The School of Mathematics

THE UNIVERSITY
of EDINBURGH

Modelling soccer scores with Integrated Nested Laplace Approximation

by

Dongrui Shen

Dissertation Presented for the Degree of
MSc in Computational Applied Mathematics

August 2012

Supervised by
Dr Daniel Paulin

Abstract

Statistical modelling of sports data is a popular topic and many research have been produced to this aim, also with regard to football. Many different models are proposed to estimate the probability of win or lose for a team, or to predict the number of goals scored of a particular match. Recently INLA has been successfully used for predicting the scores of soccer matches using a Poisson likelihood model. The objective of this project is to further improve to this model, main approaches include: (1) incorporating additional open data that covers multiple seasons, (2) and using sophisticated random effect terms that allow for modelling the time dependence of the model parameters. In this report, we propose a Poisson hierarchical model, test its predictive performance on the data of English Premier League from 2011/12 to 2020/21 and compare it with the Poisson fixed-effect model.

Acknowledgments

I wish to acknowledge the help provided by the support staff in the School of Mathematics of the University of Edinburgh. I would also like to show my deep appreciation to my supervisors Dr. Daniel Paulin who helped me finalize my project.

Own Work Declaration

I declare that this thesis is an original report of my research, has been written by me and has not been submitted for any previous degree. The experimental work is almost entirely my own work; the collaborative contributions have been indicated clearly and acknowledged. Due references have been provided on all supporting literature and resources.

Contents

List of Tables

List of Figures

1 Introduction

The problem of modelling football data has become increasingly popular in the last few years and many different models have been proposed with the aim of estimating the characteristics that bring a team to lose or win a game, or to predict the score of a particular match [1]. From the statistical point of view, an interesting issue is the distribution form of the number of scores in a single game by the two teams. Although the binomial distribution has been proposed in the late 1970s [3], the Poisson distribution has been widely accepted as a suitable model to this aim. In addition, we often use a simplifying assumption of independence between the goals scored by home team and away team. For example, a model has been proposed with two independent Poisson variables where the model parameters are constructed as the product of attack strength and defense strength. Nevertheless, we can observe low levels of negative correlation between the two quantities from the empirical study. Consequently, a correlation factor could be added to the independent Poisson model to improve the predictive performance [4]. Also, more sophisticated models have been proposed. For example, the bivariate Poisson distribution allows for a more complex formulation for the likelihood function and an additional parameter explaining the covariance between the two quantities [6]. Another interesting issue is the way modelling football data. Other than modelling the outcomes or the number of goals scored, one can model the difference of the scores by the two teams. A Bayesian network are used to implement this model [7].

In this paper, we focus on the Bayesian hierarchical models for the number of goals scored by two teams in each match. Bayesian hierarchical models with latent Gaussian layers are flexible in capturing complex stochastic behavior and hierarchical structures in high dimensional spatial and spatio-temporal data [10]. Whereas Markov Chain Monte Carlo (MCMC) methods may be extremely slow and even become computationally unfeasible when dealing with such complex models, the inferential framework of Integrated Nested Laplace Approximation (INLA), proposed by Rue et al. [12], naturally accommodates hierarchical models and could be used to provide accurate and efficient estimations to the posterior distribution of interest. Many case studies have been conducted through INLA. Blangiardo and Cameletti reviews INLA in details and gives many practical examples in their monograph [2]. The basic idea of INLA is to perform Laplace approximation in a nested manner. It makes use of Gaussian-Markov dependence structure to break up posterior integration into a nested product of low dimensional integrals, which makes it applicable to high dimensional complex models. Besides, by assuming two conditionally independent Poisson variables, we have taken account of the correlation between home scores and away scores and hence a bivariate structure is unnecessary.

This paper is structured as follows: section 2 gives a short introduction to Bayesian inference and the INLA method; section 3 describes the data used; section 4 specifies the models, describes the model checks performed and displays the estimation result; Finally, section 5 concludes with discussion. Code for implementing all of the models is available on Github (https://github.com/grantaire08/modelling-football-scores-with-INLA).

2 Methodology

In this section, we provide a brief introduction to Bayesian inference and INLA that provides necessary notation and context for the developing and estimating of the models.

2.1 Bayesian inference

Bayesian inference is a method of statistical inference based on Bayes theorem:

Theorem 2.1 (Bayes theorem) *Assume A and B are events and $P(B) \neq 0$, the probability of B given A is as follows:*

$$Pr(B|A) = \frac{Pr(A|B)Pr(B)}{Pr(A)} \tag{2.1}$$

In the scenario of Bayesian Inference, we consider a random variable Y, modeled by a probability distribution and indexed by a generic parameter θ. Let:

$$L(\theta) = p(Y = y|\theta) \tag{2.2}$$

be the *likelihood* function which specifies the probability distribution of the data Y under the moel defined by θ. We assume the data y are randomly sampled from the interested population, which means the variability on y only depends on the sampling. As for the parameter θ, we choose a suitable *prior* probability distribution $p(\theta)$ to reflect our prior belief on θ.

Solving a Bayesian inference problem is to obtain the *posterior* distribution $p(\theta|y)$ of unknown parameters θ, whose prior distribution need to be stated. Applying the Bayes theorem, we get:

$$p(\theta|y) = \frac{p(y|\theta)p(\theta)}{p(y)} \tag{2.3}$$

We call $p(y)$ the *marginal* distribution of the data y, which could be obtained using the *formula of total probability* as follows:

$$p(y) = \sum_{\theta \in \Theta} p(y|\theta)p(\theta) \quad \text{or} \quad p(y) = \int_{\theta \in \Theta} p(y|\theta)p(\theta)\mathrm{d}\theta \tag{2.4}$$

which depends on whether y is discrete or continuous. With our assumption on the variability on y, $p(y)$ could be normally recognized as a normalization constants, so an alternative form of (2.3) is:

$$p(\theta|y) \propto p(y|\theta)p(\theta) \tag{2.5}$$

When performing Bayesian inference, the choice of prior distribution is an important concern. Normally, the type of distribution are expected to reflect the nature of parameters and the hyperparameters of prior distribution tune the level of information provided. When knowledge is limited, vague priors are assumed so that the posterior distribution is driven by the observed data. From the practical perspective, for most random choices of priors, we cannot get a closed form of the posterior distribution. In order to reduce the computation complexity, we usually choose the priors with good properties. To be precise, the conjugate priors, in which the prior is of the same form as the likelihood. Common inference models using conjugate priors includes: Binomial-Beta model, Gaussian-Gaussian model and Poisson-Gamma model.

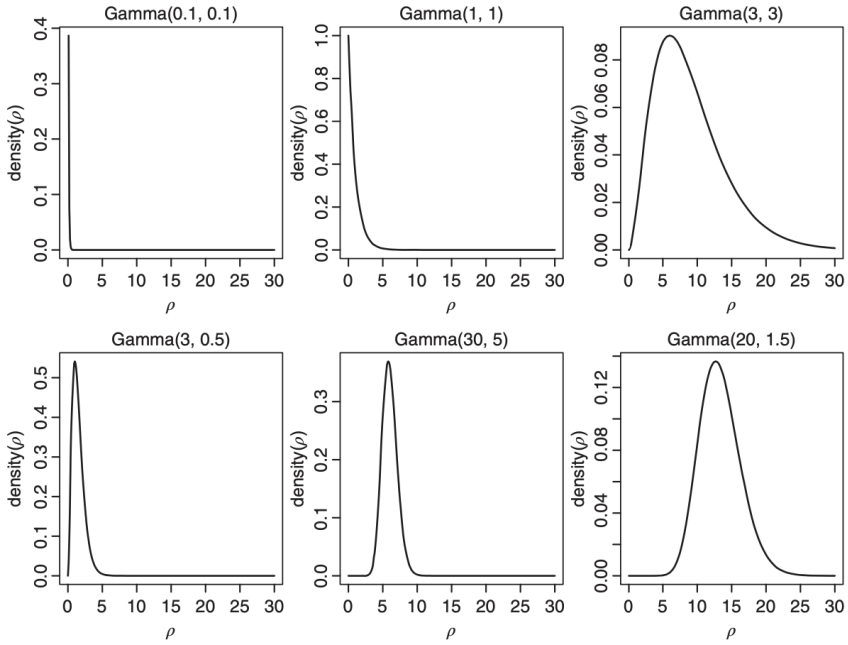


Figure 1: Density function for Gamma distributions with different values of the shape and scale hyperparameters [2].

In the study of football scores, we consider goal scoring as a counting process in which the data are the count of goals ($y \in N \cap [0, +\infty)$) . Hence, it can be modeled using a Poisson likelihood:

$$y|\lambda \sim Poisson(\lambda) \tag{2.6}$$

where $p(y|\lambda) = \frac{\lambda^y \exp(-\lambda)}{y!}$.

The conjugate prior of Poisson likelihood is the Gamma distribution:

$$\lambda \sim Gamma(a, b) \tag{2.7}$$

where $p(\lambda) = \frac{b^a}{\Gamma(a)} \lambda^{a-1} \exp(-b\lambda)$. The mean and variance of the Gamma distribution are a/b and a/b^2 respectively. By changing the values of hyperparameters a and b, we can modify the shape and scale of the distribution flexibly (see Fig. 1).

Combining the Poisson likelihood and Gamma prior, the posterior distribution is able to be expressed in the following closed form:

$$\begin{aligned} p(\lambda|y) &\propto \frac{b^a}{\Gamma(a)} \lambda^{a-1} \exp(-b\lambda) \frac{\lambda^y \exp(-\lambda)}{y!} \\ &\propto \lambda^{a+y-1} \exp(-(b+1)\lambda) \end{aligned} \tag{2.8}$$

which could be considered as a Gamma distribution, i.e., $\lambda|y \sim Gamma(a + y, b + 1)$. The corresponding posterior mean is:

$$E(\lambda|y) = \frac{a+y}{b+1} \tag{2.9}$$

The example above illustrate the convenience brought by conjugate models. Nevertheless, conjugacy does not hold in most practical cases. For example, for a generic parameter $\theta$and a

function $h(\cdot)$ of θ, the posterior mean of $h(\theta)$ is defined as:

$$E(h(\theta)|y) = \int_{\theta \in \Theta} h(\theta) p(\theta|y) \mathrm{d}\theta \tag{2.10}$$

Since the posterior $p(\theta|y)$ could be complex when conjugacy is not applicable, approximation methods may be necessary.

2.2 The integrated nested Laplace approximation

Some of the mainstream approaches to deal with the computational aspects of Bayesian inference are based on statistical simulations, which generate random values from a give density function by a computer, such as Monte Carlo (MC) and Markov chain Monte Carlo (MCMC).

MC method approximates integrals using the empirical average, which is known as Monte Carlo integration. For example, the posterior mean $E(h(\theta)|y)$ defined in (2.10) is approximated using $\frac{\sum_{i=1}^{m} h(\theta^{(i)})}{m}$, where $\{\theta^{(i)}\}_{i=1}^{m}$ is a sample of m independent values of θ. The implementation of MC methods just requires simulating values from given posterior distribution. Despite the simplicity, MC methods have some inevitable limitations in practical cases: First, it assume the posterior distribution is in a known form. Second, it is difficult to draw iid MC sample directly from the posterior distribution. In these situations, instead of simulating independent values from the posterior distribution, we draw a sample by running a Markov chain (which consists of correlated values), whose stationary distribution is the posterior density. This procedure combines MC integration with Markov Chains and is known as MCMC.

For a long time, MCMC methods have been seen as powerful tools for Bayesian inference. In particular, the Gibbs sampler and the MH algorithm allows implementing MCMC on very complex models (where the number of parameters is large or the posterior distributions are not in the closed form). Despite that, when models are complex or we deal with massive data sets, MCMC methods may be extremely slow and even become computationally unfeasible. Besides, when making inference with MCMC, particular attention is paid to the analysis of MCMC output in order to achieve convergence and find the best setting in terms of parameterization, prior distributions, initial values, and MH proposal distributions.

Recently, there is an alternative to MCMC, the integrated nested Laplace approximation (INLA) , proposed by Rue et al. [12], which could reduce the computational costs of Bayesian inference. Unlike simulation-based MC and MCMC, INLA is a deterministic algorithm for Bayesian inference. The numerical integration method Laplace approximation is adopted by INLA. INLA is especially designed for latent Gaussian models, and outperforms MCMC in terms of both accuracy and efficiency [2]. A recent review on INLA can be found in the Rue et al.'s work in 2017 [13].

2.2.1 The core of INLA: Laplace approximation

Suppose we are interested in computing the following integral:

$$\int f(x)\mathrm{d}x = \int \exp(\log f(x))\mathrm{d}x \tag{2.11}$$

where $f(x)$ is the density function of a random variable X. The second order Taylor expansion of $\log f(x)$ at $x = x_0$ is:

$$\log f(x) \approx \log f(x_0) + (x - x_0)\frac{\partial \log f(x)}{\partial x}|_{x=x_0} + \frac{(x - x_0)^2}{2}\frac{\partial^2 \log f(x)}{\partial x^2}|_{x=x_0} \tag{2.12}$$

If x_0 is set to be the mode $x^* = \mathrm{argmax}_x \log f(x)$, then $\frac{\partial \log f(x)}{\partial x}|_{x=x^*} = 0$ and the expansion above becomes:

$$\log f(x) \approx \log f(x^*) + \frac{(x - x^*)^2}{2}\frac{\partial^2 \log f(x)}{\partial x^2}|_{x=x*} \tag{2.13}$$

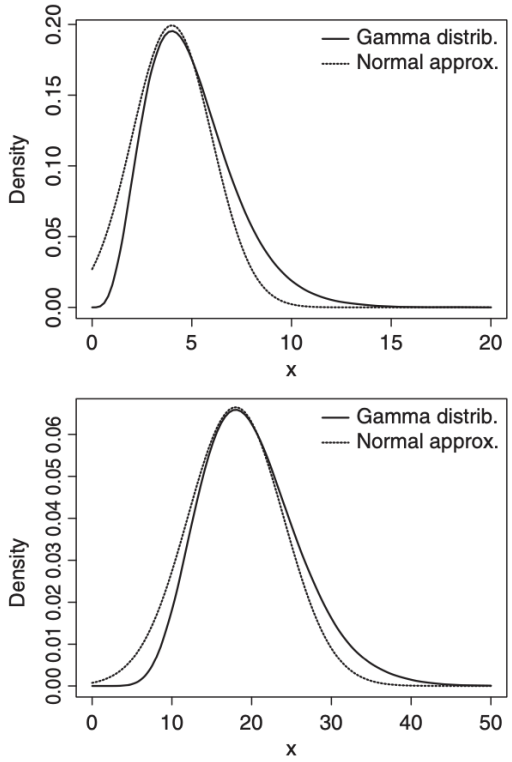


Figure 2: The Laplace Approximation of Gamma distribution is Gaussian [2].

The integral of interest is then approximated as follows:

$$\begin{aligned}\int f(x)\mathrm{d}x &\approx \int \exp(\log f(x^*) + \frac{(x-x^*)^2}{2}\frac{\partial^2 \log f(x)}{\partial x^2}|_{x=x^*})\mathrm{d}x \\ &= \exp(\log f(x^*)) \int \exp(\frac{(x-x^*)^2}{2}\frac{\partial^2 \log f(x)}{\partial x^2})\mathrm{d}x\end{aligned} \tag{2.14}$$

By setting $\sigma^{*2} = 1/\frac{\partial^2 \log f(x)}{\partial x^2}$, the approximation above becomes:

$$\int f(x)\mathrm{d}x \approx \exp(\log f(x^*)) \int \exp(-\frac{(x-x^*)^2}{2\sigma^{*2}})\mathrm{d}x \tag{2.15}$$

where the integrand can be associated with a Gaussian distribution $N(x^*, \sigma^{*2})$. Hence, the integral above can be approximated explicitly:

$$\int_\alpha^\beta f(x)\mathrm{d}x \approx f(x^*)\sqrt{2\pi\sigma^{*2}}(\Phi(\beta) - \Phi(\alpha)) \tag{2.16}$$

where $\Phi(\cdot)$ is the cumulative density function of the Gaussian distribution $N(x^*, \sigma^{*2})$. As a special case, the Laplace approximation of $Gamma(a, b)$ is $N(\frac{a-1}{b}, \frac{a-1}{b^2})$ (see Fig. 2).

2.2.2 INLA setting: latent Gaussian models

We denote the observed data by $\boldsymbol{y} = (y_1, \cdots, y_n)$. Assume the distribution $p(y_i|\phi_i)$ of y_i, is characterized by a parameter ϕ_i defined as a function of a structured linear predictor η_i through a link function $g(\cdot)$ (such as the identity function and the log function). The linear predictor η_i

has the following form:

$$\eta_i = \beta_0 + \sum_{m=1}^{M} \beta_m x_i^{(m)} + \sum_{i=1}^{L} f_l(z_i^{(l)}) \tag{2.17}$$

Here, β_0 is the intercept; $\sum_{m=1}^{M} \beta_m x_i^{(m)}$ is the linear combination of some covariates $\boldsymbol{x_i} = (x_i^{(1)}, \cdots, x_i^{(M)})$ with coefficients $\boldsymbol{\beta} = (\beta_1, \cdots, \beta_M)$, which are called *fixed effects*; $\sum_{i=1}^{L} f_l(z_i^{(l)})$ is the sum of some smooth nonlinear effects $\boldsymbol{f} = \{f_1(\cdot), \cdots, f_L(\cdot)\}$ on covariates $\boldsymbol{z} = (z_1, \cdots, z_L)$; and these nonlinear effects could be time trends and seasonal effects, random intercept and slopes as well as spatio or spatial random effects.

A simplest example of the general representation above is the linear regression model. In the linear regression assumes $y_i = \beta_0 + \sum_{m=1}^{M} \beta_m x_i^{(m)} + \varepsilon_i$, where ε_i is Gaussian noise drawn from $N(0, \sigma^2)$.

Another example are generalized linear models (GLMs), such as Poisson regression model. Poisson regression model assumes that $y_i|\boldsymbol{x_i}, \beta_0, \boldsymbol{\beta} \sim Poisson(\lambda)$, where the mean of the predicted Poisson distribution λ is given by:

$$\log(\lambda_i) = \eta_i = \log(E(y_i|\boldsymbol{x_i}, \beta_0, \boldsymbol{\beta})) = \beta_0 + \sum_{m=1}^{M} \beta_m x_i^{(m)} \tag{2.18}$$

To simplify the representation, the above could be rewrite as $\eta_i = \boldsymbol{\beta} \cdot \boldsymbol{x_i}$.

After specifying a distribution for the observed data in the general form stated above, we collect all the latent components of interest for the inference in a generic parameter $\boldsymbol{\theta}$ called a latent field, defined as $\boldsymbol{\theta} = \{\beta_0, \boldsymbol{\beta}, \boldsymbol{f}\}_{i=1}^{n}$. Further, we denote with $\boldsymbol{\psi} = \{\psi_1, \cdots, \psi_K\}$ the vector of the K hyperparameters. Under the assumption of conditional independence, the distribution of the n observations is given by the likelihood:

$$p(\boldsymbol{y}|\boldsymbol{\theta}, \boldsymbol{\psi}) = \prod_{i=1}^{n} p(y_i|\theta_i, \boldsymbol{\psi}) \tag{2.19}$$

We assume a multivariate Gaussian prior $N(0, \boldsymbol{\Sigma^{-1}(\psi)})$ on the latent field θ (i.e. *latent Gaussian field*) and hence the density is given by:

$$p(\theta|\psi) = \frac{1}{\sqrt{(2\pi)^n/|\boldsymbol{\Sigma(\psi)}|}} \exp(-\frac{1}{2}\boldsymbol{\theta^T \Sigma(\psi)\theta}) \tag{2.20}$$

Note the conditional independence between components of the latent Gaussian field $\boldsymbol{\theta}$. Suppose the components θ_i and θ_j are conditionally independent given all the other components $\theta_{-i,j}$, their joint conditional distribution can be factorized as follows:

$$p(\theta_i, \theta_j|\theta_{-i,j}) = p(\theta_i|\theta_{-i,j})p(\theta_j|\theta_{-i,j}) \tag{2.21}$$

which are denoted as $\theta_i \perp \theta_j|\theta_{-i,j}$. For any $i \neq j$, it can be concluded that:

$$\theta_i \perp \theta_j|\theta_{-i,j} \Leftrightarrow \Sigma_{ij}(\boldsymbol{\psi}) = 0 \tag{2.22}$$

where $\Sigma_{ij}(\boldsymbol{\psi})$ is the entry in the $i-$th row and $j-$th column of the precision matrix $\boldsymbol{\Sigma(\psi)}$. Hence, the precision matrix is sparse which allows for computational benefits when making inference. The specification of conditional independence is known as *Gaussian Markov random*

field (GMRF) [2].

To sum up, the specification above defines a latent Gaussian model (LGM):

$$\begin{aligned} \boldsymbol{\psi} &\sim \pi(\boldsymbol{\psi}) && \text{(Hyperprior)} \\ \boldsymbol{\theta}|\boldsymbol{\psi} &\sim N(0, \boldsymbol{\Sigma}(\boldsymbol{\psi})) && \text{(GMRF)} \\ \boldsymbol{y}|\boldsymbol{\theta}, \boldsymbol{\psi} &\sim p(\boldsymbol{y}|\boldsymbol{\theta}, \boldsymbol{\psi}) = \prod_{i=1}^{n} p(y_i|\theta_i, \boldsymbol{\psi}) && \text{(Likelihood)} \end{aligned}$$

The joint posterior distribution of $\boldsymbol{\theta}$ and $\boldsymbol{\psi}$ is:

$$\begin{aligned} p(\boldsymbol{\theta}, \boldsymbol{\psi}|\boldsymbol{y}) &\propto p(\boldsymbol{\psi}) \cdot p(\boldsymbol{\theta}|\boldsymbol{\psi}) \cdot p(\boldsymbol{y}|\boldsymbol{\theta}, \boldsymbol{\psi}) \\ &\propto p(\boldsymbol{\psi}) \cdot p(\boldsymbol{\theta}|\boldsymbol{\psi}) \cdot \prod_{i=1}^{n} p(y_i|\theta_i, \boldsymbol{\psi}) \\ &\propto p(\boldsymbol{\psi}) \cdot |\boldsymbol{\Sigma}(\boldsymbol{\psi})|^{1/2} \exp(-\frac{1}{2}\boldsymbol{\theta}^{\boldsymbol{T}}\boldsymbol{\Sigma}(\boldsymbol{\psi})\boldsymbol{\theta}) \cdot \prod_{i=1}^{n} \exp(\log(p(y_i|\theta_i, \boldsymbol{\psi}))) \\ &\propto p(\boldsymbol{\psi}) \cdot |\boldsymbol{\Sigma}(\boldsymbol{\psi})|^{1/2} \exp(-\frac{1}{2}\boldsymbol{\theta}^{\boldsymbol{T}}\boldsymbol{\Sigma}(\boldsymbol{\psi})\boldsymbol{\theta} + \sum_{i=1}^{n} \log(p(y_i|\theta_i, \boldsymbol{\psi}))) \end{aligned} \tag{2.23}$$

In addition, to simplify the discussion, we assume the likelihood is the density for a univariate distribution (only one target variable) and each observed data y_i corresponds to only one element θ_i in the latent field. Martins er al. have discussed the possibility of connecting each observation to a linear combination of elements in θ and take account the case when the data belong to several distributions, i.e. the multiple likelihoods case [9] .

2.2.3 Inference with INLA

Recall from Sec. 2.1, Bayesian inference is about obtaining the posterior distributions. To be precise, we are interested in the marginal posterior distribution for each element of parameter vector $p(\theta_i|y)$ and for each element of the hyper parameter vector $p(\psi_k|y)$.

Rewrite these interested marginal posteriors in the form of integrals, we get:

$$p(\theta_i|\boldsymbol{y}) = \int p(\theta_i, \boldsymbol{\psi}|\boldsymbol{y})\mathrm{d}\boldsymbol{\psi} = \int p(\theta_i|\boldsymbol{\psi}, \boldsymbol{y})p(\boldsymbol{\psi}|\boldsymbol{y})\mathrm{d}\boldsymbol{\psi} \tag{2.24}$$

$$p(\psi_k|\boldsymbol{y}) = \int p(\boldsymbol{\psi}|\boldsymbol{y})\mathrm{d}\boldsymbol{\psi}_{-k} \tag{2.25}$$

Hence, we need first compute $p(\boldsymbol{\psi}|\boldsymbol{y})$ to obtain all the $p(\psi_k|\boldsymbol{y})$. Together with $p(\theta_i|\boldsymbol{\psi}, \boldsymbol{y})$, we can compute all the $p(\theta_i|y)$.

The posterior distribution of the hyperparameter $\boldsymbol{\psi}$ is:

$$\begin{aligned} p(\psi|\boldsymbol{y}) &\propto \int p(\theta, \psi|\boldsymbol{y})\mathrm{d}\theta \\ &\propto \int p(\boldsymbol{y}|\boldsymbol{\theta}, \boldsymbol{\psi})p(\theta)p(\psi)\mathrm{d}\theta \end{aligned} \tag{2.26}$$

To avoid the integration, we use the following representation instead:

$$\begin{aligned} p(\boldsymbol{\psi} \mid \boldsymbol{y}) &= \frac{p(\boldsymbol{\theta}, \boldsymbol{\psi} \mid \boldsymbol{y})}{p(\boldsymbol{\theta} \mid \boldsymbol{\psi}, \boldsymbol{y})} \\ &= \frac{p(\boldsymbol{y} \mid \boldsymbol{\theta}, \boldsymbol{\psi}) p(\boldsymbol{\theta}, \boldsymbol{\psi})}{p(\boldsymbol{y})} \frac{1}{p(\boldsymbol{\theta} \mid \boldsymbol{\psi}, \boldsymbol{y})} \\ &= \frac{p(\boldsymbol{y} \mid \boldsymbol{\theta}, \boldsymbol{\psi}) p(\boldsymbol{\theta} \mid \boldsymbol{\psi}) p(\boldsymbol{\psi})}{p(\boldsymbol{y})} \frac{1}{p(\boldsymbol{\theta} \mid \boldsymbol{\psi}, \boldsymbol{y})} \\ &\propto \frac{p(\boldsymbol{y} \mid \boldsymbol{\theta}, \boldsymbol{\psi}) p(\boldsymbol{\theta} \mid \boldsymbol{\psi}) p(\boldsymbol{\psi})}{p(\boldsymbol{\theta} \mid \boldsymbol{\psi}, \boldsymbol{y})} \\ &\approx \left. \frac{p(\boldsymbol{y} \mid \boldsymbol{\theta}, \boldsymbol{\psi}) p(\boldsymbol{\theta} \mid \boldsymbol{\psi}) p(\boldsymbol{\psi})}{\tilde{p}(\boldsymbol{\theta} \mid \boldsymbol{\psi}, \boldsymbol{y})} \right|_{\boldsymbol{\theta} = \boldsymbol{\theta}^*(\boldsymbol{\psi})} =: \tilde{p}(\boldsymbol{\psi} \mid \boldsymbol{y}) \end{aligned} \tag{2.27}$$

where $\tilde{p}(\cdot)$ is the Laplace approximation. In fact, $p(\boldsymbol{\theta}|\boldsymbol{\psi}, \boldsymbol{y})$ is almost Gaussian under the assumption of GMRF. Since the observed data $\boldsymbol{y}$ is generally uninformative, we can use the GMRF $p(\boldsymbol{\theta}|\boldsymbol{\psi})$ to approximate $p(\boldsymbol{\theta}|\boldsymbol{\psi}, \boldsymbol{y})$. It turns out that the Laplace approximation is of a high level of accuracy.

In the next, we rewrite $p(\theta_i|\boldsymbol{\psi}, \boldsymbol{y})$ in a similar way:

$$\begin{aligned} p\left(\theta_i \mid \boldsymbol{\psi}, \boldsymbol{y}\right) &= \frac{p\left((\theta_i, \boldsymbol{\theta}_{-i}) \mid \boldsymbol{\psi}, \boldsymbol{y}\right)}{p\left(\boldsymbol{\theta}_{-i} \mid \theta_i, \boldsymbol{\psi}, \boldsymbol{y}\right)} \\ &= \frac{p(\boldsymbol{\theta}, \boldsymbol{\psi} \mid \boldsymbol{y})}{p(\boldsymbol{\psi} \mid \boldsymbol{y})} \frac{1}{p\left(\boldsymbol{\theta}_{-i} \mid \theta_i, \boldsymbol{\psi}, \boldsymbol{y}\right)} \\ &\propto \frac{p(\boldsymbol{\theta}, \boldsymbol{\psi} \mid \boldsymbol{y})}{p\left(\boldsymbol{\theta}_{-i} \mid \boldsymbol{\theta}_i, \boldsymbol{\psi}, \boldsymbol{y}\right)} \\ &\approx \left. \frac{p(\boldsymbol{\theta}, \boldsymbol{\psi} \mid \boldsymbol{y})}{\tilde{p}\left(\boldsymbol{\theta}_{-i} \mid \theta_i, \boldsymbol{\psi}, \boldsymbol{y}\right)} \right|_{\boldsymbol{\theta}_{-i} = \boldsymbol{\theta}^*_{-i}(\theta_i, \boldsymbol{\psi})} =: \tilde{p}\left(\theta_i \mid \boldsymbol{\psi}, \boldsymbol{y}\right) \end{aligned} \tag{2.28}$$

Again, since $\boldsymbol{\theta_{-i}}|\theta_i, \boldsymbol{\psi}, \boldsymbol{y}$ are generally normal distributed, the Laplace approximation here is very accurate. Although it could be computationally expensive compared to the previous one. Some modifications to Laplace approximation could help reduce the computational costs here. Another faster alternative method to approximate $p(\theta_i|\boldsymbol{\psi}, \boldsymbol{y})$ is *simplified Laplace approximation*. This is based on a third order Taylor expansion of both the numerator and denominator of $p(\theta_i|\boldsymbol{\psi}, \boldsymbol{y})$. It effectively corrects the Laplace approximation of $p(\theta_i|\boldsymbol{\psi}, \boldsymbol{y})$ for location and skewness, leading to better accuracy. In practice, simplified Laplace approximation is a prior option compared to the full Laplace approximation [12].

Combining the approximated $p(\boldsymbol{\theta}|\boldsymbol{\psi}, \boldsymbol{y})$ And $p(\theta_i|\boldsymbol{\psi}, \boldsymbol{y})$, the marginal posterior distribution $p(\theta_i|y)$ are approximated by:

$$\tilde{p}\left(\theta_i \mid \boldsymbol{y}\right) \approx \int \tilde{p}\left(\theta_i \mid \boldsymbol{\psi}, \boldsymbol{y}\right) \tilde{p}(\boldsymbol{\psi} \mid \boldsymbol{y}) \mathrm{d}\boldsymbol{\psi} \tag{2.29}$$

Note that the integral can be solved through a finite weighted sum of values at some relevant integration points $\{\boldsymbol{\psi}^{(j)}\}$ with the corresponding weights $\{\Delta_i\}$.

Generally, INLA works in the following steps:

- Explore the marginal of hyperparameters $p(\boldsymbol{\psi}|\boldsymbol{y})$;
  - Find the mode $\boldsymbol{\psi}^*$ of $\tilde{p}(\boldsymbol{\psi}|\boldsymbol{y})$ by optimizing $\log \tilde{p}(\boldsymbol{\psi}|\boldsymbol{y})$;

- Compute the negative Hessian H at the mode;
- Reparameterize the $\boldsymbol{\psi}-$Space by defining the new variable z that satisfies $\boldsymbol{\psi(z)} = \boldsymbol{\psi^*}+\boldsymbol{V\Gamma^{1/2}z}$ to standardize the variables, improve conditioning and simply numerical integration;
- Detect good integration points $\{\boldsymbol{\psi}^{(j)}\}$ using grid strategy or CCD strategy [12];
- Obtain the marginal of hyperparameters $p(\boldsymbol{\psi}|\boldsymbol{y})$ by interpolation using the values of $\tilde{p}(\boldsymbol{\psi^{(j)}}|\boldsymbol{y})$ and $\{\boldsymbol{\psi}^{(j)}\}$;

- For each value in $\{\boldsymbol{\psi}_k^{(j)}\}$, evaluating the values of $\tilde{p}(\boldsymbol{\theta_i}|\boldsymbol{\psi^{(j)}}, \boldsymbol{y})$ on a grid of selected values for θ_i; and obtain the marginals of parameters $p(\boldsymbol{\theta}|\boldsymbol{y})$ in (2.29) by numerical integration.

In the following, we will illustrate how the INLA works through a toy example from [2]. Suppose the observed data $\boldsymbol{y} = (y_1, \cdots, y_n)$ are normally distributed with $y_i|\mu, \sigma^2 \sim N(\mu, \sigma^2)$ and independent with each other. Priors for μ and $\psi = 1/\sigma^2$ are:

$$\mu \sim N(\mu_0, \sigma_0^2), \psi \sim Gamma(a, b) \tag{2.30}$$

Using the LGM notation, we get:

$$\begin{aligned} \psi &\sim Gamma(a, b) \\ \theta &\sim N(\mu_0, \sigma_0^2) \\ \boldsymbol{y}|\theta, \psi &\sim \prod_{i=1}^{n} N(\theta, 1/\psi) \end{aligned} \tag{2.31}$$

where $\theta = \eta_i = \mu$ and $\psi = 1/\sigma^2$.

First, explore the marginal of hyperparameters $p(\psi|y)$:

$$\begin{aligned} p(\psi|\mathrm{y}) &= \frac{p(\theta, \psi|\boldsymbol{y})}{p(\theta|\psi, \boldsymbol{y})} \\ &\propto \frac{p(\boldsymbol{y}|\boldsymbol{\theta}, \boldsymbol{\psi})p(\theta)p(\psi)}{p(\theta|\psi, \boldsymbol{y})} \end{aligned} \tag{2.32}$$

Normally, the denominator $p(\theta|\psi, \mathrm{y})$ is approximated by the Laplace approximation. In this particular case with normally distributed observations, we have:

$$p(\theta|\psi, \mathrm{y}) \propto p(\mathrm{y}|\theta, \psi) \cdot p(\theta) \sim N(\theta_n, \sigma_n^2) \tag{2.33}$$

where $\theta_n = \frac{\psi \sum_{i=1}^{n} y_i + \mu_0/\sigma_0^2}{n\psi + 1/\sigma_0^2}$ and $\sigma_n^2 = \frac{1}{n\psi + 1/\sigma_0^2}$.

Further, since the terms depending on θ in the numerator and denominator of (2.32) have to cancel out, we can fix the value of θ to be any arbitrary value. Let $\theta = \theta_n$, we get:

$$p(\psi|\boldsymbol{y}) \propto \frac{p(\boldsymbol{y}|\boldsymbol{\theta}, \boldsymbol{\psi})p(\theta)p(\psi)}{p(\theta|\psi, \boldsymbol{y})}|_{\theta=\theta_n} = \frac{1}{\sqrt{2\pi\sigma_n^2}} p(\boldsymbol{y}|\boldsymbol{\theta}, \boldsymbol{\psi})p(\theta)p(\psi)|_{\theta=\theta_n} \tag{2.34}$$

In order to evaluate the posterior $p(\psi|\boldsymbol{y})$, some values for ψ are chosen, denoted as $\{\psi^{(j)}\}$. Here we simply choose the most probable values for ψ without using the gird strategy and CCD

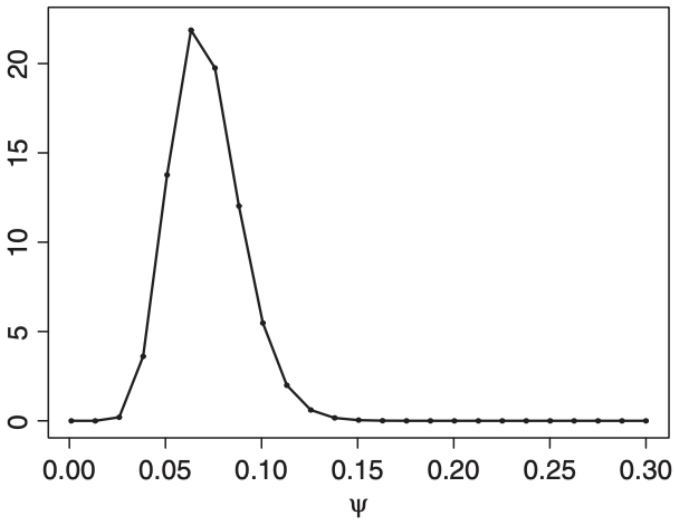


Figure 3: The normalized posterior distribution for ψ. The distribution is skewed [2].

strategy. At $\psi = \psi^{(j)}$, the density is computed:

$$p(\psi^{(j)}|\boldsymbol{y}) \propto \frac{1}{\sqrt{2\pi\sigma_n^2}} p(\boldsymbol{y}|\theta_n, 1/\psi^{(j)}) p(\theta_n) p(\psi^{(j)}) \tag{2.35}$$

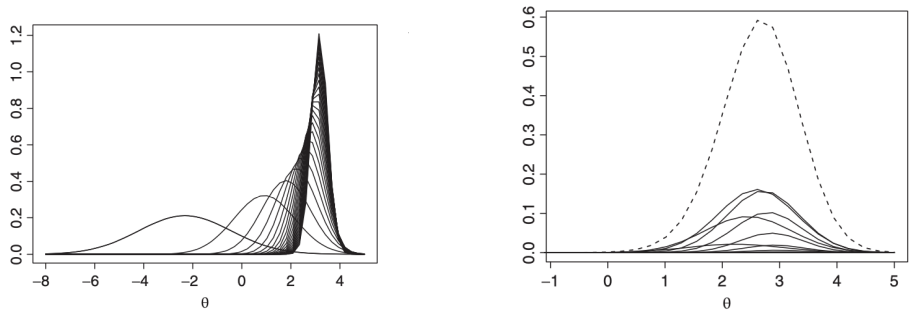
where $\theta_n = \frac{\psi^{(j)} \sum_{i=1}^n y_i + \mu_0/\sigma_0^2}{n\psi^{(j)} + 1/\sigma_0^2}$ and $\sigma_n^2 = \frac{1}{n\psi^{(j)} + 1/\sigma_0^2}$.

Fig. 3 demonstrates the normalized posterior distribution for ψ. The obvious skewness of the distribution is an effect of not reparamterizing the $\psi-$space into some normal-shaped distribution.

Next, we evaluate the full conditional normal distribution $p(\theta|\psi, \boldsymbol{y})$ for each value of ψ in $\{\psi^{(j)}\}$ and of θ in $\{\theta^{(l)}\}$. The marginal of parameter $p(\theta|y)$ can be obtained by the following numerical integration:

$$p(\theta^{(l)}|\boldsymbol{y}) \propto \sum_j p(\theta^{(l)}|\psi^{(j)}, \boldsymbol{y}) p(\psi^j|\boldsymbol{y}) \Delta_j \tag{2.36}$$

where we set $\Delta_j = \Delta = \frac{1}{\sum_j p(\psi^{(j)}|\boldsymbol{y})}$ (see Fig. 4).



(a) (b)

Figure 4: Left: the conditional distributions of $\theta|\psi^{(j)}, \boldsymbol{y}$ for each value of $\{\psi^{(j)}\}$; Right: The solid curve represent the weight joint posterior distribution $p(\theta^{(l)}|\psi^{(j)}, \boldsymbol{y}) p(\psi^{(j)}|\boldsymbol{y}) \Delta_j$; the dashed curve is the posterior distributions $p(\theta|\boldsymbol{y})$ [2].

2.3 The R-INLA package

INLA implemented as an R package is called **R-INLA**. We will use a simple Poisson regression example to introduce the **R-INLA** syntax. The INLA method for Bayesian inference can be applied using the command:

```
> output <- inla(formula, data, family, ...)
```

where **formula** specifies the linear predictor; **data** contains the dataframe including the observations and covariates; and **family** defines the likelihood model adopted.

Consider the case of n observed data $\boldsymbol{y}$, two covariates $(\boldsymbol{x_1}, \boldsymbol{x_2})$ and a function $f(\boldsymbol{z})$ of covariate $\boldsymbol{z}$, then the linear predictor $\eta_i = \beta_0 + \beta_1 x_i^{(1)} + \beta_2 x_i^{(2)} + f(z_i)$ are specified in **R-INLA** through:

```
> formula <- y ~ 1 + x1 + x2 + f(z, model="...")
```

model="..." in the function $f(\cdot)$ specifies the type of $f(\cdot)$ function. Generally, the type of covariate z must be integer or factor (When z is a factor variable, the corresponding levels need to be passed by specifying **values=...**). The default choice is **model="iid"** which the random variable indexed by z are independent and Gaussian distributed. We can check all available latent effects by typing the command below (see details of each of them at http://www.r-inla.org/models/latent-models):

```
> names(inla.models()\$latent)
```

There are some other options can be specified in the $f(\cdot)$ function, such as **hyper**, **constr**, **diagonal** and etc. All available likelihood distributions can be found using the command below see details of each of them at http://www.r-inla.org/models/likelihoods.):

```
> names(inla.models()\$likelihood)
```

By setting **family="poisson"**, we could adopt a poisson regression model. Finally, we run the INLA algorithm using the **inla** function:

```
> output <- inla(formula, family="poisson", data=df)
```

The **inla** function returns an object of class **inla**, which is a list consists of many information about the trained model. One of the most common command to check the result is:

```
> summary(output)
Call:
   "inla(formula = formula, family = \"poisson\", data = df)"
Time used:
    Pre = 9.34, Running = 0.613, Post = 0.237, Total = 10.2
Fixed effects:
                   mean    sd  0.025quant  0.5quant  0.975quant  mode  kld
(Intercept)       0.079 0.212    -0.349     0.084       0.483   0.092    0
x1Aston Villa     0.008 0.191    -0.367     0.008       0.382   0.008    0
x1Brighton       -0.312 0.208    -0.724    -0.310       0.093  -0.308    0
x1Burnley        -0.495 0.220    -0.935    -0.493      -0.069  -0.488    0
x1Chelsea         0.050 0.188    -0.319     0.050       0.421   0.050    0
x1Crystal Palace -0.266 0.207    -0.676    -0.265       0.136  -0.263    0
                                    ... ...
x2Aston Villa     0.165 0.218    -0.260     0.165       0.595   0.163    0
x2Brighton        0.150 0.218    -0.276     0.149       0.580   0.148    0
x2Burnley         0.322 0.210    -0.085     0.321       0.737   0.318    0
```

```
x2Chelsea         -0.077 0.231   -0.532    -0.076       0.376 -0.075    0
x2Crystal Palace  0.513 0.202    0.122     0.511       0.916  0.507    0
                                    ... ...
Random effects:
  Name    Model
    z IID model

Model hyperparameters:
                  mean   sd  0.025quant  0.5quant  0.975quant  mode
Precision for z 18660.63 17200.12  487.41 13453.85 64801.67 112.62

Expected number of effective parameters(stdev): 39.39(1.36)
Number of equivalent replicates : 19.30

Marginal log-Likelihood:  -1298.75
```

As above, we display the result of a Poisson mixed effects model used for football scores prediction. There are some important statics such as computing times, statics of posterior distributions for fixed effects and for the hyperparameters of random effects. Further, `R-INLA` provides an estimated of the effective number of parameters which can be used as a measure of the model complexity. The number of equivalent replicates can be considered as the average number of observations available to estimate each parameter in the model. `R-INLA` computes the marginal log-likelihood of the model by default, which could be used as a model comparison criterion.

3 Data exploration and preprocessing

Before developing and estimating the models, it is useful to have a brief introduction to the competition format of English Premier League. In this section, we introduce the data sets used and the necessary preprocessing performed. In addition, we perform a basic analysis of the data sets to summarize the main characteristics.

3.1 Data description

English Premier League (EPL) is the top level of the English football league system and it consists of 20 teams. The Premier League season goes from August to May and involves the teams playing each other home and away across the season, a total of 380 matches. The league began in 1992 and has seen seven different winners: Manchester United, Arsenal, Chelsea, Manchester City, Blackburn Rovers, Leicester City and Liverpool. Man United have had the most success with 13 titles in the 28 seasons so far. Man City have the Premier League record for the biggest winning margin, when they finished19 points ahead of second-placed Manchester United in 2017/18. The English Premier League operates on a system of promotion and relegation with the English Football League. Three points are awarded for a win, one point for a draw and none for a lose. The team with the most points at the end of the season wins the Premier League title. If any clubs finish with the same number of points, their position in the Premier League table is determined by goal difference, then the number of goals scored. If the teams still cannot be separated, they will be awarded the same position in the table. The teams that finish in the bottom three of the league table at the end of the campaign are relegated to the Championship, the second tier of English football.

We use the English Premier League data set from 2011 to present which contains 10 seasons covering 35 clubs. Normally, seasons run from August to May contested with 20 teams and each team plays 38 rounds, i.e. 38 matches. The 2019/20 season is an exception. COVID-19 resulted in a three-month lay-off of the Premier League since March 2020. The data can be found at https://www.football-data.co.uk/. The original data set consists of 70 variables, including basic game information(e.g. date, time, home team, away team and scores), some match statistics and some betting odds data from bookmakers. Since we are not interested in analyzing either the detailed game performance or the information from bookmakers, we only keep the basic game information. Besides, we expect that our model could reflect the effect of important events, for example, the arrival of new managers. An additional data set of Premier League Managers (see https://en.wikipedia.org/wiki/List_of_Premier_League_managers) is used for this purpose.

3.2 Data preparation

We prepare the data by three steps: first, tidy up the data; second, restructure the data; finally, merge the Premier League Managers data with the English Premier League data.

Restructuring the data is most important step. Observing the original data, we find the target variable team score distributing as two variables, home team score and away team score, in the original data which is not applicable to models with one output. Hence, in the new data, we record each game using two rows; the first row takes home team as attack team and away team as defense team and the second row does the opposite. The score of attack team is the

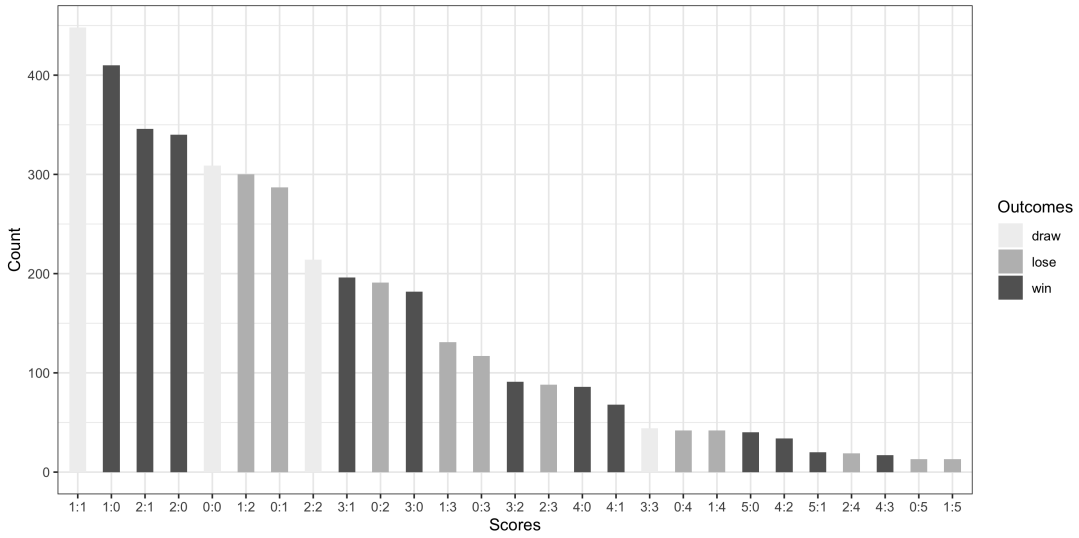


Figure 5: The frequency of match outcomes in the EPL data (2011-2021). The most frequent outcomes are draw and home win.

target variable. Note that the original data can be applied to models with multiple outputs (e.g. bivariate Poisson model) which is not the focus of this study.

In addition, we find some anomalies in the data to make note of. The dates in the original data are stored as strings in two formats, "dd/mm/yyyy" and "dd/mm/yy" respectively. They need to be transformed to Date variables with the uniform format, "yyyy-mm-dd". Besides, the English Premier League data uses the abbreviation of club names while the Premier League Managers data uses the full name. Extra attention should be paid when merging the two data sets.

3.3 Basic analysis of English Premier League data

First, we check the most frequent scoreline in the EPL data (see Fig. 5) which turns out to be 1:1, a draw. The top 5 most frequent scoreline are home win or draw, implying the existence of home field advantages. Fig. 6 shows the distribution of scores and goals difference respectively. We find most football matches have less than 3 scores, and the situation of zero goal is very common. Also, the distribution goals difference is almost symmetrical around 0 with slight right skewness. Despite that the final outcome of the majority of football matches is uncertain until the end, such typical distribution characteristics of scores make it possible to make the prediction based on goals alone. In Fig. 7, we check the number of goals scored by different teams. We can find that the teams with the highest points also scored the highest number of goals. Manchester City and Liverpool occupy the two top leading spots and they also scored the most goals. Clubs that have been relegated from the EPL including Middlesbrough, Reading, Bolton and etc. score the lowest number of goals.

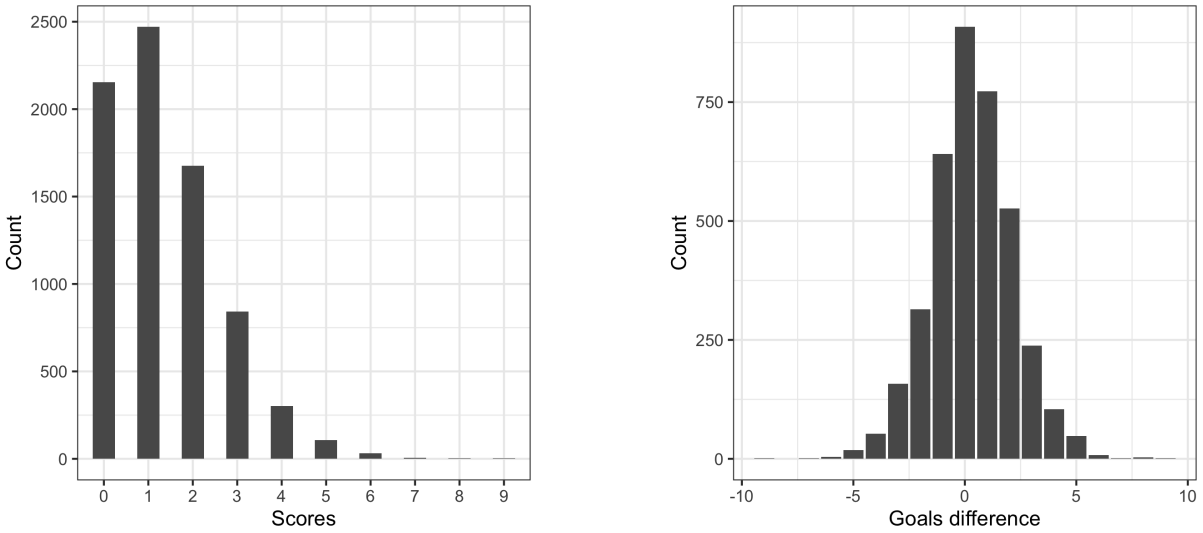


Figure 6: Left: the distribution of the number of goals scored; Right: goals difference in the EPL data (2011-2021).

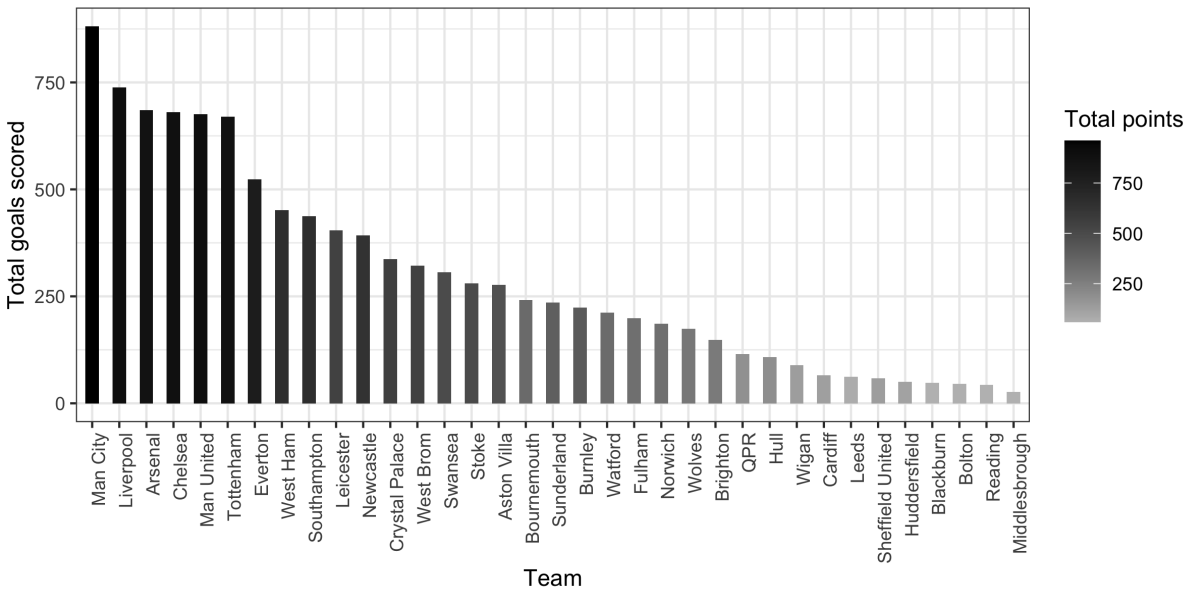


Figure 7: The number of goals scored by different teams in the EPL data (2011-2021). Teams with the highest points also scored the highest number of goals.

4 Soccer scores prediction

4.1 Models

We predict the number of goals by modelling the attack and defense strength of each team. The data consists of T teams, denoted by $t_1, t_2, \cdots, t_T$. For team t_i, the number of games played is n_T. The number of goals is denoted by $\boldsymbol{y} = (y_1, \cdots, y_{2G})$, where G is the number of games in the data. We consider goal scoring as a counting process and assume a Poisson likelihood:

$$y_i|\phi_i \sim Poisson(\phi_i) \tag{4.1}$$

where $\eta_i = \log(\phi_i)$. The ϕ_i represents the scoring intensity which is affected by both attack strength and the defense strength. The scoring intensity could be linked to a linear predictor η_i as introduced in Sec. 2.2.2. The specification of the linear predictor are introduced as follows.

4.1.1 Simple Poisson Regression: baseline

By assuming the attack/defense strength for each team is a constant, we have the simple Poisson regression specification as follows:

$$\eta_i = 1 + \sum_{g\in\mathcal{G}} \beta_{a_g,\alpha}\mathbb{I}_{a_i} + \sum_{g\in\mathcal{G}} \beta_{d_g,\xi}\mathbb{I}_{d_i} \tag{4.2}$$

where $g \in \mathcal{G}$ traverses all rows in the data, a_g and d_g represent the attack or defense team in row g, and the regression coefficients $\beta_{tm,\alpha}$ and $\beta_{tm,\xi}$ represent the team-specific attack or defense strength. In `R-INLA`, the corresponding formula is `y ~ 1 + attack +defense`.

4.1.2 Team strength as a time-invariant random effect: iid model

In the specification, we model the team strength using independent and identically distributed (iid) random effects. In fact, the scores of football are influenced by numerous factors. However, we model the scoring intensity only with very limited features. A model like the iid model accounts for disorganized variability in the data will be helpful to improve the performance. We use two vectors of random variables $\boldsymbol{u}_\alpha = (\alpha_{t_1}, \alpha_{t_2}, \cdots, \alpha_{t_T})$ and $\boldsymbol{u}_\xi = (\xi_{t_1}, \xi_{t_2}, \cdots, \xi_{t_T})$ to represent the attack and defense strength of each team respectively. In the model, we have:

$$\eta_i = 1 + \sum_{g\in\mathcal{G}} \alpha_{a_g}\mathbb{I}_{a_i} + \sum_{g\in\mathcal{G}} \xi_{d_g}\mathbb{I}_{d_i} \tag{4.3}$$

where $\alpha_t \sim N(0, 1/\tau_\alpha)$ and $\xi_t \sim N(0, 1/\tau_\xi)$ for any arbitrary team t. Note that the attack (or defense) strength for different teams are independent and identically distributed. In `R-INLA`, the corresponding formula is `y ~ 1 + f(attack, model='iid') + f(defense, model='iid')`.

4.1.3 Team strength as a time-variant random effect: blocked random walk model

Assuming the strength of each team is a random variable changing over time, we take $\boldsymbol{u}_\alpha$ (or $\boldsymbol{u}_\xi$) as a vector of random variables to represent the attack strength (or defense strength) of each team. To be precise, $\boldsymbol{u}_\alpha = (\alpha_{t_1,1}, \cdots, \alpha_{t_1,n_1}, \alpha_{t_2,1}, \cdots, \alpha_{t_2,n_2}, \cdots, \alpha_{t_T,1}, \cdots, \alpha_{t_T,n_T})$, where $\alpha_{t,n}$ represents the attack strength of the $n-$th game of team t ($\boldsymbol{u}_\xi$ is defined in the same way).

the process could be regarded as a random walk:

$$\eta_i = 1 + \boldsymbol{u}_\alpha[j_\alpha] + \boldsymbol{u}_\xi[j_\xi] \tag{4.4}$$

where j_α (or j_ξ) is the index of game i in $\boldsymbol{u}_\alpha$(or $\boldsymbol{u}_\xi$), $\Delta u_{\alpha j} = \boldsymbol{u}_\alpha[j] - \boldsymbol{u}_\alpha[j-1] \sim N(0, 1/\tau_\alpha)$, $\Delta u_{\xi i} = \boldsymbol{u}_\xi[j] - \boldsymbol{u}_\xi[j-1] \sim N(0, 1/\tau_\alpha)$ and the increments are independent from each other. Take the density for $\boldsymbol{u_{\alpha t}} = (\alpha_{t,1}, \cdots, \alpha_{t,n})$ as an example:

$$\begin{aligned} p(\boldsymbol{u}_{\alpha t}|\tau_\alpha) &\propto \tau_\alpha^{(n-1)/2} \exp\left(-\tau_\alpha/2 \sum_{i=1}^{n-1} (\Delta u_{\alpha i})^2\right) \\ &\propto \tau_\alpha^{(n-1)/2} \exp(-\tau_\alpha/2\, \boldsymbol{u}_{\alpha t}^T \boldsymbol{Q} \boldsymbol{u}_{\alpha t}) \end{aligned} \tag{4.5}$$

where $\boldsymbol{Q}$ is the structure matrix of random walk:

$$\boldsymbol{Q} = \begin{pmatrix} 1 & -1 & & & & & \\ -1 & 2 & -1 & & & & \\ & -1 & 2 & -1 & & & \\ & & \ddots & \ddots & \ddots & & \\ & & & -1 & 2 & -1 & \\ & & & & -1 & 2 & -1 \\ & & & & & -1 & 1 \end{pmatrix}_{n \times n} \tag{4.6}$$

The structure matrix $\boldsymbol{Q}$ is constructed by adding up n dimensional matrices $\boldsymbol{U}^{(i)}$ that is zero everywhere except for $U_{i,i} = U_{i+1,i+1} = -U_{i,i+1} = -U_{i+1,i} = 1$. $\boldsymbol{U}^{(i)}$ has the following form:

$$\boldsymbol{U}^{(i)} = \begin{pmatrix} \ddots & & & \\ & 1 & -1 & \\ & -1 & 1 & \\ & & & \ddots \end{pmatrix}_{n \times n} \tag{4.7}$$

We have $\boldsymbol{Q} = \sum_{i=1}^{n-1} \boldsymbol{U}^{(i)}$.

The default setting of 'generic0' in `R-INLA` could be used to implement this specification. Random effects in `R-INLA` are defined with multivariate Gaussian distribution with mean zero and precision matrix $\tau\boldsymbol{\Sigma}$, i.e. $N(0, (\tau\boldsymbol{\Sigma})^{-1})$. The 'generic0' assumes $\boldsymbol{\Sigma}$ is a constant matrix. Further, random walk could be implemented using 'generic0' by setting $\boldsymbol{\Sigma}$ to be the structure matrix of random walk $\boldsymbol{Q}$.

Besides, second order random walk could also be implemented by setting $\boldsymbol{\Sigma}$ to be the struc-

ture matrix of second order random walk:

$$\begin{pmatrix} 1 & -2 & 1 & & & & & & \\ -2 & 5 & -4 & 1 & & & & & \\ 1 & -4 & 6 & -4 & 1 & & & & \\ & 1 & -4 & 6 & -4 & 1 & & & \\ & & \ddots & \ddots & \ddots & \ddots & \ddots & & \\ & & & 1 & -4 & 6 & -4 & 1 & \\ & & & & 1 & -4 & 6 & -4 & 1 \\ & & & & & 1 & -4 & 5 & -2 \\ & & & & & & 1 & -2 & 1 \end{pmatrix} \tag{4.8}$$

We call this specification, blocked random walk (BRW). Note that we assume all teams' time-varying strength share the same precision τ. In `R-INLA`, the corresponding formula is `y` $\sim$ `1 + f(ID.a, model='generic0', Cmatrix =` Σ`) + f(ID.d, model='generic0', Cmatrix =` Σ`)`, where `ID.a` and `ID.d` are corresponding indices in $\boldsymbol{u}_\alpha$ and $\boldsymbol{u}_\xi$ of each game. Further, we could generalize BRW by considering:

1. the time-invariant team strength (fixed effects or iid effects);
2. the effect from the time difference between two games next to each other;
3. the effect from the the honeymoon period (30 days) for new managers.

By saying considering the last two effects, we mean decreasing the precision (increasing the variance) by dividing one-step blocks Eq. by a constant related to those effects. New one-step blocks are:

$$\boldsymbol{U}^{(i)} = \frac{1}{\Delta d_i + \eta\, \mathbb{I}_h} \begin{pmatrix} \ddots & & & \\ & 1 & -1 & \\ & -1 & 1 & \\ & & & \ddots \end{pmatrix} \tag{4.9}$$

where Δd_i is the number of days elapsed before the next game of the game in row i; $\mathbb{I}_h$ indicates whether the game is in the honeymoon period for new managers and η is the scaling constant (the default value is 10).

We call the generalized BRW model as gBRW model. Its formula in `R-INLA` is:

1. gBRW1: `y` $\sim$ `1 + attack + defense + f(ID.a, model='generic0', Cmatrix =` Σ`) + f(ID.d, model='generic0', Cmatrix =` Σ`)`;
2. gBRW2: `y` $\sim$ `1 + f(attack, model='iid') + f(defense, model='iid') + f(ID.a, model='generic0', Cmatrix =` Σ`) + f(ID.d, model='generic0', Cmatrix =` Σ`)`.

The models above are characterized by both team-specific and time-specific covariates. It could be written in terms of a hierarchical structure. The first level is identified by the observations, the number of scores, $\boldsymbol{y} = (y_1, \cdots, y_{2G})$, which are assumed to be independent and identically distributed Poisson variables given the latent field $\boldsymbol{\theta}$ (including regression parameters, random effects of covariates, and etc.) and some hyperparameters $\boldsymbol{\psi}_1$. Each observation in the first level belongs to one of the groups (divided by team and date). The groups here

are also called as second level units. At the second level, the latent field $\boldsymbol{\theta}$ given some other hyperparameters $\boldsymbol{\psi}_2$ (namely, the precision τ) is a multivariate Normal distribution. Finally, the hyperparameters $\boldsymbol{\psi} = \{\boldsymbol{\psi}_1, \boldsymbol{\psi}_2\}$ have the prior distribution (the default setting of `R-INLA` is log-Gamma(1,5e-5)). Such a hierarchical structure ensures observations that belong to the same second level unit are more similar to each other.

4.2 Estimation and evaluation

To predict the scores of matches, we create a new dataframe where the matches of interest have NA in their response variables $\boldsymbol{y}$. We fit the Poisson regression model in INLA for this new data set, including the options needed for sampling and then simulate 1000 samples from the joint approximated predictive distribution of the number of goals by home and away teams respectively, using the `inla.posterior.sample` method of `R-INLA`. Further, we can estimate the probabilities of home win, draw and lose to compute the predictive criteria we need.

Some posterior predictive model checks are done to assess model fit. For instance, in Fig. 8, we plot the attack strength and defense strength of different teams estimated by the marginal posterior mean of the fixed effects in the gBRW1 model. High attack strength and low defense strength increases the scoring intensity of a team. We can find those massive clubs such as Man City, Liverpool, Chelsea, Man United and Tottenham located at the lower bottom. Thus, this provides evidence that the model is able to make sensible predictions.

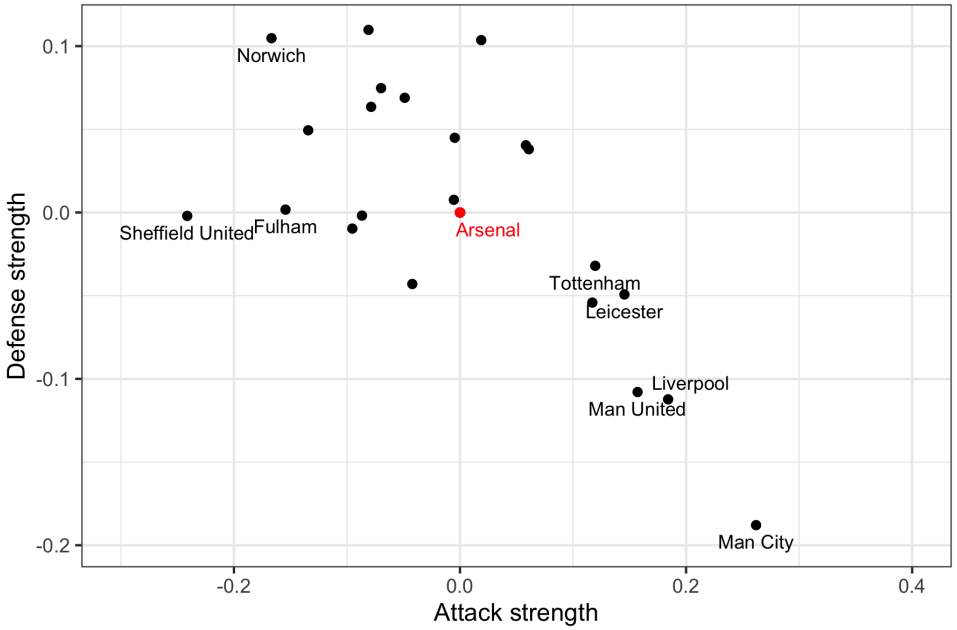
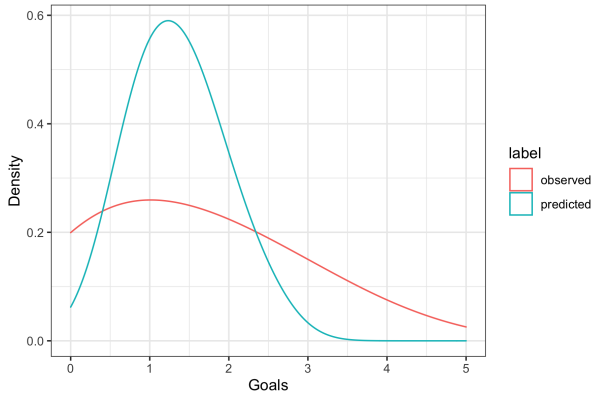
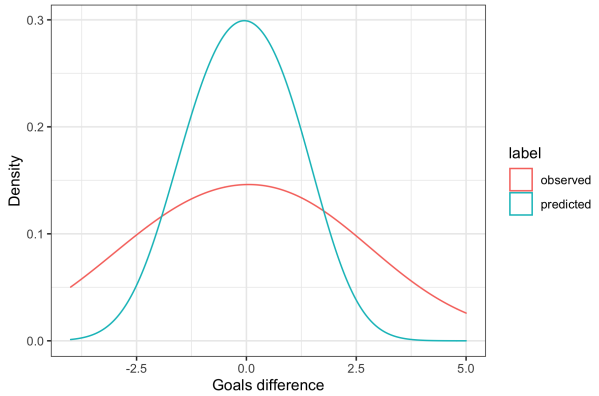


Figure 8: The attack strength and defense strength of different teams estimated by the marginal posterior mean of the fixed effects in the gBRW1 model. In this model, `R-INLA` sets the strength of Arsenal to zero automatically to ensure the identifiability of the team strengths.

We compare the distribution of observed scores and goals difference with the ones predicted by the gBRW1 model in Fig. 9. It shows that the two compared distribution share a similar mode. The majority of predicted scores are between 0 and 2 and the predicted goals difference are symmetrically distributed around 0 which is consistent with our analysis in Sec. 3.3 We find the observed distributions have a higher variance than the predicted ones, which is due to the small size of test data (games of last three rounds in season 2020/21). To maintain the proportion of training data to test data, we should increase the size of both. Alternatively, a more complex way to increase the size of test data without increasing the size of training data correspondingly is to dynamically predict new instances of games. Note that the distribution of both observed and predicted goals difference shows a low level of left skewness rather than right skewness. This might be due to the absence of home field advantage during the pandemic period.



(a)



(b)

Figure 9: Top: compare the observed scores distribution with the one predicted by the gBRW1 model estimated with the test data (EPL, 2019 - 2021); Bottom: compare the observed goals difference (home team score - away team score) distribution with the one predicted by the gBRW1 model.

Other than marginal log-likelihood and classification accuracy, the main evaluation criterion we used is the ranked probability score (RPS). The ranked probability score was introduced by Epstein in 1969 (Epstein, 1969). Let r be the number of possible outcomes, $\boldsymbol{p} = (p_1, \cdots, p_r)$ be the vector of predicted probabilities and $\boldsymbol{a} = (a_1, \cdots, a_r)$ be the observed outcomes. In the scenario of football, r=3. The RPS is defined as:

$$RPS = \frac{1}{r-1} \sum_{i=1}^{r-1} \{ \sum_{j=1}^{i} (p_j - a_j) \}^2 \tag{4.10}$$

Compared with the classification accuracy, the RPS gives a more subtle measure of how good the predicted probability distributions are in terms of matching observed outcomes. The score lies between 0 and 1, with lower scores being better.

In order to test the generalization ability of the trained model, we design a temporal evaluation framework that consists of 10 experiments. We predict the scores of matches in last three rounds of each season with all previous data in each of the 10 experiments. Notably, the K-fold cross-validation framework is not suitable in the situation of football scores predicting as the order of temporal data must be preserved.

4.3 Results

First, we present the result of the temporal evaluation proposed in Sec. 4.2 to evaluate the generalization ability of different Poisson likelihood models. We take the mean RPS and the mean accuracy of the 10 experiments as the evaluation score. As one can see from Tab. 1 and Tab. 2, the scores of the 5 models are very close to each other. gBRW1 has the lowest mean RPS and BRW has the highest mean accuracy. The scores of the BRW model and two generalized BRW model have higher standard deviation. Besides, we find all the BRW models have the lowest RPS in experiment 9 , where we use the data of 2011-2020 to predict the scores of matches in the last three rounds of season 2019/20. In this season, COVID-19 resulted in a three-month lay-off. This might means mixed-effect models have better performance when dealing with disorganized variability in the data.

ID	BL	IID	BRW	gBRW1	gBRW2
1	0.2155570	0.2178141	0.2166568	0.2169360	0.2176411
2	0.2528338	0.2539935	0.2427084	0.2428017	0.2399410
3	0.2108165	0.2083761	0.2192847	0.2156414	0.2222196
4	0.2732045	0.2716485	0.2806356	0.2741033	0.2784465
5	0.2413852	0.2418975	0.2346799	0.2318446	0.2327678
6	0.1960751	0.1941502	0.2068492	0.2060962	0.2042430
7	0.2267888	0.2273949	0.2338893	0.2360776	0.2373802
8	0.2435840	0.2441601	0.2767293	0.2708122	0.2716151
9	0.2224038	0.2223710	0.1979354	0.1963483	0.1944676
10	0.2190248	0.2174663	0.2119270	0.2077405	0.2072559
Mean	0.2301674	0.2299272	0.2321296	<u>0.2298402</u>	0.2305978
Std	0.0226516	0.0230354	0.0280562	0.0266476	0.0276743

Table 1: Comparison between RPS of different models computed in the temporal evaluation framework. We can't observe obvious correlation between the size of training data and the value of RPS.

ID	BL	IID	BRW	gBRW1	gBRW2
1	0.4209667	0.4141000	0.4192000	0.4159000	0.4140000
2	0.4282000	0.4195667	0.4270333	0.4288000	0.4289667
3	0.4569333	0.4540667	0.4777667	0.4789667	0.4725667
4	0.3679667	0.3668000	0.3617000	0.3648333	0.3604667
5	0.3359000	0.3335333	0.3405000	0.3397667	0.3391000
6	0.4870000	0.4835667	0.5253667	0.5235000	0.5260000
7	0.4203333	0.4159000	0.4454333	0.4435333	0.4365667
8	0.4042333	0.4062000	0.4303667	0.4248333	0.4324667
9	0.4337333	0.4310000	0.4394667	0.4350667	0.4359333
10	0.4372667	0.4348333	0.4476333	0.4524667	0.4520667
Mean	0.4192533	0.4159567	<u>0.4314467</u>	0.4307667	0.4298133
Std	0.0427283	0.0420684	0.0524881	0.0521359	0.0526558

Table 2: Comparison between classification accuracy of different models computed in the temporal evaluation framework. A minus upward trend as the training data size increases could be found in the first six experiments.

Next, we use the data from 2019/20 and 2020/21, 760 games in total, to test these models.

Still, Tab. 3 shows that the results are not ideal in terms of classification accuracy (slightly better than the accuracy of random guess, 1/3). Nevertheless, including random effects terms has proven helpful to improve the predictive performance of baseline model that only accounts for fixed effects.

	BL	**IID**	**BRW**	**gBRW1**	**gBRW2**
RPS	0.3028665	0.3040542	0.2868499	<u>0.2836997</u>	0.2852408
Accuracy	0.4407667	0.4356333	0.4453667	<u>0.4503000</u>	0.4458333

Table 3: The RPS and classification accuracy of applying different models to the test data (EPL, 2019-2021).

5 Discussion

The goal of this project is to predict the number of goals scored of EPL using improved Poisson likelihood models. Models presented in this paper is a simple application of Bayesian hierarchical modelling. Including both team-specific and time-specific random effect terms allows us to account for disorganized variability in the data. Such models naturally accommodates the INLA framework, and hence can be easily implemented and estimated using `R-INLA`.

An analysis of model fit showed that the model was adequately fitting the real data. With the well-designed temporal evaluation system, it can be concluded that the model including complicated random effects improves the capability to explain the variability in the data of the baseline model that only takes account of fixed team strength. Care should be taken when increasing variability of the model, as the variability in predictive performance might increase correspondingly.

One of the limitations of our models is that, we only make use of basic information (date and clubs) of history data for simplicity. Adding additional features, such as team-specific home field advantage, the distance traveled, injuries, suspensions, etc., may improve the predictive ability. In addition, predictions are obtained in one batch. A more more recommended way is to predict each game using the model estimated with newly updated data despite more computation required. Also, when constructing precision matrix for BRW models, we set some fixed time-specific hyperparameters that account for the arrival of new managers and the time difference between two games next to each other. It might could be improved if there is a way to include prior information for these hyperparameters in `R-INLA`.

References

[1] G. Baio and M. Blangiardo. Bayesian hierarchical model for the prediction of football results. *Journal of Applied Statistics*, 37(2):253–264, 2010.

[2] M. Blangiardo and M. Cameletti. *Spatial and spatio-temporal Bayesian models with R-INLA*. John Wiley & Sons, 2015.

[3] S. Chib and R. Winkelmann. Markov chain monte carlo analysis of correlated count data. *Journal of Business & Economic Statistics*, 19(4):428–435, 2001.

[4] M. J. Dixon and S. G. Coles. Modelling association football scores and inefficiencies in the football betting market. *Journal of the Royal Statistical Society: Series C (Applied Statistics)*, 46(2):265–280, 1997.

[5] E. S. Epstein. A scoring system for probability forecasts of ranked categories. *Journal of Applied Meteorology (1962-1982)*, 8(6):985–987, 1969.

[6] D. Karlis and I. Ntzoufras. On modelling soccer data. *Student*, 3(4):229–244, 2000.

[7] D. Karlis and I. Ntzoufras. Bayesian modelling of football outcomes: using the skellam's distribution for the goal difference. *IMA Journal of Management Mathematics*, 20(2):133–145, 2009.

[8] S. Martino and H. Rue. Case studies in bayesian computation using inla. In *Complex data modeling and computationally intensive statistical methods*, pages 99–114. Springer, 2010.

[9] T. G. Martins, D. Simpson, F. Lindgren, and H. Rue. Bayesian computing with inla: new features. *Computational Statistics & Data Analysis*, 67:68–83, 2013.

[10] T. Opitz. Latent gaussian modeling and inla: A review with focus on space-time applications. *Journal de la société française de statistique*, 158(3):62–85, 2017.

[11] H. Rue and L. Held. *Gaussian Markov random fields: theory and applications*. CRC press, 2005.

[12] H. Rue, S. Martino, and N. Chopin. Approximate bayesian inference for latent gaussian models by using integrated nested laplace approximations. *Journal of the royal statistical society: Series b (statistical methodology)*, 71(2):319–392, 2009.

[13] H. Rue, A. Riebler, S. H. Sørbye, J. B. Illian, D. P. Simpson, and F. K. Lindgren. Bayesian computing with inla: a review. *Annual Review of Statistics and Its Application*, 4:395–421, 2017.

Appendices

A The INLA implementation of Poisson likelihood models

```
#' Model attack and defense strength as fixed effects
#' Simple Poisson regression

m.pois = function(df) {
  formula <- y ~ 1 + attack + defense

  m.bl <- inla(formula, family = "poisson", data=df,
               control.predictor = list(compute = TRUE),
               control.compute = list(config = TRUE))

  return(m.bl)
}


#' Model attack and defense strength as iid random effects
#' The default setting of 'iid' in R-INLA could be used to implement this
    specification

m.pois = function(df) {
  formula <- y ~ 1 +
    f(attack, model="iid", hyper=list(theta=list(initial=1, fixed=T))) +
    f(defense, model="iid", hyper=list(theta=list(initial=1, fixed=T)))
    # To save time, we fixed the hyperparameters

  m.iid <- inla(formula, family = "poisson", data=df,
               control.predictor = list(compute = TRUE),
               control.compute = list(config = TRUE))

  return(m.iid)
}

#' Model the attack and defense strength as random effects
#' Assume the strength of each team is a random variable changing over time
#' Regard the process as a random walk
#' The default setting of 'generic0' in R-INLA could be used to implement this
    specification

m.pois = function(df) {
  ls = get_Sig(data, cd=0, nmg=0)
  # The attack/defense strength share the same precision matrix
  Sig = ls$Sig
  id.levels = ls$id

  df.brw = df %>% mutate(
    ID.a = factor(paste(date, attack), levels=id.levels),
    ID.d = factor(paste(date, defense), levels=id.levels)
  )


  formula <- y ~ 1 +
```

```
    f(ID.a, model='generic0', Cmatrix = Sig, values = id.levels,
    diagonal = 1e-5, rankdef = Ts,
      hyper = list(
        theta = list(
          prior = "loggamma",
          param = c(1,.00005),
          initial = log(100000),
          fixed = T)) # To save time, we fixed the hyperparameters
    ) +
    f(ID.d, model='generic0', Cmatrix = Sig, values = id.levels,
    diagonal = 1e-5, rankdef = Ts,
      hyper = list(
        theta = list(
          prior = "loggamma",
          param = c(1,.00005),
          initial = log(100000),
          fixed = T))
    )

  m.brw <- inla(formula, family='poisson', data = df.brw,
                control.predictor = list(compute = TRUE),
                control.compute = list(config = TRUE))

  return(m.brw)
}


#' Combining the fixed strength with the time-varying strength

m.pois =  function(df)  {
  ls = get_Sig(df, cd=1, nmg=1)  # The attack/defense strength share the same
    precision matrix
  Sig = ls$Sig
  id.levels = ls$id

  df.brw = df %>% mutate(
    ID.a = factor(paste(date, attack), levels=id.levels),
    ID.d = factor(paste(date, defense), levels=id.levels)
  )


  formula <- y ~ 1 +
    attack + defense +
    f(ID.a, model='generic0', Cmatrix = Sig, values = id.levels,
    diagonal = 1e-5, rankdef = Ts,
      hyper = list(
        theta = list(
          prior = "loggamma",
          param = c(1,.00005),
          initial = log(100000),
          fixed = T)) # To save time, we fixed the hyperparameters
    ) +
    f(ID.d, model='generic0', Cmatrix = Sig, values = id.levels,
    diagonal = 1e-5, rankdef = Ts,
```

```
    hyper = list(
      theta = list(
        prior = "loggamma",
        param = c(1,.00005),
        initial = log(100000),
        fixed = T))
  )

 m.gbrw1 <- inla(formula, family='poisson', data = df.brw,
                 control.predictor = list(compute = TRUE),
                 control.compute = list(config = TRUE))


 return(m.gbrw1)
}


#' Combining the iid strength with the time-varying strength

m.pois =  function(df)  {
 ls = get_Sig(df, cd=1, nmg=1)
 # The attack/defense strength share the same precision matrix
 Sig = ls$Sig
 id.levels = ls$id

 df.brw = df %>% mutate(
   ID.a = factor(paste(date, attack), levels=id.levels),
   ID.d = factor(paste(date, defense), levels=id.levels)
 )


 formula <- y ~ 1 +
   f(attack, model="iid") + f(defense, model="iid") +
   f(ID.a, model='generic0', Cmatrix = Sig, values = id.levels,
   diagonal = 1e-5, rankdef = Ts,
     hyper = list(
       theta = list(
         prior = "loggamma",
         param = c(1,.00005),
         initial = log(100000),
         fixed = T)) # To save time, we fixed the hyperparameters
   ) +
   f(ID.d, model='generic0', Cmatrix = Sig, values = id.levels,
   diagonal = 1e-5, rankdef = Ts,
     hyper = list(
       theta = list(
         prior = "loggamma",
         param = c(1,.00005),
         initial = log(100000),
         fixed = T))
   )

 m.gbrw2 <- inla(formula, family='poisson', data = df.brw,
                 control.predictor = list(compute = TRUE),
```

```
                    control.compute = list(config = TRUE))


  return(m.gbrw2)
}
```

B Construct the blocked random walk precision matrix

```
#' Model the attack and defense strength as random effects
#' Assume the strength of each team is a random variable changing over time
#' Regard the process as a random walk
#' The default setting of 'generic0' in R-INLA could be used to implement this specification
#'
#' Random effects in R-INLA are defined with multivariate Gaussian distribution with mean zero and precision matrix tau*Sigma
#' The 'generic0' assumes Sigma is a constant matrix
#'
#' Random walk could be implemented using 'generic0' by setting Sigma to be the structure matrix of random walk:
#'  1  -1
#' -1   2  -1
#'     ...
#'     -1   2  -1
#'          -1  1
#' the structure matrix is constructed by adding the matrix (represents one step in the random walk) with following form:
#'      i   j
#'  i   1  -1
#'  j  -1   1
#'  (unspecified entries are zero)
#'
#'  Back to our specification, we take u as a vector of random variables to represent each teams attack/defense
#'  ability at each games, to be precise
#'  u = (xi_{tm1,day1}, ..., xi_{tm1,dayn1},...,xi{tmTs,day1},...,xi_{tmTs,daynTs}) (arranged by team first and then the dates)
#'  Since each team's strength changes over time as a random walk
#'  the corresponding precision matrix will be a block matrix whose block is the structure matrix of random walk
#'  We call this specification, blocked random walk (BRW)
#'
#'  Further, we could generalize BRW by:
#'  1. Consider the effect from the time difference between two games next to each other
#'  2. Consider the effect from the arrival of new managers (honeymoon period)
#'  Saying considering those effects, we mean decreasing the precision (increase the variance) by dividing
#'  one-step blocks by a constant related to those effects

# Construct the precision matrix

get_Sig <- function(df, cd = 0, nmg = 0){
  # Sort df by team and date; save the new order by id
  id.levels = within(df %>% arrange(attack, date),
    id <- paste(date, attack))$id  # The order of block matrix
  id = (df %>% mutate(id = as.numeric(factor(paste(date, attack), levels=id.levels))))$id
  idx = 1:nrow(df) # get the index of df
  df$idx = idx  # add a index column
```

```
43
44  # Construct the precision matrix by adding up one-step precision matrix
45  # is.last.game indicates whether the game is the last game in the dataset for
      each team
46  # if so stop adding up
47
48  Sig = inla.as.sparse(matrix(rep(0, 4*G*G), 2*G, 2*G))  # initialize the output
49  for (i in idx) {
50    if ( df$is.last.game[i] == FALSE ) {  # if the current game is the last game
       of the team, skip the update step
51      j = idx[id == (id[i] + 1)]
52
53      eff.cd = (as.numeric(df$date[j] - df$date[i]))*cd  # time difference
      between two games next to each other
54      eff.nmg = (df$is.hm.a[i])*nmg  # if the game effected by the arrival of
      new managers
55
56      eff = 1 + eff.cd + eff.nmg
57
58      Sig[i,i] = Sig[i,i] + eff
59      Sig[j,j] = Sig[j,j] + eff
60      Sig[i,j] = Sig[i,j] - eff
61      Sig[j,i] = Sig[j,i] - eff
62    }
63  }
64
65  # So far, the precision matrix Sigma follows the order of idx
66  # Arrange the matrix by id
67  idx.sorted = (data.frame(idx, id) %>% arrange(id))$idx
68  Sig.sorted = Sig[idx.sorted, idx.sorted]
69
70  return(list(Sig=Sig.sorted, id=id.levels))
71 }
```

C The temporal evaluation framework

```
# Import the data set
source("code/0_prep_data.r")

# Load the model list
model.ls = list()
i = 1
for (filename in list.files("code/models", full.names = T)){
  model.ls[[i]] = filename
  i = i+1
}

# Load the function that calculates the precision matrix for 'generic0' model
source(model.ls[[1]])

# Define the function to calculate rps
RPS_fun<-function(p0,p1,a0,a1){
  s1 = (p0-a0)
  s2 = (p0-a0+p1-a1)
  r = (s1*s1 + s2*s2)/2
  return(r)
}

# Initialize the output
rps.ma = matrix(rep(0,6*10), nrow = 10, ncol = 6)
acc.ma = matrix(rep(0,6*10), nrow = 10, ncol = 6)

for (m in c(1:5)){  # load the model m
  source(model.ls[[m+1]])

  for (n in 1:10) {   # conduct the experiment n
    #We create a new dataframe where the matches of the last three rounds (30
    games) in each season
    # have NA in their response variables y
    r = 30
    data=df[c(1:(380*n), (G+1):(G+380*n)),]
    data$goals = data$y
    G.tr = nrow(data)/2L
    data[(G.tr-r+1):G.tr,"y"]<-NA
    data[(2*G.tr-r+1):(2*G.tr),"y"]<-NA

    # Train the model with new data
    mdl = m.pois(data)
    print("CPU used:")
    print(mdl$cpu.used)
    print(paste("Model",m,"Trained with",G.tr,"games,",n,"seasons."))

    #We obtain the samples from the linear predictors
    nbsamp = 1000
    samp = inla.posterior.sample(nbsamp, mdl)
    predictors = inla.posterior.sample.eval(function(...) {Predictor}, samp)

    #We will store the scores.samples from the posterior predictive in a matrix
```

```
    scores.samples

    scores.samples=matrix(0,nrow=2*r,ncol=nbsamp)

    rates.H=exp(predictors[(G.tr-r+1):G.tr,])
    vector.rates.H=as.vector(rates.H)
    scores.samples[1:r,]=matrix(rpois(length(vector.rates.H),vector.rates.H),
    nrow=r,ncol=nbsamp)

    rates.A=exp(predictors[(2*G.tr-r+1):(2*G.tr),])
    vector.rates.A=as.vector(rates.A)
    scores.samples[(r+1):(2*r),]=matrix(rpois(length(vector.rates.A),vector.
    rates.A),nrow=r,ncol=nbsamp)

    # Get the probability of draw and lose (for fact and sample)
    a2 = as.numeric(df[(G.tr-r+1):G.tr, "y"] >  df[(2*G.tr-r+1):(2*G.tr),"y"])
    a1 = as.numeric(df[(G.tr-r+1):G.tr, "y"] == df[(2*G.tr-r+1):(2*G.tr),"y"])
    a0 = as.numeric(df[(G.tr-r+1):G.tr, "y"] <  df[(2*G.tr-r+1):(2*G.tr),"y"])

    p2 = rowMeans(scores.samples[1:r,] >  scores.samples[(r+1):(2*r),])
    p1 = rowMeans(scores.samples[1:r,] == scores.samples[(r+1):(2*r),])
    p0 = rowMeans(scores.samples[1:r,] <  scores.samples[(r+1):(2*r),])

    # Pass those probabilities to RPS function to calculate RPS
    rps.ma[n,m] = mean(RPS_fun(p0, p1, a0, a1))
    print(paste("RPS is",rps.ma[n,m],"."))

    # Compute the accuracy
    scores.obs = data$goals[is.na(data$y)]
    df.pred = (scores.samples[1:r,] >  scores.samples[(r+1):(2*r),])*2 + (scores
    .samples[1:r,] == scores.samples[(r+1):(2*r),])
    df.true = matrix(rep((scores.obs[1:r] >  scores.obs[(r+1):(2*r)])*2 + (
    scores.obs[1:r] == scores.obs[(r+1):(2*r)]),1000), nrow=30, ncol=1000)
    acc.ma[n,m] = mean(df.pred == df.true)
    print(paste("Accuracy is",acc.ma[n,m],"."))

    print("
    ================================================================================")
  }

  # Check the statics of the validation rps
  rps.stats = list(mean=mean(rps.ma[,m]),sd=sd(rps.ma[,m]),quantile=quantile(rps
    .ma[,m]))
  print("RPS statics:")
  print(rps.stats)
  print("
    ================================================================================")
  print("
    ================================================================================")
}

# Display the rps.ma and acc.ma
rps.ma
apply(rps.ma, 2, mean)
```

```
apply(rps.ma, 2, sd)

acc.ma
apply(acc.ma, 2, mean)
apply(acc.ma, 2, sd)
```