Chapter 8
Modeling and Prediction for Processes on Network Graphs

8.1 Introduction

Throughout this book so far, we have seen numerous examples of network graphs that provide representations—useful for various purposes—of the interaction among elements in a system under study. Often, however, it is some quantity (or attribute) associated with each of the elements that ultimately is of most interest. In such settings it frequently is not unreasonable to expect that this quantity be influenced in an important manner by the interactions among the elements. For example, the behaviors and beliefs of people can be strongly influenced by their social interactions; proteins that are more similar to each other, with respect to their DNA sequence information, often are responsible for the same or related functional roles in a cell; computers more easily accessible to a computer infected with a virus may in turn themselves become more quickly infected; and the relative concentration of species in an environment (e.g., animal species in a forest or chemical species in a vat) can vary over time as a result of the nature of the relationships among species.

Quantities associated with such phenomena can usefully be thought of as stochastic processes defined on network graphs. More formally, they can be represented in terms of collections of random variables, say $X$, indexed on a network graph $G = (V, E)$, either of the form $\{X_i\}$, for $i \in V$, or $\{X_i(t)\}$, with $t$ varying in a discrete or continuous manner over a range of times. For example, the functionality of proteins can be viewed as categorical variables associated with each $i \in V$. So too can various behaviors and beliefs of individuals in a social community be represented using such variables, possibly indexed in time. Similarly, the spread of a computer virus can be captured using a set of binary variables (i.e., ‘infected’ or ‘not infected’) that evolve over time.

We will refer to processes $\{X_i\}$ as static processes and $\{X_i(t)\}$ as dynamic processes. Given appropriate measurements, statistical tasks that arise in the study of such processes include their modeling and the inference of model parameters, and also, in particular, prediction. To date, most such work arguably has been done in the context of static processes, although this situation is changing.
Accordingly, our presentation in this chapter will concentrate largely on the case of static processes, with the development of methods and models in Sect. 8.2 through 8.4. In addition, we briefly discuss the modeling and prediction of dynamic processes in Sect. 8.5. Some further discussion related to this latter topic may be found in Sect. 11.5.

### 8.2 Nearest Neighbor Methods

We begin by focusing on the problem of predicting a static process on a graph and demonstrate the feasibility of doing so by examining one of the simplest of methods for this task—nearest-neighbor prediction.

Consider a collection of vertex attributes, which, as in previous chapters, we will express succinctly in vector form as $X = (X_i)$. Such attributes may be inherently independent of time, and hence form a truly static process, or perhaps more commonly, may constitute a ‘snapshot’ of a dynamic process in a given ‘slice’ of time. In Chaps. 6 and 7, such attributes were used in modeling and predicting the presence or absence of edges in a network graph $G$. That is, we modeled the behavior of the variables $Y = [Y_{ij}]$, conditional on $X$. Alternatively, however, in some contexts it may be the behavior of $X$, conditional on $Y$, that is of interest instead.

We illustrate through the problem of protein function prediction. Recall, from our discussion of validation of graph partitioning in Sect. 4.4.3 and of assortativity and mixing in Sect. 4.5, that the affinity of proteins to physically bind to each other is known to be directly related to their participation in common cellular functions. And, in fact, we found in our analysis of the data set yeast that the external assignment of proteins to functional classes correlated to a reasonable extent with their assignment to ‘communities’ by our graph partitioning algorithms. Furthermore, we saw examples of strong assortative mixing of protein function in the underlying network of protein–protein interactions. While the gold standard for establishing the functionality of proteins is through direct experimental validation (or ‘assays’), results like these have been taken to suggest that, given the vast number of proteins yet to be annotated, it is natural to approach this problem from the perspective of statistical prediction. Approaches to protein function prediction that incorporate network-based information are standard.

The data set ppiCC

```r
#8.1 1 > set.seed(42)
2 > library(sand)
3 > data(ppi.CC)
```

contains a network data object, called ppi.CC, that consists of a network of 241 interactions among 134 proteins, as well as various vertex attributes.
These data pertain to Baker’s yeast—the organism formally known as *Saccharomyces cerevisiae*. They were assembled by Jiang et al. [12], from various sources, and pertain to only those proteins annotated, as of January 2007, with the term cell communication in the gene ontology\(^1\) (GO) database—a standard database for terms describing protein function. The vertex attribute ICSC is a binary vector

\[ V(ppi.CC)[ICSC == 1]$color <- "yellow" \]
\[ V(ppi.CC)[ICSC == 0]$color <- "blue" \]
\[ plot(ppi.CC, vertex.size=5, vertex.label=NA) \]

We can see that there is a good deal of homogeneity to the vertex labels, with neighbors frequently sharing the same color. This observation suggests that local prediction of ICSC on this network graph should be feasible.

A simple, but often quite effective, method for producing local predictions is the nearest-neighbor method. See Hastie et al. [10, Sect. 2.3.2], for example, for general background on nearest-neighbor methods. For networks, the nearest-neighbor method centers on the calculation, for a given vertex \(i \in V\), of the nearest-neighbor average

\[ \frac{\sum_{j \in \mathcal{N}_i} x_j}{|\mathcal{N}_i|}, \]

i.e., the average of the values of the vertex attribute vector \(X\) in the neighborhood \(\mathcal{N}_i\) of \(i\). Here \(|\mathcal{N}_i|\) denotes the number of neighbors of \(i\) in \(G\). Calculation of these averages over all vertices \(i \in V\) corresponds to a nearest-neighbor smoothing of \(X\) across the network.

In the context of protein function prediction, \(X\) is a binary vector, with entries indicating whether or not each protein is or is not annotated with a function of interest (e.g., ICSC). In predicting binary vertex attributes, the nearest-neighbor averages (8.1) typically are compared to some threshold. For example, a threshold of 0.5

\(^{1}\)http://www.geneontology.org.
is commonly used, with a nearest-neighbor average greater than this value meaning that a majority of neighbors have the characteristic indicated by \( X = 1 \), resulting in a prediction for \( X_i \) of 1 as well. Such methods are also known as ‘guilt-by-association’ methods in some fields.

In order to obtain some sense as to how effective the nearest-neighbor method might be in predicting ICSC in our yeast data set, utilizing the information available through protein–protein interactions, we can calculate the nearest-neighbor average for each of the proteins in the giant connected component of our network.

```r
#8.5 1 > clu <- components(ppi.CC)
2 > ppi.CC.gc <- induced_subgraph(ppi.CC,
3 +   clu$membership==which.max(clu$csize))
4 > nn.ave <- sapply(V(ppi.CC.gc),
5 +   function(x) mean(V(ppi.CC.gc)[nei(x)]$ICSC))
```

We then plot histograms of the resulting values, separated according to the status of the vertex defining each neighborhood, i.e., according to the status of the ‘ego’ vertex, in the terminology of social networks.

```r
#8.6 1 > par(mfrow=c(2,1))
2 > hist(nn.ave[V(ppi.CC.gc)$ICSC == 1], col="yellow",
3 +   ylim=c(0, 30), xlab="Proportion Neighbors w/ ICSC",
4 +   main="Egos w/ ICSC")
5 > hist(nn.ave[V(ppi.CC.gc)$ICSC == 0], col="blue",
6 +   ylim=c(0, 30), xlab="Proportion Neighbors w/ ICSC",
7 +   main="Egos w/out ICSC")
```

The results, shown in Fig. 8.2, confirm that ICSC can be predicted with fairly good accuracy.\(^2\) In particular, using a threshold of 0.5 would yield an error rate of roughly 25%.

\(^2\)A more rigorous assessment of predictive performance would use some version of cross-validation, similar to what was used in Sect. 6.4. However, for the purposes of illustration, here and elsewhere
8.2 Nearest Neighbor Methods

![Histograms of nearest-neighbor averages for the network shown in Fig. 8.1, separated according to the status of the vertex defining each neighborhood (i.e., ‘ego’)](image)

Fig. 8.2 Histograms of nearest-neighbor averages for the network shown in Fig. 8.1, separated according to the status of the vertex defining each neighborhood (i.e., ‘ego’)

```r
#8.7 1 > nn.pred <- as.numeric(nn.ave > 0.5)
2 > mean(as.numeric(nn.pred != V(ppi.CC.gc)$ICSC))
3 [1] 0.2598425
```

Interestingly, we can push this illustration a bit further by taking advantage of the evolving nature of biological databases like GO. In particular, the proteins annotated in GO as not having a given biological function include both (1) those that indeed are known not to have that function, and (2) those whose status is simply unknown. As a result, by comparing against more recent versions of GO, it is sometimes possible to identify proteins whose status has changed for a given functional annotation, indicating that in the interim it has been discovered to in fact have that function.

The R package GOstats, a part of the Bioconductor package, may be used to manipulate and analyze Gene Ontology, as contained in the database GO.db.

```r
#8.8 1 > if (!requireNamespace("BiocManager", quietly = TRUE))
2 +   install.packages("BiocManager")
3 > BiocManager::install()
4 > BiocManager::install(c("GOstats","GO.db"))
5 > library(GOstats)
6 > library(GO.db)
```

In this chapter we instead content ourselves with examination of the fitted values produced by the various methods discussed, which, for sufficiently low to moderate numbers of predictions to be made, should be reflective of actual predictive performance.
And the annotations specific to the organism yeast can be obtained from the org.Sc.sgd.db database.

```r
#8.9 1 > BiocManager::install("org.Sc.sgd.db")
2 > library(org.Sc.sgd.db)
```

At the time of this writing, these annotations were last updated in January of 2017, roughly ten years after the data in ppi.CC were assembled.

We extract those proteins with the function ICSC—now subsumed\(^3\) under the term intercellular signaling transduction (ICST), or GO label 003556—and keep only those that have been identified from direct experimental assay, as indicated by the evidence code ‘IDA’.

```r
#8.10 1 > x <- as.list(org.Sc.sgdGO2ALLORFS)
2 > current.icst <- x[names(x) == "GO:0035556"]
3 > ev.code <- names(current.icst[[1]])
4 > icst.ida <- current.icst[[1]][ev.code == "IDA"]
```

We then separate out the names of those proteins that had ICSC in our original data

```r
#8.11 1 > orig.icsc <- V(ppi.CC.gc)[ICSC == 1]$name
```

and similarly extract the names of those proteins under the new annotations that were present in the giant connected component of our original network.

```r
#8.12 1 > candidates <- intersect(icst.ida, V(ppi.CC.gc)$name)
```

Among these candidates, there are seven that have been newly discovered to have ICSC, with the following names.

```r
#8.13 1 > new.icsc <- setdiff(candidates, orig.icsc)
2 > new.icsc
3 [1] "YDL159W" "YDL235C" "YHL007C" "YIL033C"
4 [5] "YIL147C" "YLR006C" "YLR362W"
```

And among these seven, we find that three of them would have been correctly predicted by comparing the value of their nearest-neighbor averages to a threshold of 0.5.

```r
#8.14 1 > nn.ave[V(ppi.CC.gc)$name %in% new.icsc]
2 YIL033C YLR362W YDL159W YLR006C YHL007C
3 0.7500000 0.4166667 0.3333333 0.6666667 0.8750000
4 YDL235C YIL147C
5 0.0000000 0.0000000
```

\(^3\)Not only do the protein annotations in GO evolve over time, as biological knowledge continues to be developed and refined, but so too (to a lesser extent) do the actual names of those annotations, as is the case here, with intercellular signaling cascade being replaced in the GO database by intercellular signaling transduction.
8.3 Markov Random Fields

The principles underlying the nearest-neighbor method can be formalized and extended through the construction of appropriate statistical models. Such modeling can allow, for example, for probabilistically rigorous predictive statements as well as estimation and testing of model parameters. In addition, a modeling perspective facilitates a systematic approach to the inclusion of both network (endogenous) and non-network (exogenous) effects. It can also facilitate the handling of missing data. Markov random fields (MRFs) represent one well-developed modeling paradigm that achieves all of these goals.

8.3.1 General Characterization

Let \( G = (V, E) \) be a graph and \( \mathbf{X} = (X_1, \ldots, X_{N_v})^T \) be a collection of discrete random variables defined on \( V \). We say that \( \mathbf{X} \) is a Markov random field (MRF) on \( G \) if

\[
P(\mathbf{X} = \mathbf{x}) > 0, \quad \text{for all possible outcomes } \mathbf{x}, \quad (8.2)
\]

and

\[
P\left( X_i = x_i \mid \mathbf{X}_{(-i)} = \mathbf{x}_{(-i)} \right) = P\left( X_i = x_i \mid \mathbf{X}_{\mathcal{N}_i} = \mathbf{x}_{\mathcal{N}_i} \right), \quad (8.3)
\]

where \( \mathbf{X}_{(-i)} \) is the vector \((X_1, \ldots, X_{i-1}, X_{i+1}, \ldots, X_{N_v})^T\) and \( \mathbf{X}_{\mathcal{N}_i} \) is the vector of all \( X_j \) for \( j \in \mathcal{N}_i \). The positivity assumed in (8.2) for the joint distribution of \( \mathbf{X} \) is simply a useful technical condition. The expression in (8.3) is the key Markov condition, asserting that \( X_i \) is conditionally independent of all other \( X_k \), given the values of its neighbors, where the neighborhood structure is determined by \( G \).

The concept of an MRF can be seen as a generalization of a Markov chain (common in the modeling of temporal data) and has its roots in statistical mechanics, going back to the work of Ising [11] on ferromagnetic fields. MRFs are used extensively in spatial statistics (e.g., see Cressie [6, Sect. 6.4]) and in image analysis (e.g., see Li [15]).

A key feature facilitating the practical usage of Markov random fields is their equivalence, under appropriate conditions, with Gibbs random fields i.e., random vectors \( \mathbf{X} \) with distributions of the form

\[
P(\mathbf{X} = \mathbf{x}) = \left( \frac{1}{\kappa} \right) \exp \{ U(\mathbf{x}) \}. \quad (8.4)
\]

Here \( U(\cdot) \) is called the energy function and

\[
\kappa = \sum_x \exp \{ U(\mathbf{x}) \}, \quad (8.5)
\]
the partition function. Importantly, the energy function can be decomposed as a sum over cliques in $G$, in the form

$$U(x) = \sum_{c \in \mathcal{C}} U_c(x),$$  \hspace{1cm} (8.6)

where $\mathcal{C}$ denotes the set of all cliques of all sizes in $G$, and a clique of size 1 consists of just a single vertex $v \in V$.

In this abstract form, MRF models can involve extremely complicated expressions, a fact which, while arguably an indication of their richness on the one hand, can adversely impact both interpretability and computations. In practice, these models often are simplified by assumptions of homogeneity, in the sense that the form of the clique potentials $U_c$ is assumed not to depend on the particular positions of the cliques $c \in \mathcal{C}$. Furthermore, usually cliques of only a limited size are defined to have non-zero partition functions $U_c$, which reduces the complexity of the decomposition in (8.6). This later step has direct implications on the nature of the assumed dependency in $X$.

Here we will focus on a class of MRFs commonly used in network analysis for modeling binary vertex attribute data, like the indicators of protein function (e.g., ICSC) discussed previously. These models are sometimes referred to as auto-logistic models.\footnote{For continuous-valued data, there is an analogous framework of so-called auto-Gaussian models, which bear close structural resemblance to the class of Gaussian graphical models we have seen in Sect. 7.3.3. We refer the reader to [13, Sect 8.3.1] for a brief discussion of these models, in the context of network analysis, and additional references.}

### 8.3.2 Auto-Logistic Models

The class of auto-logistic models goes back to Besag [3], who suggested introducing the additional conditions on MRFs that (i) only cliques $c \in \mathcal{C}$ of size one or two have non-zero potential functions $U_c$, and (ii) the conditional probabilities in (8.3) have an exponential family form (i.e., a form like that in (6.1)). The first condition is sometimes referred to as ‘pairwise-only dependence.’ Under these conditions, the energy function takes the form

$$U(x) = \sum_{i \in V} x_i H_i(x_i) + \sum_{\{i, j\} \in E} \beta_{ij} x_i x_j,$$  \hspace{1cm} (8.7)

for some functions $H_i(\cdot)$ and coefficients $\{\beta_{ij}\}$. Besag called the class of Markov random field models with energy functions like that in (8.7) auto-models.

Now suppose that the $X_i$ are binary random variables (i.e., taking on just the values zero and one). Under appropriate normalization conditions, the functions $H_i$
can be made to only contribute to the expansion of $U(x)$ in (8.6) in a non-trivial fashion when $x_i = 1$, in which case (8.7) can be shown to be equivalent in form to

$$U(x) = \sum_{i \in V} \alpha_i x_i + \sum_{(i,j) \in E} \beta_{ij} x_i x_j , \quad (8.8)$$

for certain parameters $\{\alpha_i\}$. The resulting MRF model is called an auto-logistic model, because the conditional probabilities in (8.3) have the form

$$P_{\alpha,\beta}(X_i = 1 | X_{N_i} = x_{N_i}) = \frac{\exp(\alpha_i + \sum_{j \in N_i} \beta_{ij} x_j)}{1 + \exp(\alpha_i + \sum_{j \in N_i} \beta_{ij} x_j)} , \quad (8.9)$$

indicating logistic regression of $x_i$ on its neighboring $x_j$’s.

Assumptions of homogeneity can further simplify this model. For example, specifying that $\alpha_i \equiv \alpha$ and $\beta_{ij} \equiv \beta$, the probability in (8.9) reduces to

$$P_{\alpha,\beta}(X_i = 1 | X_{N_i} = x_{N_i}) = \frac{\exp(\alpha + \beta \sum_{j \in N_i} x_j)}{1 + \exp(\alpha + \beta \sum_{j \in N_i} x_j)} . \quad (8.10)$$

This model can be read as dictating that the logarithm of the conditional odds that $X_i = 1$ scales linearly in the number of neighbors $j$ of $i$ with the value $X_j = 1$.

$$\log P_{\alpha,\beta}(X_i = 1 | X_{N_i} = x_{N_i}) = \log P_{\alpha,\beta}(X_i = 0 | X_{N_i} = x_{N_i}) = \alpha + \beta \sum_{j \in N_i} x_j . \quad (8.11)$$

Hence, we see that such (homogeneous) auto-logistic models effectively can be viewed as probabilistic extensions of nearest-neighbor methods. The R package \texttt{ngspatial} allows for the specification and fitting of such models.

\#8.15

More precisely, it allows for models with both endogenous and exogenous effects, in which the logarithm of the conditional odds in (8.11) takes the form

$$\log \frac{P_{\alpha,\beta}(X_i = 1 | X_{N_i} = x_{N_i}, Z_i = z_i)}{P_{\alpha,\beta}(X_i = 0 | X_{N_i} = x_{N_i}, Z_i = z_i)} = z_i^T \alpha + \beta \sum_{j \in N_i} (x_j - \mu_j) . \quad (8.12)$$

Here the $Z_i$ are vectors of exogenous variables, indexed by vertex $i$, and $\alpha$ is a corresponding vector of coefficients. The values $\mu_j = \{1 + \exp(-z_j^T \alpha)\}^{-1}$ are the expectations of the $X_j$ under independence, i.e., when $\beta = 0$. The presence of the $\mu_j$ amounts to a centering of the model, which has been found to be useful for both interpretation (i.e., $\beta$ in this model still reflects only dependence due to the underlying network, just as in (8.11)) and for improved fitting [5].
The specification of such models thus requires three pieces: the network process $X$ to be modeled, the network $G$, and the set of relevant exogenous variables (if any). Returning to our problem of predicting the protein function ICSC in the network ppi.CC, the first two specifications are accomplished in `ngspatial` through the following assignments.

```
#8.16 1 > X <- V(ppi.CC.gc)$ICSC
2 > A <- as_adjacency_matrix(ppi.CC.gc, sparse=FALSE)
```

The last specification depends on what additional information we wish to incorporate. For example, if, as in (8.11), we wish only to have an intercept, then we indicate this by

```
#8.17 1 > formula1 <- X ~ 1
```

Alternatively, biology tells us that various types of protein-specific information besides interactions can be useful in predicting protein function. An example is information on the genetic sequence underlying that gene coding for a given protein. For instance, genetic motifs are short sequences of DNA thought to have biological significance, such as by influencing the spatial configuration of proteins. Indicators of the presence or absence of six such motifs are included with the network ppi.CC as vertex attribute variables, each starting with the letters ‘IPR’. These are natural candidates for exogenous variables in our model.

```
#8.18 1 > gene.motifs <- cbind(V(ppi.CC.gc)$IPR000198,
2 + V(ppi.CC.gc)$IPR000403,
3 + V(ppi.CC.gc)$IPR001806,
4 + V(ppi.CC.gc)$IPR001849,
5 + V(ppi.CC.gc)$IPR002041,
6 + V(ppi.C.C.gc)$IPR003527)
7 > formula2 <- X ~ gene.motifs
```

We will see below that they indeed have nontrivial predictive power.

### 8.3.3 Inference and Prediction for Auto-Logistic Models

As with our treatment of the nearest-neighbor method, we focus again on the task of prediction of network processes $X$. However, unlike previously, we require knowledge of the handful of parameters in our models in order to generate predictions. This will, of course, be true of Markov random field models in general. In the specific context of the auto-logistic models (8.12), it is the parameters $\alpha$ and $\beta$ that are needed. Given measurements of $X$, we can try to infer these parameters from the data.

In principle, the task of inferring the vector $(\alpha^T, \beta)$ is most naturally approached through the method of maximum likelihood. But in practice this method often proves to be intractable. Consider, for example, the auto-logistic model without exogenous effects, in (8.10). The maximum likelihood estimate (MLE) of $(\alpha, \beta)$ is defined as
the value \((\hat{\alpha}, \hat{\beta})\) maximizing the log-likelihood \(\log P_{\alpha,\beta}(X = x)\), a task that can be shown to be equivalent to maximizing the expression

\[ \alpha M_1(x) + \beta M_{11}(x) - \kappa(\alpha, \beta). \]  

(8.13)

Here \(M_1(x)\) is the number of vertices with the attribute value 1, \(M_{11}(x)\) is twice the number of adjacent pairs of vertices where both have the attribute value 1, and \(\kappa(\alpha, \beta)\) is the partition function for this model, corresponding to the generic function defined in (8.5). Unfortunately, calculation of \(\kappa(\alpha, \beta)\) is prohibitive, as it requires evaluation of \(M_1\) and \(M_{11}\) across all \(2^{N_v}\) binary vectors \(x\) of length \(N_v\), with respect to the network graph \(G\).

The method of maximum pseudo-likelihood, originally proposed by Besag [4] for the analysis of spatial data, is a popular, computationally-feasible alternative to maximum likelihood in the context of MRF models. For the specific case of our auto-logistic models, instead of optimizing the marginal log-likelihood \(\log P_{\alpha,\beta}(X = x)\), we instead seek to maximize the so-called pseudo log-likelihood

\[ \sum_{i \in V} \log P_{\alpha,\beta}(X_i = x_i | X_{N_i} = x_{N_i}), \]  

(8.14)

which is the logarithm of the product of the conditional probabilities of each observed \(x_i\), given the values of its neighbors.

Importantly, these conditional probabilities do not involve the partition function \(\kappa(\alpha, \beta)\). For the auto-logistic model without exogenous effects, in (8.10), the maximum pseudo-likelihood estimate (MPLE) can be shown to be that value \((\hat{\alpha}, \hat{\beta})_{MPLE}\) which maximizes

\[ \alpha M_1(x) + \beta M_{11}(x) - \sum_{i=1}^{N_v} \log \left[ 1 + \exp \left( \alpha + \beta \sum_{j \in N_i} x_j \right) \right]. \]  

(8.15)

While the solution to this optimization does not have a closed-form expression, the piece defined by the summation over vertices \(i\), replacing \(\kappa(\alpha, \beta)\) in (8.13), can now be computed easily. In fact, the overall estimate can be computed using standard software for logistic regression, with the \(N_v\) pairs \((x_i, \sum_{j \in N_i} x_j)\) serving as response and predictor variables, respectively.

Maximum pseudo-likelihood estimates typically will differ from maximum likelihood estimates. Experience, however, has found them to be fairly accurate, as long as the dependencies inherent in the full joint distribution are not too substantial to be ignored. These estimates may be calculated for the centered auto-logistic model
defined in (8.12) using the autologistic function in \texttt{ngspatial}.\textsuperscript{5} For example, we can fit the simpler of our two models to the protein interaction data as follows.

```r
#8.19 1 > m1.mrf <- autologistic(formula1, A=A,
  2 + control=list(confint="none"))
```

With the resulting coefficients estimated\textsuperscript{6} to be

```r
#8.20 1 > m1.mrf$coefficients
2 (Intercept) eta
3 0.2004949 1.1351942
```

we see, for example, that in these data the addition of one neighboring protein with
the function ICSC is estimated to increase the log-odds of the ego protein having
ICSC by a factor of roughly 1.135.

In order to obtain some sense as to how effective we might be in predicting ICSC
using this model,\textsuperscript{7} we consider the simple prediction rule that a protein have ICSC
if the fitted probability $P_{\hat{\alpha},\hat{\beta}}(X_i = 1 | X_{N_i} = x_{N_i})$ is greater than 0.5,

```r
#8.21 1 > mrf1.pred <- as.numeric((m1.mrf$fitted.values > 0.5))
```

which yields an error rate of roughly 20%.

```r
#8.22 1 > mean(as.numeric(mrf1.pred != V(ppi.CC.gc)$ICSC))
2 [1] 0.2047244
```

This may be compared to the 25% error rate we witnessed with the nearest-neighbor
method. However, with respect to the seven proteins that were discovered to have
ICSC between 2007 and 2017, we find that this model and the nearest-neighbor
method are similar, i.e, the same four new annotations are correctly predicted.

```r
#8.23 1 > m1.mrf$fitted.values[V(ppi.CC.gc)$name %in% new.icsc]
2 [1] 0.7519142 0.1658647 0.2184092 0.6451897
3 [5] 0.9590030 0.2595863 0.3956048
```

\textsuperscript{5}In fact, the function \texttt{autologistic} also implements a Bayesian approach to estimation of the
parameters $\alpha$ and $\beta$. However, as this method is decidedly more computationally intensive, and
its implementation requires the use of parallel processing, we discuss only the pseudo-likelihood
method here, which is the default for \texttt{autologistic}.

\textsuperscript{6}It is also possible to obtain approximate confidence intervals of $\alpha$ and $\beta$ using \texttt{autologistic}. However, this option too requires the use of parallel processing and is thus omitted here to simplify
our exposition.

\textsuperscript{7}Note that if we observe only some of the elements of $X$, say $X^{obs} = x^{obs}$, and we wish to
predict the remaining elements $X^{miss}$, then formally we should do so based on the distribution
$P_{\alpha,\beta}(X^{miss} | X^{obs} = x^{obs})$. Explicit evaluation of this distribution will be prohibitive, but it is rela-
tively straightforward to simulate from this distribution using the Gibbs sampler, a type of Markov
chain Monte Carlo algorithm. See [13, Sect. 8.3.2.2] for a brief sketch of this approach.
The inclusion of gene motif information in our model

```r
#8.24 1 > m2.mrf <- autologistic(formula2, A=A,
2 + control=list(confint="none"))
```

actually leads to an estimated network effect that is greater than in the previous model, i.e., approximately 1.30.

```r
#8.25 1 > m2.mrf$coefficients
2 (Intercept) gene.motifs1 gene.motifs2 gene.motifs3
3 5.081573e-02 1.876848e+00 1.875217e+01 1.875217e+01
4 gene.motifs4 gene.motifs5 gene.motifs6 eta
5 1.824990e+01 8.487244e-08 -1.837997e+01 1.297921e+00
```

And the error rate improves slightly.

```r
#8.26 1 > mrf.pred2 <- as.numeric((m2.mrf$fitted.values > 0.5))
2 > mean(as.numeric(mrf.pred2 != V(ppi.CC.gc)$ICSC))
3 [1] 0.1889764
```

But perhaps most interestingly, this model appears to come much closer than the simpler model to correctly predicting ICSC function for five of the seven proteins of interest (using a threshold of 0.5).

```r
#8.27 1 > m2.mrf$fitted.values[V(ppi.CC.gc)$name %in% new.icsc]
2 [1] 0.7829254 0.4715219 0.4962188 0.6570828 0.7829254
3 [6] 0.2175373 0.3510037
```

### 8.3.4 Goodness of Fit

Similar to as was discussed earlier in Chap. 6, in the context of modeling network graphs, it is important to assess the goodness-of-fit here too in the context of MRF models. Again, at this point in time, simulation appears to be the primary tool available for this purpose. Here, given a fitted MRF model, we would like to simulate realizations of the network process \( X \) from this model. Various statistics summarizing characteristics of these realizations may then be computed and compared to what results when the same is done for the original data.

The function `rautologistic` in the `ngspatial` package can be used to simulate realizations of centered autologistic models. The following code simulates 100 realizations from each of the two auto-logistic models considered above for the prediction of the protein function ICSC. For each realization \( X \), the assortativity coefficient \( r_a \) is calculated with respect to the network `ppi.CC.gc`.

```r
#8.28 > set.seed(42)  # random seed for rautologistic
2 > ntrials <- 100
3 > a1.mrf <- numeric(ntrials)
4 > a2.mrf <- numeric(ntrials)
5 > Z1 <- rep(1,length(X))
6 > Z2 <- cbind(Z1, gene.motifs)
7 > for(i in 1:ntrials){
8 + X1.mrf <- rautologistic(as.matrix(Z1), A=A,
9 + theta=m1.mrf$coefficients)
10 + X2.mrf <- rautologistic(as.matrix(Z2), A=A,
11 + theta=m2.mrf$coefficients)
12 + a1.mrf[i] <- assortativity(ppi.CC.gc, X1.mrf+1,
13 + directed=FALSE)
14 + a2.mrf[i] <- assortativity(ppi.CC.gc, X2.mrf+1,
15 + directed=FALSE)
16 + }

The assortativity coefficient for the originally observed labels of ICSC function is roughly 0.37—fairly high, consistent with our findings that ICSC may be predicted reasonably well from protein–protein interactions.

#8.29 > assortativity(ppi.CC.gc, X+1, directed=FALSE)
2 [1] 0.3739348

Comparing this value to the distribution of those values obtained under our two models, we find that it falls in the upper quartile.

#8.30 > summary(a1.mrf)
2 Min. 1st Qu. Median  Mean 3rd Qu.  Max.
3 0.06736 0.22324 0.28733 0.27485 0.34203 0.46534
4 > summary(a2.mrf)
5 Min. 1st Qu. Median  Mean 3rd Qu.  Max.
6 -0.02656 0.20478 0.27621 0.26621 0.32519 0.45848

This suggests that the goodness-of-fit of our models, while not bad, can likely still be improved upon.

### 8.4 Kernel Methods

The probabilistic models we have just seen postulate a precise form for the dependency structure among vertex attributes $X_i$, with respect to the topology of the underlying graph $G$. But in some contexts, such as when prediction of unobserved vertex attributes is the only goal, it may be felt sufficient simply to ‘learn’ from the data a function relating the vertices to their attributes. The nearest-neighbor methods discussed in Sect. 8.2 in principle yield such a function, albeit implicitly. For a more explicit construction, a regression-based approach i.e., essentially regression on the graph $G$, would seem appealing. However, standard methods of regression, such as classical least squares regression, being set up as they are for relating response
and predictor variables in Euclidean space, are not immediately applicable to graph-indexed data.

Kernel methods have been found to be useful for extending the classical regression paradigm to various settings with non-traditional data. At the most basic level, these methods consist of (i) a generalized notion of predictor variables (i.e., encoded in a so-called ‘kernel’), and (ii) regression of a response on these generalized predictors using a penalized regression strategy. In this section we first introduce the notion of a kernel on a graph, and we then discuss the basic kernel approach to regression modeling in the context of graphs.

### 8.4.1 Designing Kernels on Graphs

At the heart of kernel methods is the notion of a kernel function. Broadly speaking, kernels can be thought of as functions that produce similarity matrices. The predictor variables used in the kernel regression are in turn derived from these similarity matrices. In the present context of kernel regression on network graphs, the kernel describes the similarity among vertices in the underlying graph \( G \). Since \( G \) itself often is defined to represent such similarities, it is common to construct kernels that summarize the topology of \( G \).

Formally, a function \( K \), receiving vertex pairs \((i, j)\) as input and returning a real value as output, is called a (positive semi-definite) kernel if, for each \( m = 1, \ldots, N_v \) and subset of vertices \( \{i_1, \ldots, i_m\} \in V \), the \( m \times m \) matrix \( K^{(m)} = [K(i_j, i_{j'})] \) is symmetric and positive semi-definite.

Although vertex proximity—and, hence, hopefully similarity—is naturally encoded in the adjacency matrix \( A \), it is more common to see the graph Laplacian used in the context of kernel regression. Recall from Sect. 4.4.2 that the graph Laplacian is defined as \( L = D - A \), where \( D = \text{diag}\{(d_v)\} \). The Laplacian kernel is defined simply as the (pseudo)inverse of the Laplacian, i.e., \( K = L^{-1} \).

More precisely, write \( L = \Phi \Gamma \Phi^T \), where \( \Phi \) and \( \Gamma = \text{diag}\{(\gamma_i)\} \) are \( N_v \times N_v \) orthogonal and diagonal matrices, respectively, arising through the eigen-decomposition of \( L \). Then the (pseudo)inverse is given by

\[
L^{-1} = \sum_{i=1}^{N_v} f(\gamma_i) \phi_i \phi_i^T, \tag{8.16}
\]

where \( \phi_i \) is the \( i \)-th column of \( \Phi \) and

\[
f(\gamma) = \begin{cases} 
\gamma^{-1}, & \text{if } \gamma \neq 0, \\
0, & \text{otherwise}.
\end{cases} \tag{8.17}
\]

\(8\) A matrix \( M \in \mathbb{R}^{m \times m} \) is positive semi-definite if \( x^T M x \geq 0 \), for all \( x \in \mathbb{R}^m \).
It is not difficult to show that $L^-$ is symmetric and positive semi-definite, and hence a proper kernel matrix.

Why might $L^-$ be a reasonable choice of kernel? We shall see shortly that in kernel regression the values of a vertex process $X$ on a graph $G$ are predicted through linear combinations of the eigen-vectors $\phi_i$. That is, through quantities of the form $h = \Phi \beta$. However, in order to better constrain the choice of such quantities, in fitting to data, penalized regression strategies are used, wherein $\beta^T \Gamma \beta = \sum_{i=1}^{N_v} \gamma_i \beta_i^2$ is encouraged to be ‘small’. But

$$\beta^T \Gamma \beta = \beta^T \Phi^T \Phi \Gamma \Phi^T \Phi \beta$$

$$= h^T L h .$$

(8.18)

Furthermore, it can be shown that

$$h^T L h = \sum_{\{i,j\} \in E} (h_i - h_j)^2 .$$

(8.19)

So $\beta^T \Gamma \beta$ in (8.18) will be small if and only if $h^T L h$ in (8.19) is small, which in turn is the case when the values of $h$ assigned to vertices $i$ and $j$ adjacent in $G$ are close, so as to reduce the magnitude of the differences $h_i - h_j$.

In other words, use of the kernel $K = L^-$ will encourage kernel regression to seek vectors $h = \Phi \beta$ that are locally ‘smooth’ with respect to the topology of $G$. In this sense, therefore, kernel regression can be made to behave similarly on network graphs to the manner in which we have already seen nearest-neighbor and Markov random field methods to behave.

By way of illustration, consider again the network ppi.CC of protein interactions. In Fig. 8.3 are shown the weights $f(\gamma_i)$ defining the Laplacian kernel $K = L^-$ in (8.16) for the giant connected component of this network.

```r
#8.31 1 > par(mfrow=c(1,1))
  2 > L <- as.matrix(laplacian_matrix(ppi.CC.gc))
  3 > e.L <- eigen(L)
  4 > nv <- vcount(ppi.CC.gc)
  5 > e.vals <- e.L$values[1:(nv-1)]
  6 > f.e.vals <- c((e.vals)^(-1), 0)
  7 > plot(f.e.vals, col="magenta", xlim=c(1, nv),
         xlab=c("Index i"), ylab=expression(f(gamma[i])))
```

Because this subgraph is connected, the smallest eigenvalue $\gamma_{min}$ of $L$ is zero, and all others are positive. Upon inversion of the nonzero eigenvalues, through application of the function $f$ in (8.17), we see that there are a relatively small percentage (e.g., the last 20 or so) of rather large weights $f(\gamma_i)$, with the rest being comparatively much smaller.

As a result of this behavior of the weights, we know that the structure of $K$ is largely governed by the structure of a correspondingly small percentage of eigen-vectors. In Fig. 8.4 is shown a visual representation of the eigen-vectors corresponding to the three largest weights. The first is produced through the following code.
8.4 Kernel Methods

**Fig. 8.3** Plot of the weights $f(\gamma_i)$ defining the Laplacian kernel $K = L^-$ in (8.16)

**Fig. 8.4** Visual representation of the eigenvectors $\phi_i$ corresponding to the largest, second largest, and third largest (left to right) weights $f(\gamma_i)$ for the protein interaction network `ppi.CC.gc`. Negative values are shown in blue, and positive values, in red, with the area of each vertex proportional to the magnitude of its entry in the corresponding eigenvector.

```r
#8.32 1 > e.vec <- e.L$vectors[, (nv-1)]
2 > v.colors <- character(nv)
3 > v.colors[e.vec >= 0] <- "red"
4 > v.colors[e.vec < 0] <- "blue"
5 > v.size <- 15 * sqrt(abs(e.vec))
6 > l <- layout_with_fr(ppi.CC.gc)
7 > plot(ppi.CC.gc, layout=l, vertex.color=v.colors,
8 + vertex.size=v.size, vertex.label=NA)
```

The other two are produced similarly, with `nv-1`, in the first line, replaced by `nv-2` and `nv-3`, respectively. For each of these eigenvectors we see evidence of large clusters of vertices with values of similar sign and magnitude, indicative of smooth behavior on the network graph.
We will use the R package `kernlab` to apply kernel regression methods to our problem of modeling and predicting processes on network graphs.

```r
#8.33 1 > library(kernlab)
```

This package contains implementations of a variety of kernel-based machine learning methods. The Laplacian kernel for our protein–protein interaction network is constructed and declared a kernel object as follows.

```r
#8.34 1 > K1.tmp <- e.L$vectors %*% diag(f.e.vals) %*% t(e.L$vectors)
3 > K1 <- as.kernelMatrix(K1.tmp)
```

The notion of a kernel is quite general and so, not surprisingly, there are a variety of other ways that a kernel can be defined in our context. For example, Smola and Kondor [19] introduced a more general class of kernels, based on a certain notion of ‘regularization,’ wherein the function $f$ in (8.17) is replaced by various other choices of smooth function. Alternatively, we may wish to combine multiple sources of information in constructing our kernel. A common approach to doing so is to encode each separate source of information into its own kernel and to then define the kernel $K$ as a convex combination of those kernels (so as to maintain symmetry and positive definiteness).

For example, recall that inclusion of the gene motif information in the matrix `gene.motifs` was useful for predicting ICSC protein function with Markov random field models. A straightforward way in which to encode this information in a kernel is through the use of inner products, resulting in a so-called inner-product kernel.

```r
#8.35 1 > K.motifs <- gene.motifs %*% t(gene.motifs)
```

Giving equal weight\(^9\) to both this and our Laplacian kernel results in a kernel that now incorporates both endogenous and exogenous network information.

```r
#8.36 1 > K2.tmp <- 0.5 * K1.tmp + 0.5 * K.motifs
2 > K2 <- as.kernelMatrix(K2.tmp)
```

### 8.4.2 Kernel Regression on Graphs

We now formalize the notion of kernel regression on graphs and illustrate the type of performance that can be obtained in the context of our prediction problem. Let $G = (V, E)$ be a network graph and $X = (X_1, \ldots, X_{N_v})$ a vertex attribute process. From the perspective of kernel regression, our goal is to learn from the data an appropriate function, say $\hat{h}$, mapping from $V$ to $\mathbb{R}$, that describes well the manner in which attributes vary across the vertices. More precisely, given a kernel $K$, with

\(^9\) More ambitiously, these weights in turn may be chosen in a data-adaptive fashion. See, for example, [7, 8, 14, 16, 20].
eigen-decomposition $\mathbf{K} = \mathbf{\Phi} \Delta \mathbf{\Phi}^T$, in kernel regression we seek to find an optimal choice of $\mathbf{h}$ within the class

$$\mathcal{H}_K = \{ \mathbf{h} : \mathbf{h} = \mathbf{\Phi} \mathbf{\beta} \text{ and } \mathbf{\beta}^T \Delta^{-1} \mathbf{\beta} < \infty \} ,$$

where $\mathbf{h}$ is an $N_v$-length vector.

In order to choose an appropriate element $\mathbf{h}$ in $\mathcal{H}_K$, say $\hat{\mathbf{h}}$, a penalized regression strategy is employed in kernel regression in an effort to enforce that $\hat{\mathbf{h}}$ both be close to the observed data and be sufficiently smooth (i.e., in the sense of (8.19) or, equivalently, (8.18), being small). Specifically, an estimate $\hat{\mathbf{h}} = \mathbf{\Phi} \hat{\mathbf{\beta}}$ is produced by selecting that $\hat{\mathbf{\beta}}$ that minimizes

$$\sum_{i \in V_{obs}} C(\mathbf{x}_i ; (\mathbf{\Phi} \mathbf{\beta})_i) + \lambda \mathbf{\beta}^T \Delta^{-1} \mathbf{\beta} ,$$

where $V_{obs} \subseteq V$ denotes the set of vertices $i$ at which we have observations $X_i = x_i$, $C(\cdot ; \cdot)$ is a convex function that measures the loss incurred through predicting its first argument by its second, $(\mathbf{\Phi} \mathbf{\beta})_i$ denotes that element of $\mathbf{\Phi} \mathbf{\beta}$ corresponding to $i \in V_{obs}$, and $\lambda$ is a tuning parameter.

The optimization in (8.21) is a type of complexity-penalized estimation strategy. The role of the predictor variable is played by the columns of the matrix $\mathbf{\Phi}$ (i.e., the eigenvectors of the kernel matrix $\mathbf{K}$), and that of the response variable, by the observed elements of $\mathbf{X}$. The loss captured by $C(\cdot ; \cdot)$ encourages goodness of fit in the model, while the term $\mathbf{\beta}^T \Delta^{-1} \mathbf{\beta}$ penalizes excessive complexity, in the sense that eigenvectors with small eigenvalues are penalized more harshly than those with large eigenvalues. The parameter $\lambda$ dictates the relative importance of the loss versus the complexity penalty, and is typically chosen in a data-dependent fashion.

For certain choices of loss, the optimal solution to the minimization of the expression in (8.21) has a closed form. In general, however, numerical methods must be used. The details of the implementation depend on the specific choices of loss function $C(\cdot ; \cdot)$. In kernlab, for the problem of regression with binary response variables, a standard choice of loss function is implemented in the function ksvm. Specifically, the loss

$$C(x; h) = \left[ \max (0, 1 - \tilde{x}h) \right]^2$$

is used, where $\tilde{x} = 2x - 1$, mapping the values $x = 0$ or 1 to $-1$ and $+1$, respectively. This choice corresponds to what is known as a 2-norm soft-margin support vector machine.

Returning to our motivating problem of predicting ICSC using our network of protein interactions, we use the Laplacian kernel $\mathbf{K}_1$ defined earlier as input to ksvm and extract the resulting fitted values, which ksvm produces in the form of 0 or 1 values.10

---

10Alternatively, using the option prob.model=TRUE, probabilities may be calculated.
Comparing these fitted values to the originally observed indicators of ICSC function, we see that this kernel regression produces an error rate of 11%, roughly half of that of the analogous MRF model.

```r
> mean(as.numeric(m1.svm.fitted != V(ppi.CC.gc)$ICSC))
[1] 0.1102362
```

Furthermore, the model accurately predicts ICSC function for four of the seven proteins of interest.

```r
> m1.svm.fitted[V(ppi.CC.gc)$name %in% new.icsc]
[1] 1 1 1 0 1 0 0
```

If we now incorporate gene motif information as well, by using the kernel K2 instead of K1,

```r
> m2.svm <- ksvm(K2, X, type="C-svc")
> m2.svm.fitted <- fitted(m2.svm)
> mean(as.numeric(m2.svm.fitted != V(ppi.CC.gc)$ICSC))
[1] 0.06299213
```

although in this case, only two of the seven proteins of interest are predicted correctly.

```r
> m2.svm.fitted[V(ppi.CC.gc)$name %in% new.icsc]
[1] 1 0 0 0 1 0 0
```

### 8.5 Modeling and Prediction for Dynamic Processes

As remarked earlier in this chapter, many processes defined on networks are more accurately thought of as dynamic, rather than static, processes. Examples include a cascade of failures (e.g., as an electrical power-grid strains under a heat-wave), the diffusion of knowledge (e.g., as a rumor spreads in a population), the search for information (e.g., as an Internet-based search engine formulates a response to a query), the spread of disease (e.g., as a virus propagates through a population of humans or computers), the synchronization of behavior (e.g., as neurons fire in the brain), and the interaction of ‘species’ in an environment (e.g., as genes in a cell auto-regulate among themselves).

Conceptually, we may think of processes like these as time-indexed vertex attribute processes \( X(t) = (X_i(t))_{i \in V} \), with \( t \) varying in a discrete or continuous manner over a range of times. Both deterministic and stochastic perspectives are commonly adopted for modeling such processes. Deterministic models are based on difference
and differential equations, whereas stochastic models are based on time-indexed stochastic processes—usually Markov processes.\textsuperscript{11}

While there has been a substantial amount of work done in the past 15–20 years on the mathematical and probabilistic modeling of dynamic processes on network graphs (see [2], for example, for a survey), there has been comparatively much less work on the statistics. As a result, our treatment of this topic here will be similarly limited in scope, with the goal being simply to take a quick peek at the modeling and simulation of such processes. We illustrate within the context of one particular class of dynamic process—epidemic processes.

8.5.1 Epidemic Processes: An Illustration

The term \textit{epidemic} refers to a phenomenon that is prevalent in excess to what might be expected. It is most commonly used in the context of diseases and their dissemination throughout a population—such as with malaria, COVID-19, and AIDS—but it is also at times used more broadly in other contexts, such as in describing the spread of perceived problems in a society or the adoption of a commercial product.

Epidemic modeling has been an area of intense interest among researchers working on network-based dynamic process models. We begin our discussion here by briefly introducing a classical model from traditional (i.e., non-network) epidemic modeling. We then examine a network-based analogue.

Traditional Epidemic Modeling

The most commonly used class of continuous-time epidemic models is the class of \textit{susceptible-infected-removed} (SIR) models. In this section, we will focus on the stochastic formulation of what is arguably the simplest member of this class—the so-called \textit{general epidemic model}.

Imagine a (closed) population of, say, \(N + 1\) elements such that, at any point in time \(t\), there are some random number \(N_S(t)\) of elements susceptible to infection (called ‘susceptibles’), \(N_I(t)\) elements infected (called ‘infectives’), and \(N_R(t)\) elements recovered and immune (or, alternatively, removed). Starting with one infective and \(N\) susceptibles, that is, with \(N_I(0) = 1\) and \(N_S(0) = N\), and letting \(s\) and \(i\) generically denote some numbers of susceptibles and infectives, respectively, it is assumed that the triple \((N_S(t), N_I(t), N_R(t))\) evolves according to the instantaneous transition probabilities

\textsuperscript{11}Informally speaking, a \textit{Markov process} is a stochastic process in which the future of the process at any given point in time depends only upon its present state and not its past.
\begin{align}
\mathbb{P}(N_S(t + \delta t) = s - 1, N_I(t + \delta t) = i + 1 | N_S(t) = s, N_I(t) = i) & \approx \beta s i \delta t \\
\mathbb{P}(N_S(t + \delta t) = s, N_I(t + \delta t) = i - 1 | N_S(t) = s, N_I(t) = i) & \approx \gamma i \delta t \\
\mathbb{P}(N_S(t + \delta t) = s, N_I(t + \delta t) = i | N_S(t) = s, N_I(t) = i) & \approx 1 - (\beta s + \gamma) i \delta t ,
\end{align}

where \( \delta t \) refers to the usual infinitesimal and the role of \( N_R(t) \) is omitted due to the constraint \( N_S(t) + N_I(t) + N_R(t) = N + 1 \).

The above model states that, at any given time \( t \), a new infective will emerge from among the susceptibles (due to contact with and infection by one of the infectives) with instantaneous probability proportional to the product of the number of susceptibles \( s \) and the number of infectives \( i \). Similarly, infectives recover with instantaneous probability proportional to \( i \). These probabilities are scaled by the parameters \( \beta \) and \( \gamma \), usually referred to as the infection and recovery rates, respectively. The product form for the probability with which infectives emerge corresponds to an assumption of ‘homogeneous mixing’ (or ‘mass-action,’ in chemistry) among members of the population, which asserts that the population is (i) homogeneous and (ii) well mixed, in the sense that any pair of members are equally likely to interact with each other.

An equivalent formulation of the model states that, given \( s \) susceptibles and \( i \) infectives at time \( t \), the process remains in the state \((s, i)\) for an amount of time distributed as an exponential random variable, with rate \( (\beta s + \gamma) i \). A transition then occurs, which will be either to the state \((s - 1, i + 1)\), with probability \( \beta s / [(\beta s + \gamma) i] \), or to the state \((s, i - 1)\), with probability \( \gamma i / [(\beta s + \gamma) i] \).

Figure 8.5 shows a schematic characterization of the typical behavior of our stochastic SIR process under simulation.\(^{12}\) Starting with a population composed almost entirely of susceptibles and just a few infectives, we see an initial exponential increase in the number of infectives and a corresponding decrease in the number of susceptibles. This initial period is followed by a peak in the number of infectives, after which this number decays exponentially, as the supply of susceptibles is depleted and the infectives recover.

Unfortunately, despite the fact that the general epidemic SIR model captures the gross characteristics of a canonical epidemic, the underlying assumption of homogeneous mixing is admittedly simple and, for many diseases, too poor of an approximation to reality. A key element lacking from these models is the natural structure often inherent to populations. Such structure might derive from spatial proximity (e.g., diseases of plants, in which infection occurs through the help of carriers over short distances), social contact (e.g., sexual contact in the transmission of AIDS), or demographics (e.g., households, age brackets, etc.). More sophisticated models assume contact patterns that take into account such structure(s) within the population of interest. And frequently it is convenient to represent these patterns in the form of a graph.

\(^{12}\)Tools in the \texttt{R} package \texttt{EpiModel} may be used to generate and analyze such curves, as well as curves corresponding to individual epidemic processes. As our primary focus is on network-based epidemics, however, we do not explore those tools here.
Network-Based Epidemic Modeling

Let $G$ be a network graph describing the contact structure among $N_v$ elements in a population. We assume that initially, at time $t = 0$, one vertex is infected and the rest are susceptible. Infected vertices remain infected for an amount of time distributed exponentially, with rate $\gamma$, after which they are considered recovered. During the infectious period a vertex has infectious contacts independently with each neighbor, according to an exponential distribution with rate $\beta$, where an infectious contact automatically results in infection if the other individual is susceptible. Define $X_i(t) = 0, 1, 2$, according to whether vertex $i$ is susceptible, infected, or removed at time $t$, respectively.

Let $\mathbf{X}(t) = (X_i(t))_{i \in V}$ be the resulting (continuous) time-indexed process on the network graph $G$. Denote by $\mathbf{x}$ the state of the process at a given time $t$ (i.e, the particular pattern of 0’s, 1’s, and 2’s across the vertices in $G$ at time $t$). Successive changes of states, say from $\mathbf{x}$ to $\mathbf{x}'$, will involve a change in one and only one element at a time. Suppose that $\mathbf{x}$ and $\mathbf{x}'$ differ in the $i$-th element. Then it may be shown that the model just described is equivalent to specifying that

$$
\mathbb{P} \left( \mathbf{X}(t + \delta t) = \mathbf{x}' \mid \mathbf{X}(t) = \mathbf{x} \right) \approx \begin{cases} 
\beta M_i(\mathbf{x}) \delta t, & \text{if } x_i = 0 \text{ and } x'_i = 1, \\
\gamma \delta t, & \text{if } x_i = 1 \text{ and } x'_i = 2, \\
1 - (\beta M_i(\mathbf{x}) + \gamma) \delta t, & \text{if } x_i = 2 \text{ and } x'_i = 2,
\end{cases}
$$

(8.24)

13Technically, independence pertains only to sufficiently small time intervals. Over the life of the epidemic, the events that an element in the population infects two neighbors are not independent, but rather positively correlated.

14Formally, $\mathbf{X}(t)$ constitutes a continuous-time Markov chain.
where we define $M_i(x)$ to be the number of neighbors $j \in N_i$ for which $x_j = 1$ (i.e., the number of neighbors of $i$ infected at time $t$). Our network-based analogue of the traditional SIR process then follows by defining the processes $N_S(t)$, $N_I(t)$, and $N_R(t)$, counting the numbers of susceptible, infective, and removed vertices at time $t$, respectively, in analogy to the traditional case.

In light of the expressions in (8.24), we can expect the characteristics of the processes $N_S(t)$, $N_I(t)$, and $N_R(t)$ to be affected to at least some extent by the characteristics of the network graph $G$. Simulation can be used to confirm this expectation.

We begin by generating examples of three different random graphs introduced earlier, in Chap. 5.

```r
#8.43 1 > set.seed(42)
2 > gl <- list()
3 > gl$ba <- sample_pa(250, m=5, directed=FALSE)
4 > gl$er <- sample_gnm(250, 1250)
5 > gl$ws <- sample_smallworld(1, 250, 5, 0.01)
```
The parameters have been chosen so as to guarantee graphs of the same number of vertices and roughly the same average degree (i.e., around 10), since both are basic characteristics expected to fundamentally affect the progression of an epidemic in ways that do not reflect interesting differences in topology.

Setting the infection rate to be $\beta = 0.5$, and the recovery rate, to be $\gamma = 1.0$,

```r
#8.44 1 > beta <- 0.5
2 > gamma <- 1
```
we use the function `sir` in `igraph` to produce

```r
#8.45 1 > ntrials <- 100
```
simulated epidemics on each network.

```r
#8.46 1 > sim <- lapply(gl, sir, beta=beta, gamma=gamma,
2 + no.sim=ntrials)
```
The output from each simulation is an `sir` object, containing information about the times at which changes of states occurred and the values of the processes $N_S(t)$, $N_I(t)$, and $N_R(t)$ at those times.

The results of plotting the total number of infectives $N_I(t)$ for each of these three networks is shown in the first three panels of Fig. 8.6.

```r
#8.47 1 > plot(sim$er)
2 > plot(sim$ba, color="palegoldenrod",
3 + median_color="gold", quantile_color="gold")
4 > plot(sim$ws, color="pink", median_color="red",
5 + quantile_color="red")
```
Individual simulation paths, as well as their medians and 10 and 90% quantiles, are shown in each panel.
For all three network topologies we observe an exponential rise and decay that is qualitatively similar to that of a traditional SIR epidemic process, as shown in Fig. 8.5. Nevertheless, it is clear that they also differ in important ways, driven by differences in network topology. For example, the epidemic peaks earlier in the small-world network and the Barabási–Albert network than in the Erdős and Rényi network, yet the latter two networks yield substantially more infected nodes than the former. These differences may be better seen plotting the median of the curves $N_I(t)$, for each graph, on one plot.
8.4.8 1 > x.max <- max(sapply(sapply(sim, time_bins), max))
2 > y.max <- 1.05 * max(sapply(sapply(sim, function(x)
3 + median(x)["NI"], max, na.rm=TRUE))
4
5 > plot(time_bins(sim$er), median(sim$er)["NI"],
6 + type="l", lwd=2, col="blue", xlim=c(0, x.max),
7 + ylim=c(0, y.max), xlab="Time",
8 + ylab=expression(N[I](t))
9 > lines(time_bins(sim$ba), median(sim$ba)["NI"],
10 + lwd=2, col="gold")
11 > lines(time_bins(sim$ws), median(sim$ws)["NI"],
12 + lwd=2, col="red")
13 > legend("topright", c("ER", "BA", "WS"),
14 + col=c("blue", "gold", "red"), lty=1)

8.6 Additional Reading

General discussion of nearest neighbor methods may be found in Hastie et al. [10, Sect. 2.3.2]. For an extensive development of Markov random fields, see Cressie [6, Sect. 6.4], where the emphasis is on modeling data on spatial lattices. Standard references on kernel methods include Schölkopf and Smola [17] and Shawe-Taylor and Cristianini [18]. Finally, for a general introduction to epidemic modeling, see Daley and Gani [9], for example, or Anderson and May [1], for a more in-depth treatment.

References

1. R. Anderson and R. May, Infectious Diseases of Humans: Dynamics and Control. Oxford: Oxford University Press, 1991.
2. A. Barrat, M. Barthélemy, and A. Vespignani, Dynamical Processes on Complex Networks. New York: Cambridge University Press, 2008.
3. J. Besag, “Spatial interaction and the statistical analysis of lattice systems,” Journal of the Royal Statistical Society, Series B, vol. 36, no. 2, pp. 192–236, 1974.
4. J. Besag, “Statistical analysis of non-lattice data,” The Statistician, vol. 24, no. 3, pp. 179–195, 1975.
5. P. C. Caragea and M. S. Kaiser, “Autologistic models with interpretable parameters,” Journal of Agricultural, Biological, and Environmental Statistics, vol. 14, no. 3, pp. 281–300, 2009.
6. N. Cressie, Statistics for Spatial Data. New York: Wiley-Interscience, 1993.
7. N. Cristianini, J. Kandola, A. Elisseeff, and J. Shawe-Taylor, “On kernel-target alignment,” in Innovations in Machine Learning: Theory and Applications, D. Holmes and L. Jain, Eds. New York: Springer-Verlag, 2006.
8. N. Cristianini, J. Shawe-Taylor, A. Elisseeff, and J. Kandola, “On kernel-target alignment,” Advances in Neural Information Processing Systems 14, 2002.
9. D. Daley and J. Gani, Epidemic Modeling. New York: Cambridge University Press, 1999.
10. T. Hastie, R. Tibshirani, and J. Friedman, The Elements of Statistical Learning. New York: Springer, 2001.
11. E. Ising, “Beitrag zur theorie des ferromagnetismus,” Zeit. fur Physik, vol. 31, pp. 253–258, 1925.
12. X. Jiang, N. Nariai, M. Steffen, S. Kasif, and E. Kolaczyk, “Integration of relational and hierarchical network information for protein function prediction,” *BMC Bioinformatics*, vol. 9, no. 350, 2008.
13. E. Kolaczyk, *Statistical Analysis of Network Data: Methods and Models*. Springer Verlag, 2009.
14. G. Lanckriet, N. Cristianini, P. Bartlett, L. El Ghaoui, and M. Jordan, “Learning the kernel matrix with semidefinite programming,” *Journal of Machine Learning Research*, vol. 5, pp. 27–72, 2004.
15. S. Li, *Markov Random Field Modeling in Computer Vision*. New York: Springer-Verlag, 1995.
16. Y. Lin and H. Zhang, “Component selection and smoothing in multivariate nonparametric regression,” *Annals of Statistics*, vol. 34, no. 5, pp. 2272–2297, 2006.
17. B. Schölkopf and A. Smola, *Learning with Kernels: Support Vector Machines, Regularization, Optimization, and Beyond*. Cambridge, MA: MIT Press, 2002.
18. J. Shawe-Taylor and N. Cristianini, *Kernel Methods for Pattern Analysis*. Cambridge University Press, 2004.
19. A. Smola and R. Kondor, “Kernels and regularization on graphs,” in *Proceedings of the 16th Annual Conference on Learning Theory (COLT)*, 2003.
20. H. Zhang and Y. Lin, “Component selection and smoothing for nonparametric regression in exponential families,” *Statistica Sinica*, vol. 16, pp. 1021–1042, 2006.