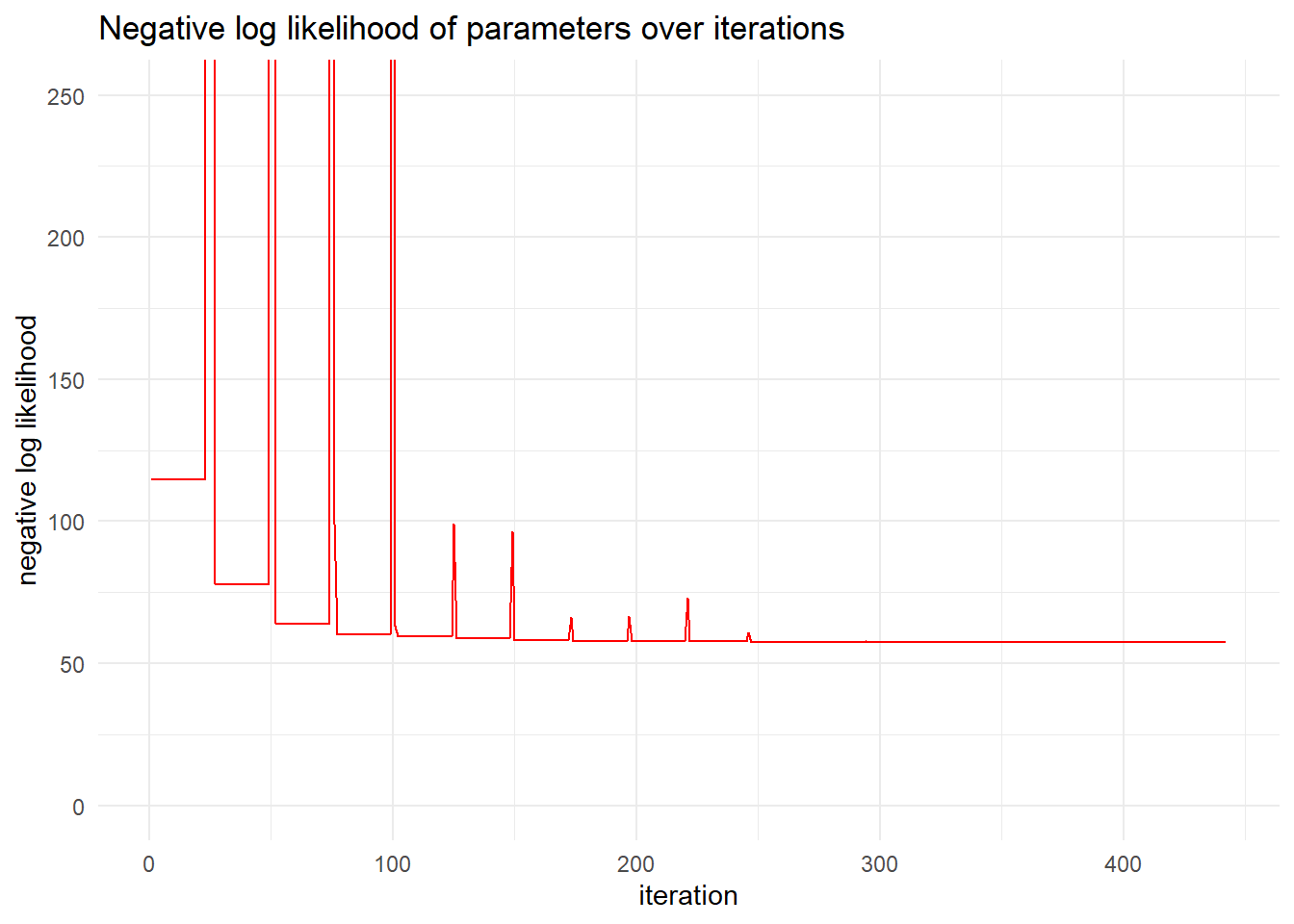
labs(title = "Negative log likelihood of parameters over iterations",

**y** = "negative log likelihood",

**x** = "iteration") +

theme\_minimal()

p3



The final parameters can also be extracted from the output from optim() and plotted:

p4 <- optimised\_parameters4$par %>%

*# relist to add in first team*

relist\_params() %>%

unlist() %>%

*# select team parameters*

.[grepl("beta|alpha", names(.))] %>%

data.frame(value = .,

parameter = names(.)) %>%

separate(parameter, into = c("parameter", "team"), "\\.") %>%

*# spread into wide format*

spread(parameter, value) %>%

*# pipe into a plot*

ggplot(aes(x = alpha, y = beta)) +

geom\_point() +

ggrepel::geom\_text\_repel(aes(label = team)) +

stat\_smooth(method = "lm", se = **FALSE**) +

labs(title = "Optimal parameters for teams",

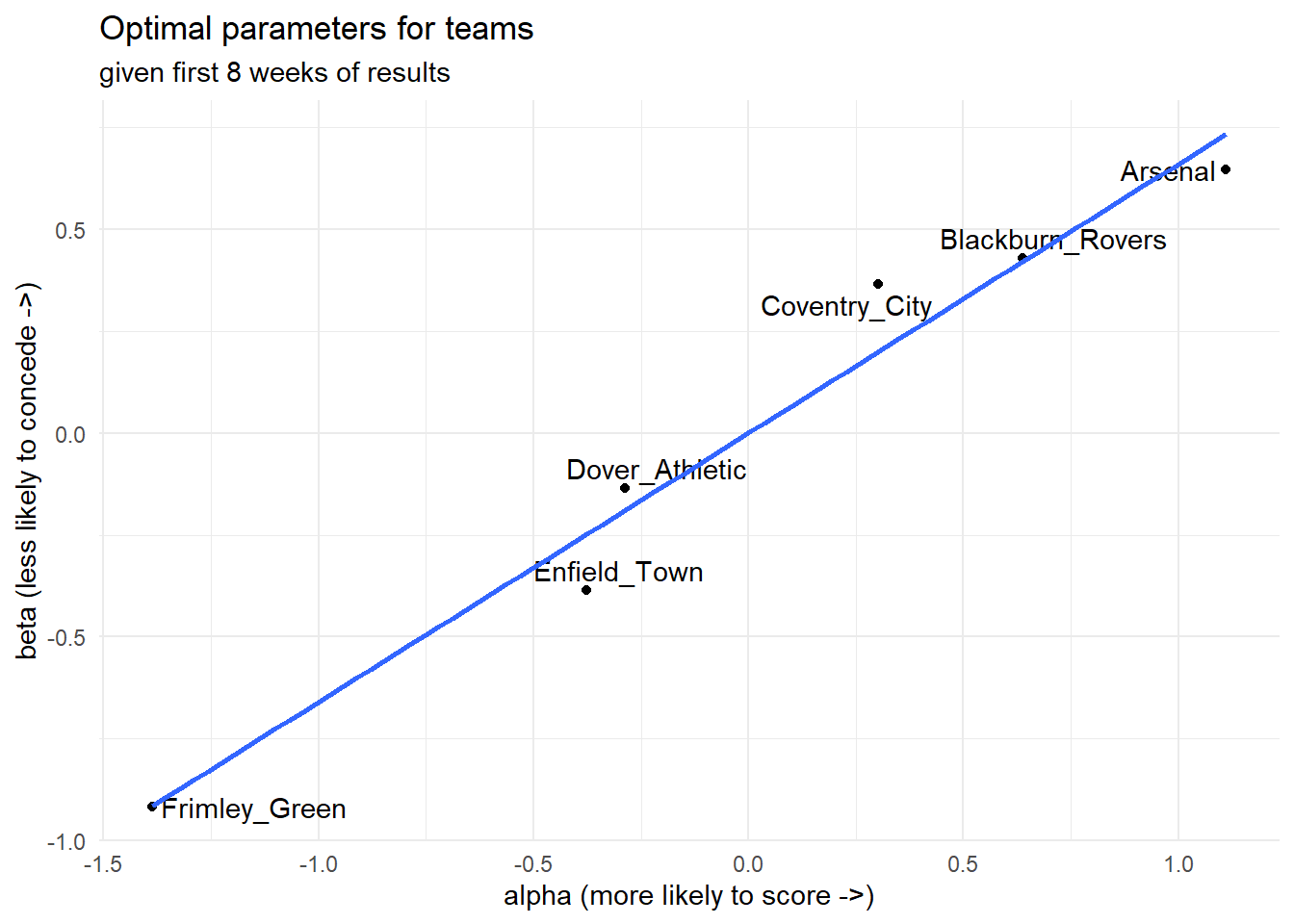
subtitle = "given first 8 weeks of results",

x = "alpha (more likely to score ->)",

y = "beta (less likely to concede ->)") +

theme\_minimal()

p4

Notice how the teams monotonically increase in both attack and defensive ability. This is by design on how the results were created (see the bottom of this post). With only 8 games per team however, there is quite a lot of noise in the signal. Hitting the crossbar instead of scoring in one game could make a fairly large difference in how the function rates a team.

Also note how the regression line passes through the origin- this is a result of us constraining the parameters to sum to zero.

If we want to see how optim() selects these, we can plot how they change over iterations. You can see how it jumps around then settles on incremental improvements to the model.

p5 <- current\_parameters %>%

*# get the parameters for arsenal for each iteration*

lapply(., **function**(x){ unlist(relist\_params(x))}) %>%

map\_df(bind\_rows, .id = "iteration") %>%

*# melt data and split parameters into team and parameter*

gather("parameter", "value", -iteration) %>%

*# get rid of the gamma parameter*

filter(parameter != "gamma.gamma") %>%

separate(parameter, into = c("parameter", "team"), sep = "\\.") %>%

*# spread data back by parameter*

spread(parameter, value) %>%

mutate(iteration = **as**.numeric(iteration)) %>%

*# plot alpha against beta for each iteration*

ggplot(aes(x = alpha, y = beta)) +

geom\_text(aes(label = team)) +

labs(title = 'Parameters for Iteration {floor(frame\_time)}',

subtitle = "given first 8 weeks of results",

x = "alpha (more likely to score ->)",

y = "beta (less likely to concede ->)") +

*# using gganimate package*

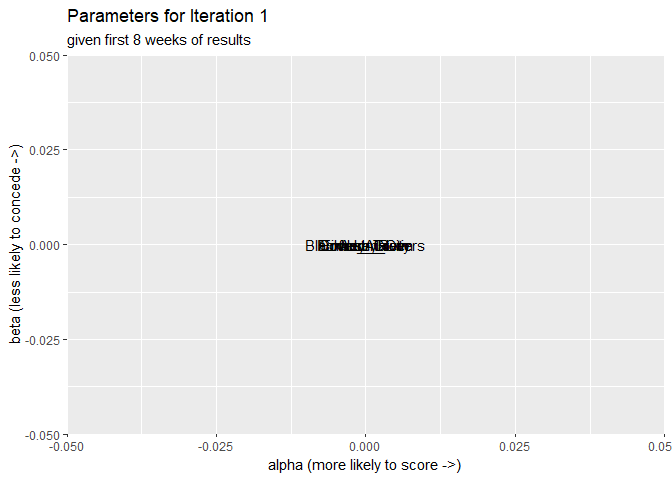
gganimate::transition\_time(iteration) +

gganimate::ease\_aes('linear') +

gganimate::view\_follow()

*# animate the plot*

gganimate::animate(p5, nframes = i)



Predict Remaining Matches

Now we have rated each teams attack/defense, and the advantage to a team to play at home, we can predict the remaining matches between the teams.

For this, we just have to use the predict\_results() function we defined earlier, except this time the output will be the expected goals per team. Earlier we were measuring the deviance from expectation, but not we assume the most likely result is exactly equal to the expected results. If we wanted to we could work out how likely this result is, and what the most likely results are.

This post is long enough however, so for now, we’ll just detail the most likely results.

predicted\_results <- predict\_results(unplayed\_games$home,

unplayed\_games$away,

relist\_params(optimised\_parameters4$par)) %>%

mutate\_if(is.numeric, round, 2) %>%

print()

**#***# home away e\_hgoal e\_agoal*

**#***# 1 Coventry\_City Arsenal 0.86 2.11*

**#***# 2 Blackburn\_Rovers Dover\_Athletic 2.62 0.49*

**#***# 3 Frimley\_Green Enfield\_Town 0.44 1.72*

**#***# 4 Arsenal Blackburn\_Rovers 2.39 0.99*

**#***# 5 Coventry\_City Frimley\_Green 4.09 0.17*

**#***# 6 Dover\_Athletic Enfield\_Town 1.33 0.79*

All of these look reasonable, with better teams beating worse ones. The only match that the model thinks might well end in a draw is Dover at home to Enfield, which is not entirely unreasonable.

We can add these predictions to our earlier matrix of results to get a sense if these fit in with the trend from the observed matches:

p6 <- rbind(

predicted\_results %>%

rename\_if(is.numeric, gsub, pattern = "e\_", replacement = "") %>%

mutate(type = "predicted"),

results %>%

select(-gameweek) %>%

mutate(type = "result")

) %>%

ggplot(., aes(x = away, y = home, fill = hgoal-agoal)) +

geom\_tile() +

*# add the scorelines*

geom\_label(aes(label = paste(hgoal, agoal, sep = "-"), colour = type), fill = "white") +

*# colour where black for actual results and red for predictions*

scale\_colour\_manual(values = c("red", "black")) +

*# colour where green shows home win and red an away win*

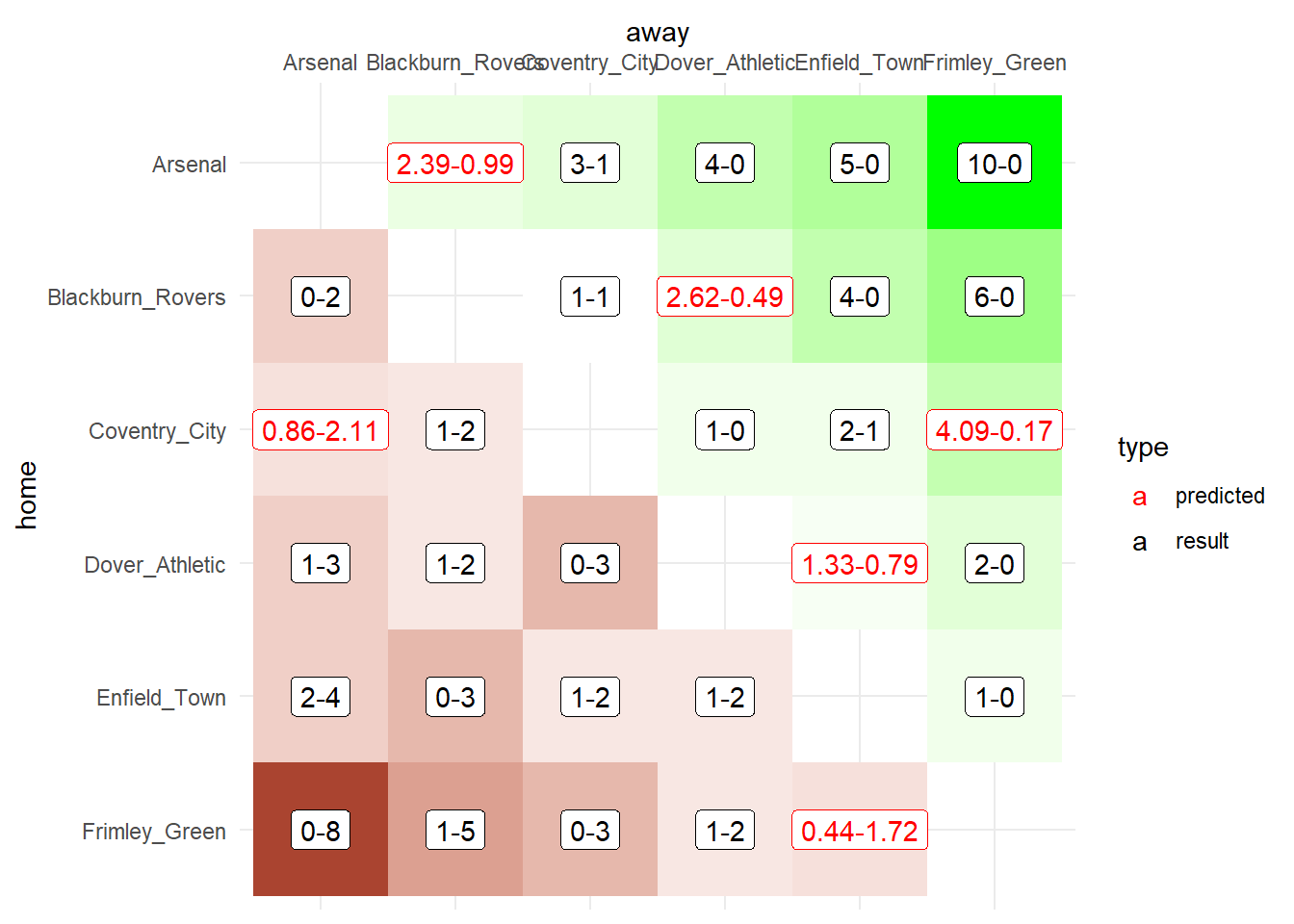
scale\_fill\_gradient2(low = "darkred", high = "green", midpoint = 0, guide = FALSE) +

scale\_x\_discrete(limits = levels(results$home), position = "top") +

scale\_y\_discrete(limits = rev(levels(results$away))) +

theme\_minimal()

p6



Which they do! The predicted results fit in with the gradient of heavier defeats for home teams towards the bottom left, progressing to easy home victories in the top right.

That’s all for this post. Hopefully using the Poisson distribution to model football matches is a little clearer now. Feel free to email me any questions and check out the packages I stole all the codes/idea from.

Next time, I’ll go over how to quantify the probability of a range of results for any single match in (hopefully) a shorter post; until then!

Results Generation

First we need to create a data.frame of fixtures for each team

*# https://stackoverflow.com/questions/54099990/is-there-an-efficient-algorithm-to-create-this-type-of-schedule*

create\_fixtures <- function(teams) {

*# keep team 1 in place*

team1 <- as.character(teams[1])

*#rotate other teams around team 1*

other\_teams <- as.character(teams[!teams %in% team1])

**length** <- **length**(other\_teams)

*# generate fixtures each week*

**for**(week in seq((length(teams)-1)\*2)) {

**if**(week %% 2 == 0) {

fixtures <- data.frame(home = c(team1, other\_teams[1:2]),

away = other\_teams[**length**:3],

gameweek = week)

} **else** {

fixtures <- data.frame(home = other\_teams[**length**:3],

away = c(team1, other\_teams[1:2]),

gameweek = week)

}

**if**(week == 1) {

fixtures\_df <- fixtures

} **else** {

fixtures\_df <- rbind(fixtures\_df, fixtures)

}

*# rotate other teams around*

other\_teams <- c(other\_teams[**length**], other\_teams[1:**length**-1])

}

**return**(fixtures\_df)

}

*# create the fixtures*

fixtures <- create\_fixtures(teams) %>%

mutate\_if(is.factor, as.character)

*# print the fixture list*

fixtures

**#***# home away gameweek*

**#***# 1 Frimley\_Green Arsenal 1*

**#***# 2 Enfield\_Town Blackburn\_Rovers 1*

**#***# 3 Dover\_Athletic Coventry\_City 1*

**#***# 4 Arsenal Enfield\_Town 2*

**#***# 5 Frimley\_Green Dover\_Athletic 2*

**#***# 6 Blackburn\_Rovers Coventry\_City 2*

**#***# 7 Dover\_Athletic Arsenal 3*

**#***# 8 Coventry\_City Enfield\_Town 3*

**#***# 9 Blackburn\_Rovers Frimley\_Green 3*

**#***# 10 Arsenal Coventry\_City 4*

**#***# 11 Dover\_Athletic Blackburn\_Rovers 4*

**#***# 12 Enfield\_Town Frimley\_Green 4*

**#***# 13 Blackburn\_Rovers Arsenal 5*

**#***# 14 Frimley\_Green Coventry\_City 5*

**#***# 15 Enfield\_Town Dover\_Athletic 5*

**#***# 16 Arsenal Frimley\_Green 6*

**#***# 17 Blackburn\_Rovers Enfield\_Town 6*

**#***# 18 Coventry\_City Dover\_Athletic 6*

**#***# 19 Enfield\_Town Arsenal 7*

**#***# 20 Dover\_Athletic Frimley\_Green 7*

**#***# 21 Coventry\_City Blackburn\_Rovers 7*

**#***# 22 Arsenal Dover\_Athletic 8*

**#***# 23 Enfield\_Town Coventry\_City 8*

**#***# 24 Frimley\_Green Blackburn\_Rovers 8*

**#***# 25 Coventry\_City Arsenal 9*

**#***# 26 Blackburn\_Rovers Dover\_Athletic 9*

**#***# 27 Frimley\_Green Enfield\_Town 9*

**#***# 28 Arsenal Blackburn\_Rovers 10*

**#***# 29 Coventry\_City Frimley\_Green 10*

**#***# 30 Dover\_Athletic Enfield\_Town 10*

and then create the results

*# using goalmodel package*

*# https://github.com/opisthokonta/goalmodel*

library(goalmodel)

*# have to manually create a list of parameters*

model <- list()

*# stratify teams abilities in attack and defense*

model$parameters <- list(attack = seq(1, -1 + 2/length(teams), by = -2/(**length**(teams)-1)) %>%

append(-sum(.)) %>%

`names<-`(teams),

defense = seq(1, -1 + 2/length(teams), by = -2/(**length**(teams)-1)) %>%

append(-sum(.)) %>%

`names<-`(teams),

*# no base rate of goals*

intercept = 0,

*# roughly accurate hfa for English professional football*

hfa = 0.3)

*# add in teams*

model$all\_teams <- teams

*# use a simple Poisson model with 8 goals max*

model$model <- "poisson"

model$maxgoal <- 8

*# use the model to predict results using regista package*

results <- predict\_expg(model, fixtures$home, fixtures$away, return\_df = TRUE) %>%

*# add some noise*

mutate(noise1 = rnorm(nrow(.), 0, 0.5),

noise2 = rnorm(nrow(.), 0, 0.5)) %>%

mutate(hgoal = round(expg1 + noise1,0 ),

agoal = round(expg2 + noise2,0),

home = as.factor(team1),

away = as.factor(team2)) %>%

*# merge to fixtures*

merge(., fixtures, by = c("home", "away")) %>%

*# cant score less than zero goals*

mutate\_at(vars(hgoal:agoal), funs(replace(., .<0, 0))) %>%

**select**(home, away, hgoal, agoal, gameweek) %>%

arrange(gameweek, home) %>%

*# treat only first 8 weeks as played*

filter(gameweek <= 8)

*# print results*

results

**#***# home away hgoal agoal gameweek*

**#***# 1 Dover\_Athletic Coventry\_City 0 2 1*

**#***# 2 Enfield\_Town Blackburn\_Rovers 1 3 1*

**#***# 3 Frimley\_Green Arsenal 0 6 1*

**#***# 4 Arsenal Enfield\_Town 6 0 2*

**#***# 5 Blackburn\_Rovers Coventry\_City 2 0 2*

**#***# 6 Frimley\_Green Dover\_Athletic 0 3 2*

**#***# 7 Blackburn\_Rovers Frimley\_Green 8 0 3*

**#***# 8 Coventry\_City Enfield\_Town 3 1 3*

**#***# 9 Dover\_Athletic Arsenal 1 3 3*

**#***# 10 Arsenal Coventry\_City 3 0 4*

**#***# 11 Dover\_Athletic Blackburn\_Rovers 1 2 4*

**#***# 12 Enfield\_Town Frimley\_Green 2 1 4*

**#***# 13 Blackburn\_Rovers Arsenal 2 2 5*

**#***# 14 Enfield\_Town Dover\_Athletic 0 2 5*

**#***# 15 Frimley\_Green Coventry\_City 1 3 5*

**#***# 16 Arsenal Frimley\_Green 9 0 6*

**#***# 17 Blackburn\_Rovers Enfield\_Town 5 0 6*

**#***# 18 Coventry\_City Dover\_Athletic 1 2 6*

**#***# 19 Coventry\_City Blackburn\_Rovers 0 2 7*

**#***# 20 Dover\_Athletic Frimley\_Green 3 1 7*

**#***# 21 Enfield\_Town Arsenal 0 5 7*

**#***# 22 Arsenal Dover\_Athletic 4 1 8*

**#***# 23 Enfield\_Town Coventry\_City 1 2 8*

**#***# 24 Frimley\_Green Blackburn\_Rovers 1 4 8*

For a given string of a home team and an away team, the function finds the relevant parameters from a third argument (param\_list) and calculates the expected goal for each team.

predict\_results <- function(home, away, param\_list) {

e\_goals\_home <- exp(param\_list$alpha[home] - param\_list$beta[away] + param\_list$gamma)

e\_goals\_away <- exp(param\_list$alpha[away] - param\_list$beta[home])

df <- data.frame(home = home, away = away,

e\_hgoal = as.numeric(e\_goals\_home),

e\_agoal = as.numeric(e\_goals\_away))

return(df)

}

If we run this for two example teams for example:

#two example teams

home <- "Blackburn\_Rovers"

away <- "Arsenal"

prediction <- predict\_results(home, away, model)

prediction

## home away e\_hgoal e\_agoal

## 1 Blackburn\_Rovers Arsenal 1.198128 1.977293

We can see that it gives Arsenal (the away team) a slightly more optimistic chance than Blackburn. The expected goals for each team of course can be rewritten as the mean, and in our Poisson model refers to λ (lambda)- the mean times an event (goal) happens per a time interval (match). We also set a maximum number of possible goals (7 in this case\*) to bound the area under the distribution so we aren’t sampling forever.

\*sharp readers might notice that this is actually *lower* than the lambda for our more extreme cases (e.g. Arsenal at home to Frimley Green), but for realistic matches (even between wildly different professional sides) this is a fair enough assumption.

We then use dpois() to calculate the probability of this Poisson function returning a value (0:7 goals) given it’s lambda value. So if we run this over the prediction we made for Blackburn Rovers vs. Arsenal we get:

#set a limit of where we'll calculate across

max\_goals <- 7

#calculate the probability of scoring x goals for either team

blackburn\_goal\_probs <- lapply(0:max\_goals, dpois, lambda = prediction$e\_hgoal)

arsenal\_goal\_probs <- lapply(0:max\_goals, dpois, lambda = prediction$e\_agoal)

#bind together in a df

df <- data.frame(goals = rep(0:max\_goals, 2),

team = rep(c(home, away), each = max\_goals+1),

p = c(unlist(blackburn\_goal\_probs), unlist(arsenal\_goal\_probs)))

#plot the p of scoring x goals for either team

p1 <- ggplot(df, aes(x = goals, y = p, fill = team)) +

geom\_density(stat = "identity", alpha = 0.5) +

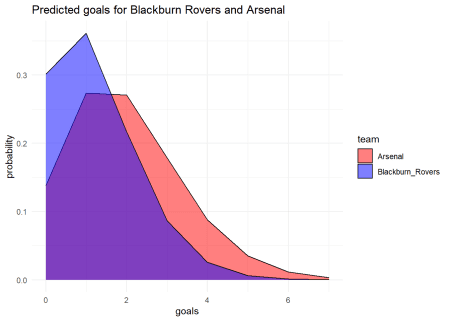
scale\_fill\_manual(values = c("red", "blue")) +

labs(title = "Predicted goals for Blackburn Rovers and Arsenal",

y = "probability") +

theme\_minimal()

p1



Because of how maths works, these curves are the same result we would get if we ran rpois() (sampling from the Poisson function) lots of times. We’ll do that quickly because it sets the stage nicely for what will come later.

#sample from the function lots of times for each team

n <- 100000

blackburn\_goals\_samples <- rpois(n, lambda = prediction$e\_hgoal)

arsenal\_goals\_samples <- rpois(n, lambda = prediction$e\_agoal)

df <- data.frame(team = rep(c(home, away), each = n),

sampled\_goals = c(blackburn\_goals\_samples, arsenal\_goals\_samples))

#look its the same plot!

p2 <- ggplot(df, aes(x = sampled\_goals, fill = team)) +

geom\_bar(stat = "count", position = "dodge", colour = "black", alpha = 0.5) +

geom\_line(aes(colour = team), stat = "count", size = 3) +

scale\_fill\_manual(values = c("red", "blue"), guide = FALSE) +

scale\_colour\_manual(values = c("red", "blue"), guide = FALSE) +

labs(title = "Predicted goals for Blackburn Rovers and Arsenal",

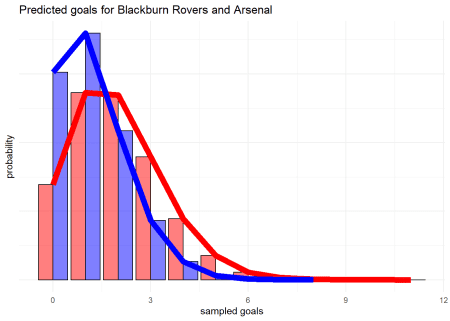
y = "probability",

x = "sampled goals") +

theme\_minimal() +

theme(axis.text.y = element\_blank())

p2



Ok great!, in terms of predicting the result, the rightwards shift of the red (Arsenal) curve here is the difference in the teams ability to generate a positive goal differential- it makes it more likely that if we sample event, Arsenal will have scored more goals than Blackburn Rovers at the end of the match. Of course, it’s also obvious that while Arsenal’s curve is right shifted, the bars for Arsenal scoring 0 goals and Blackburn scoring 6 are still sizable enough that it isn’t outside the realm of possibility.

This is a nice way of presenting the chance of each team scoring n goals, but doesn’t really help us in predicting the result of a match given that this relies on the interaction of both these distributions (we need to know how many goals BOTH Arsenal AND Blackburn will score).

To calculate this, we can do an outer product of the probabilities for both teams scoring n goals. We can then plot the probability of each *scoreline* as a tile plot:

#calculate matrix of possible results and probabilities of those

matrix <- outer(unlist(arsenal\_goal\_probs), unlist(blackburn\_goal\_probs)) %>%

as.data.frame() %>%

gather() %>%

#add in scorelines

mutate(hgoals = rep(0:max\_goals, max\_goals+1),

agoals = rep(0:max\_goals, each = max\_goals+1))

#make the tile plot

p3 <- ggplot(matrix, aes(x = hgoals, y = agoals, fill = value)) +

geom\_tile() +

geom\_text(aes(label = paste(hgoals, agoals, sep = "-"))) +

scale\_fill\_gradient2(low = "white", high = "red", guide = FALSE) +

theme\_minimal()

p3



Where we can see that the most common scorelines are low scoring (football is a low scoring game), and slightly biased towards away goals (i.e. Arsenal are more likely to win than lose). The darkest (most likely) tiles being 1-1 or a 2-1 Arsenal win seem very plausible given our calculated λs earlier.

We can then do this for every fixture and build a large graph of the expected results for each using a simple map2\_ apply. Because of the huge plot, I’ve restricted it here to a 3×3 matrix of the results for Arsenal, Coventry City, and Enfield Town, but if you click you should be linked to the full image.

#want to predict over the whole fixture space

all\_fixtures <- bind\_rows(fixtures, results) %>%

filter(!duplicated(paste(home, away), fromLast = TRUE))

#get the lambda for each team per game

predictions <- map2\_df(all\_fixtures$home, all\_fixtures$away,

predict\_results,

model)

#calc out probabilities and bind up

all\_predictions <- map2\_df(

predictions$e\_hgoal, predictions$e\_agoal,

function(lambda\_home, lambda\_away, max\_goals) {

hgoal\_prob <- dpois(0:max\_goals, lambda\_home) %>% `names<-`(0:max\_goals)

agoal\_prob <- dpois(0:max\_goals, lambda\_away) %>% `names<-`(0:max\_goals)

outer(hgoal\_prob, agoal\_prob) %>%

as.data.frame() %>%

gather() %>%

rownames\_to\_column("row") %>%

mutate(hgoal = as.numeric(row) %% (max\_goals+1)-1) %>%

mutate(hgoal = case\_when(hgoal < 0 ~ max\_goals, TRUE ~ hgoal),

agoal = as.numeric(key)) %>%

select(sample\_hgoal = hgoal, sample\_agoal = agoal, prob = value)

}, max\_goals) %>%

cbind(all\_fixtures[rep(seq\_len(nrow(all\_fixtures)), each=(max\_goals+1)^2),], .) %>%

group\_by(home, away) %>%

mutate(prob = prob / sum(prob)) %>%

ungroup()

#plot again

p3 <- all\_predictions %>%

#filter only a few out to scale plot

filter(home %in% c("Arsenal", "Coventry\_City", "Enfield\_Town"),

away %in% c("Arsenal", "Coventry\_City", "Enfield\_Town")) %>%

ggplot(aes(x = sample\_hgoal, y = sample\_agoal, fill = prob)) +

geom\_tile() +

geom\_point(aes(x = hgoal, y = agoal),

colour = "blue", size = 5, alpha = 0.5 / max\_goals^2) +

geom\_text(aes(label = paste(sample\_hgoal, sample\_agoal, sep = "-")), size = 2.3) +

scale\_fill\_gradient2(low = "white", high = "red", guide = FALSE) +

labs(

title = "predictions for final score across all fixtures",

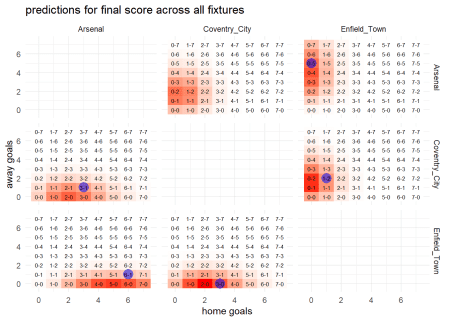
y = "away goals",

x = "home goals") +

theme\_minimal() +

facet\_grid(away~home, scales = "free")

p3



For the whole matrix, click [here](https://www.robert-hickman.eu/img/results_matrix.png)

**So what?**

These graphs are nice, but whats important is what they show: *we have a way to quantify how likely any result is in a match between two given teams*. Why is this useful

**“if events follow a known distribution\*, you can sample these events lots of times to get stochastic guesstimates, but over many samples you will reproduce exactly that distribution”**

\*a Poisson distribution for the expected number of goals scored in our case

For football, this means that while on an individual match level results are noisy (sometimes better teams lose!), if we simulate matches lots and lots of times, eventually they should converge to the ‘truth’\*

\*as defined by our Poisson distribution (which may or may not be a good/accurate ‘truth’ but go with it for now).

To work with this highly repetitive data, first we want to ‘nest’ the probabilities for each match. This basically means storing a df of all the possible results and their probabilities as a column inside a larger df so we can move between the data in those two structures easier.

For instance, the nest match results probability information for the next match to be played (Coventry City and home to Arsenal):

nested\_probabilities <- all\_predictions %>%

filter([is.na](http://is.na)(hgoal)) %>%

select(-hgoal, -agoal) %>%

nest(probabilities = c(sample\_hgoal, sample\_agoal, prob))

nested\_probabilities$probabilities[[1]] %>%

rename("Coventry City" = sample\_hgoal, "Arsenal" = sample\_agoal) %>%

arrange(-prob) %>%

#show first 15 rows

.[1:15,]

## # A tibble: 15 x 3

## `Coventry City` Arsenal prob

##

## 1 0 2 0.115

## 2 0 1 0.109

## 3 1 2 0.0983

## 4 1 1 0.0933

## 5 0 3 0.0806

## 6 1 3 0.0691

## 7 0 0 0.0516

## 8 1 0 0.0442

## 9 0 4 0.0425

## 10 2 2 0.0422

## 11 2 1 0.0400

## 12 1 4 0.0364

## 13 2 3 0.0296

## 14 2 0 0.0190

## 15 0 5 0.0179

The probability for any single result is small (otherwise match betting would be easy), but the probabilities for a 2-0 and 1-0 Arsenal wins are highest (as we found earlier). Indeed all of the most likely results are within a goal or two for either side of these.

To make sure these probabilities makes sense, we can sum them and see that the results space of 0:max\_goals for either side sums to 1

sum(nested\_probabilities$probabilities[[1]]$prob)

## [1] 1

Then we can easily use this data to simulate results. We sample a single row (a ‘result’ of the match) weighted by the probability of it occurring. For instance, when we sample from the Coventry City vs Arsenal match it picks a 3-1 Arsenal away win (not the likeliest result, but not the most unlikely either).

nested\_probabilities$probabilities[[1]] %>%

rename("Coventry\_City" = sample\_hgoal, "Arsenal" = sample\_agoal) %>%

sample\_n(1, weight = prob)

## # A tibble: 1 x 3

## Coventry\_City Arsenal prob

##

## 1 1 3 0.0691

We can of course repeat this across every match and see that the probabilities of the chosen results vary (because we’re randomly sampling we won’t always choose the most likely, or even a likely result), but all are within a reasonable range given the team playing:

nested\_probabilities %>%

mutate(sampled\_result = map(probabilities, sample\_n, 1, weight = prob)) %>%

select(-probabilities) %>%

unnest(cols = c(sampled\_result))

## # A tibble: 6 x 6

## home away gameweek sample\_hgoal sample\_agoal prob

##

## 1 Coventry\_City Arsenal 9 0 5 0.0179

## 2 Blackburn\_Rovers Dover\_Athletic 9 1 1 0.0575

## 3 Frimley\_Green Enfield\_Town 9 0 4 0.0418

## 4 Arsenal Blackburn\_Rovers 10 2 1 0.0966

## 5 Coventry\_City Frimley\_Green 10 3 0 0.170

## 6 Dover\_Athletic Enfield\_Town 10 2 1 0.0839

But when we are predicting what will happen, we want to find the *most likely* result. As mentioned earlier, if we sample enough, our average will converge towards this, so we can repeat this sampling technique n times (here I’ve done it 10 times), depending on how much time we want to wait for it to process.

You can see that as we do this many times, the results with the highest probability turn up more than others- as we would expect if we were to (e.g.) actually play Blackburn Rovers vs Arsenal many times.

rerun(10, nested\_probabilities %>%

filter(home == "Coventry\_City" & away == "Arsenal") %>%

mutate(sampled\_result = map(probabilities, sample\_n, 1, weight = prob)) %>%

select(-probabilities) %>%

unnest(cols = c(sampled\_result))

) %>%

bind\_rows() %>%

arrange(-prob)

## # A tibble: 10 x 6

## home away gameweek sample\_hgoal sample\_agoal prob

##

## 1 Coventry\_City Arsenal 9 0 2 0.115

## 2 Coventry\_City Arsenal 9 1 2 0.0983

## 3 Coventry\_City Arsenal 9 1 1 0.0933

## 4 Coventry\_City Arsenal 9 0 3 0.0806

## 5 Coventry\_City Arsenal 9 1 3 0.0691

## 6 Coventry\_City Arsenal 9 1 0 0.0442

## 7 Coventry\_City Arsenal 9 0 4 0.0425

## 8 Coventry\_City Arsenal 9 0 4 0.0425

## 9 Coventry\_City Arsenal 9 0 4 0.0425

## 10 Coventry\_City Arsenal 9 1 5 0.0154

If we do this a few more times per fixture (here 100, for a better estimate I’d advise at least 10000- it should only take a few minutes), we can then start assigning points and goal difference to each team based upon the result we’ve sampled. E.g. if one sample predicts Arsenal to beat Blackburn Rovers 4-0, we assign 3 points to Arsenal and 0 points to Blackburn Rovers for that simulation and +4 and -4 goal difference respectively.

n <- 100

fixture\_sims <- rerun(n, nested\_probabilities %>%

mutate(sampled\_result = map(probabilities, sample\_n, 1, weight = prob)) %>%

select(-probabilities) %>%

unnest(cols = c(sampled\_result)) %>%

select(-gameweek, -prob) %>%

pivot\_longer(c(home, away), names\_to = "location", values\_to = "team") %>%

mutate(points = case\_when(

location == "home" & sample\_hgoal > sample\_agoal ~ 3,

location == "away" & sample\_agoal > sample\_hgoal ~ 3,

sample\_hgoal == sample\_agoal ~ 1,

TRUE ~ 0

)) %>%

mutate(gd = case\_when(

location == "home" ~ sample\_hgoal - sample\_agoal,

location == "away" ~ sample\_agoal - sample\_hgoal

)))

fixture\_sims[1]

## [[1]]

## # A tibble: 12 x 6

## sample\_hgoal sample\_agoal location team points gd

##

## 1 0 0 home Coventry\_City 1 0

## 2 0 0 away Arsenal 1 0

## 3 4 0 home Blackburn\_Rovers 3 4

## 4 4 0 away Dover\_Athletic 0 -4

## 5 0 0 home Frimley\_Green 1 0

## 6 0 0 away Enfield\_Town 1 0

## 7 3 0 home Arsenal 3 3

## 8 3 0 away Blackburn\_Rovers 0 -3

## 9 6 1 home Coventry\_City 3 5

## 10 6 1 away Frimley\_Green 0 -5

## 11 1 1 home Dover\_Athletic 1 0

## 12 1 1 away Enfield\_Town 1 0

We can then average the points and goal difference won in these sims across each team and see what teams are predicted to win across their fixtures.

fixture\_sims %>%

bind\_rows() %>%

group\_by(team) %>%

summarise(av\_points = sum(points)/n,

av\_gd = sum(gd) / n)

## # A tibble: 6 x 3

## team av\_points av\_gd

##

## 1 Arsenal 4.19 2.44

## 2 Blackburn\_Rovers 3.16 0.7

## 3 Coventry\_City 3.61 2.42

## 4 Dover\_Athletic 2.26 -1.26

## 5 Enfield\_Town 2.95 0.23

## 6 Frimley\_Green 0.6 -4.53

Where we can see that we expect Arsenal to win 4.19 out of a possible 6 points (with games remaining against Coventry and Blackburn Rovers they are expected to drop points but win at least one and probably draw the other). Coventry City are expected to also do well- probably because their final game is at home to Frimley Green, whereas Blackburn have tougher fixtures away at Arsenal and home to Dover Athletic.

We can then add this to the calculated points teams have *already* accrued to get a prediction of where teams will end the season position wise:

table <- results %>%

pivot\_longer(c(home, away), names\_to = "location", values\_to = "team") %>%

mutate(points = case\_when(

location == "home" & hgoal > agoal ~ 3,

location == "away" & agoal > hgoal ~ 3,

hgoal == agoal ~ 1,

TRUE ~ 0

)) %>%

mutate(gd = case\_when(

location == "home" ~ hgoal - agoal,

location == "away" ~ agoal - hgoal

)) %>%

group\_by(team) %>%

summarise(points = sum(points),

gd = sum(gd))

predicted\_finishes <- map\_df(fixture\_sims, function(simulated\_fixtures, table) {

simulated\_fixtures %>%

select(team, points, gd) %>%

bind\_rows(., table) %>%

group\_by(team) %>%

summarise(points = sum(points),

gd = sum(gd)) %>%

arrange(-points, -gd) %>%

mutate(predicted\_finish = 1:n())

}, table) %>%

group\_by(team, predicted\_finish) %>%

summarise(perc = n() / n)

predicted\_finishes

## # A tibble: 10 x 3

## # Groups: team [6]

## team predicted\_finish perc

##

## 1 Arsenal 1 0.82

## 2 Arsenal 2 0.18

## 3 Blackburn\_Rovers 1 0.18

## 4 Blackburn\_Rovers 2 0.82

## 5 Coventry\_City 3 0.97

## 6 Coventry\_City 4 0.03

## 7 Dover\_Athletic 3 0.03

## 8 Dover\_Athletic 4 0.97

## 9 Enfield\_Town 5 1

## 10 Frimley\_Green 6 1

Which gives Arsenal an 82% chance of finishing champions, with only a 18% chance Blackburn manage to leapfrog them into 1st place. Given there are only 2 matches left with teams designed to have fairly large gulfs in ability, it’s not surprising most of the final positions are nailed on- e.g. Enfield Town finish 5th in every single simulation:

p4 <- ggplot(predicted\_finishes, aes(x = predicted\_finish, y = perc, fill = team)) +

geom\_bar(stat = "identity", colour = "black") +

scale\_fill\_manual(values = c("red", "blue", "skyblue", "white", "dodgerblue4", "blue")) +

labs(

title = "Predicted finish position of teams",

subtitle = "with two gameweeks left to play",

y = "fraction of finishes",

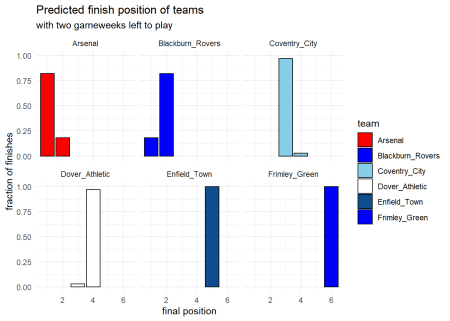
x = "final position"

) +

theme\_minimal() +

facet\_wrap(~team)

p4



**The Real Thing**

We can use the knowledge we’ve built up over these last posts to see what we expect to happen in these unplayed games, if they cannot be completed.

library(rvest)

library(regista)

First we need to download the data on the current English Premier League season. Once we have this we can split it into played matches (where we 100% know the result) and unplayed matches which we need to predict the result of. For the basis of the team strength estimates I’ve used the xg created and allowed per game, as I believe these give a better estimate of team strength

#download the match data from 2019/2020

fixtures\_2020 <- "<https://fbref.com/en/comps/9/schedule/Premier-League-Fixtures>" %>%

read\_html() %>%

html\_nodes("#sched\_ks\_3232\_1") %>%

html\_table() %>%

as.data.frame() %>%

separate(Score, into = c("hgoal", "agoal"), sep = "–") %>%

#only care about goals and expected goals

select(home = Home, away = Away, home\_xg = xG, away\_xg = xG.1, hgoal, agoal) %>%

filter(home != "") %>%

mutate(home = factor(home), away = factor(away)) %>%

#round expected goals to nearest integer

mutate\_at(c("home\_xg", "away\_xg", "hgoal", "agoal"), .funs = funs(round(as.numeric(.))))

#matches with a known result

#used for modelling

played\_matches <- fixtures\_2020 %>%

filter(![is.na](http://is.na)(home\_xg))

#matches with an unknown result

#used for simulation

unplayed\_matches <- fixtures\_2020 %>%

filter([is.na](http://is.na)(home\_xg)) %>%

select\_if(negate(is.numeric))

#fit the dixon coles model

#use xg per game, not 'actual' goals

fit\_2020 <- dixoncoles(home\_xg, away\_xg, home, away, data = played\_matches)

To get a look at what the team parameters in a real-life league look like we can extract them from the model and plot them:

#extract Dixon-Coles team strenth parameters

pars\_2020 <- fit\_2020$par %>%

.[grepl("def\_|off\_", names(.))] %>%

matrix(., ncol = 2) %>%

as.data.frame() %>%

rename(attack = V1, defence = V2)

pars\_2020$team <- unique(gsub("def\_\*|off\_\*", "", names(fit\_2020$par)))[1:20]

#plot as before

p5 <- pars\_2020 %>%

mutate(defence = 1 - defence) %>%

ggplot(aes(x = attack, y = defence, colour = attack + defence, label = team)) +

geom\_point(size = 3, alpha = 0.7) +

geom\_text\_repel() +

scale\_colour\_continuous(guide = FALSE) +

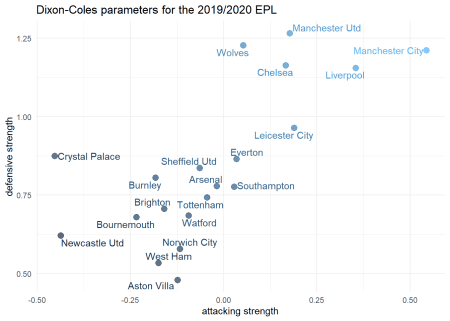
labs(title = "Dixon-Coles parameters for the 2019/2020 EPL",

x = "attacking strength",

y = "defensive strength") +

theme\_minimal()

p5

  
It might surprise some that Manchester City are predicted to be better than Liverpool by this model, but it shouldn’t given the underlying numbers for both teams. Liverpool have run very hot and Manchester City have run very cold this season.

Finally, we can then calculate the current Premier League table, and simulate remaining games to predict where teams will finish the season if the remainder of games were to be played. I’ve chosen 1000 sims just for sake of processing time, but you can scale up and down as desired.

#calculate the current EPL table

current\_epl\_table <- played\_matches %>%

select(home, away, hgoal, agoal) %>%

pivot\_longer(c(home, away), names\_to = "location", values\_to = "team") %>%

mutate(points = case\_when(

location == "home" & hgoal > agoal ~ 3,

location == "away" & agoal > hgoal ~ 3,

hgoal == agoal ~ 1,

TRUE ~ 0

)) %>%

mutate(gd = case\_when(

location == "home" ~ hgoal - agoal,

location == "away" ~ agoal - hgoal

)) %>%

group\_by(team) %>%

summarise(points = sum(points),

gd = sum(gd))

#the number of sims to run

n <- 10000

#simulate remaining matches

fixture\_sims\_2020 <- rerun(

n,

augment.dixoncoles(fit\_2020, unplayed\_matches, type.predict = "scorelines") %>%

mutate(sampled\_result = map(.scorelines, sample\_n, 1, weight = prob)) %>%

select(-.scorelines) %>%

unnest(cols = c(sampled\_result)) %>%

pivot\_longer(c(home, away), names\_to = "location", values\_to = "team") %>%

mutate(points = case\_when(

location == "home" & hgoal > agoal ~ 3,

location == "away" & agoal > hgoal ~ 3,

hgoal == agoal ~ 1,

TRUE ~ 0

)) %>%

mutate(gd = case\_when(

location == "home" ~ hgoal - agoal,

location == "away" ~ agoal - hgoal

)) %>%

select(team, points, gd))

#calculate final EPL tables

predicted\_finishes\_2020 <- map\_df(fixture\_sims\_2020, function(sim\_fixtures, table) {

sim\_fixtures %>%

select(team, points, gd) %>%

bind\_rows(., table) %>%

group\_by(team) %>%

summarise(points = sum(points),

gd = sum(gd)) %>%

arrange(-points, -gd) %>%

mutate(predicted\_finish = 1:n())

}, current\_epl\_table) %>%

group\_by(team, predicted\_finish) %>%

summarise(perc = n() / n) %>%

group\_by(team) %>%

mutate(mean\_finish = mean(predicted\_finish)) %>%

arrange(mean\_finish) %>%

ungroup() %>%

mutate(team = factor(team, levels = unique(team)))

If we then plot these predicted finishes (ordered by the chance of their highest finish position), we can get an idea of where we expect teams to end the season:

#list of team colours

team\_cols <- c("red", "skyblue", "darkblue", "darkblue", "darkred",

"orange", "red", "white", "red", "blue", "maroon",

"blue", "white", "red", "dodgerblue", "yellow",

"maroon", "red", "maroon", "yellow")

#plot the finishing position by chance based on these simualtions

p6 <- ggplot(predicted\_finishes\_2020,

aes(x = predicted\_finish, y = perc, fill = team)) +

geom\_bar(stat = "identity", colour = "black") +

scale\_fill\_manual(values = team\_cols, guide = FALSE) +

labs(

title = "Predicted finish position of teams",

subtitle = "for incomplete 2019/2020 EPL season",

y = "fraction of finishes",

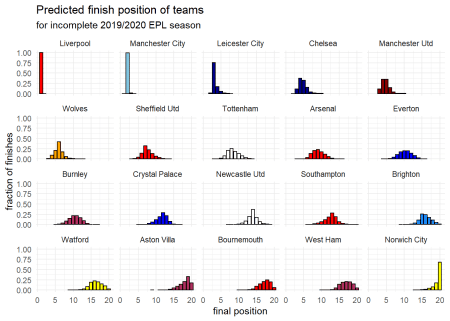
x = "final position"

) +

theme\_minimal() +

facet\_wrap(~team)

p6



So great news for Liverpool fans who the model believes have a 100% chance of finishing in first place. Leicester also might be happy with a nailed on 3rd place, with Chelsea or Manchester United probably rounding out the top four, and Wolves joining the loser of the two in the Europa League.

#get the predictions for the 2019/2020 champion

winner <- predicted\_finishes\_2020 %>%

filter(predicted\_finish < 2)%>%

mutate(prediction = "Champion chance")

winner

## # A tibble: 2 x 5

## team predicted\_finish perc mean\_finish prediction

##

## 1 Liverpool 1 1.00 1.5 Champion chance

## 2 Manchester City 1 0.0001 2.5 Champion chance

#get prediction for those who qualify for champions league

#and for europa league

champs\_league <- predicted\_finishes\_2020 %>%

filter(predicted\_finish < 5) %>%

group\_by(team) %>%

summarise(perc = sum(perc)) %>%

arrange(-perc) %>%

mutate(prediction = "Champions League chance")

champs\_league

## # A tibble: 10 x 3

## team perc prediction

##

## 1 Liverpool 1 Champions League chance

## 2 Manchester City 1 Champions League chance

## 3 Leicester City 0.933 Champions League chance

## 4 Chelsea 0.479 Champions League chance

## 5 Manchester Utd 0.46 Champions League chance

## 6 Wolves 0.106 Champions League chance

## 7 Sheffield Utd 0.0155 Champions League chance

## 8 Tottenham 0.004 Champions League chance

## 9 Arsenal 0.0018 Champions League chance

## 10 Everton 0.0005 Champions League chance

europa\_league <- predicted\_finishes\_2020 %>%

filter(predicted\_finish < 7) %>%

group\_by(team) %>%

summarise(perc = sum(perc)) %>%

arrange(-perc) %>%

mutate(prediction = "(at least) Europa League chance")

europa\_league

## # A tibble: 13 x 3

## team perc prediction

##

## 1 Liverpool 1 (at least) Europa League chance

## 2 Manchester City 1 (at least) Europa League chance

## 3 Leicester City 0.999 (at least) Europa League chance

## 4 Manchester Utd 0.954 (at least) Europa League chance

## 5 Chelsea 0.954 (at least) Europa League chance

## 6 Wolves 0.729 (at least) Europa League chance

## 7 Sheffield Utd 0.196 (at least) Europa League chance

## 8 Tottenham 0.096 (at least) Europa League chance

## 9 Arsenal 0.0479 (at least) Europa League chance

## 10 Everton 0.0139 (at least) Europa League chance

## 11 Burnley 0.0089 (at least) Europa League chance

## 12 Crystal Palace 0.0009 (at least) Europa League chance

## 13 Southampton 0.0008 (at least) Europa League chance

At the foot of the table, the model is fairly bullish on Norwich being relegated, with Aston Villa probably joining them, and then probably West Ham rounding out the relegation spots.

#get predictions for those who would be relegated

relegated <- predicted\_finishes\_2020 %>%

filter(predicted\_finish > 17) %>%

group\_by(team) %>%

summarise(perc = sum(perc)) %>%

arrange(-perc) %>%

mutate(prediction = "Relegation chance")

relegated

## # A tibble: 8 x 3

## team perc prediction

##

## 1 Norwich City 0.934 Relegation chance

## 2 Aston Villa 0.700 Relegation chance

## 3 Bournemouth 0.507 Relegation chance

## 4 West Ham 0.402 Relegation chance

## 5 Watford 0.270 Relegation chance

## 6 Brighton 0.171 Relegation chance

## 7 Newcastle Utd 0.0126 Relegation chance

## 8 Southampton 0.00270 Relegation chance

**Final Remarks**

I want to make it clear at the end of this post that this probably isn’t the most sophisticated model for predicting football matches (more to come in a part 3, maybe this time within less than a year), but does a pretty good job!