ST433 Machine Learning and Data Mining
London School of Economics

Candidate Number:
41524
46930
36536
33874

Nov 2019

1 Bike Sharing in London

1.1 Overview

Our bike-sharing dataset consists of 17414 observations and 12 variables. We want to fit a model to accurately predict the count of bike shares in London, given the time of day, weather conditions and other factors such as weekend, holiday and season of the year. The aim is to design a model with good balance of interpretability and predictive power.

Before applying any machine learning approaches, we checked the ACF and PACF of the residual of the count. As they are both low, it is safe for us to proceed with machine learning methods. We divided our dataset into training set using 80% of observations, while the rest is used as a testing set.

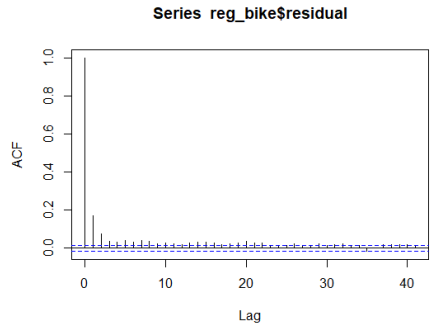

Figure 1: ACF plot of Residual

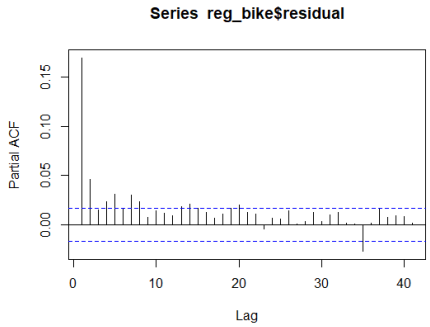

Figure 2: PACF plot of Residual

1.2 Review of Various Approaches

1.2.1 Ridge Regression

To avoid overfitting of the model, we used Ridge Regression with its coefficients shrunken towards zero. Then it provides MSE of 341,420. As Ridge Regression can not perform variable selection, it can not set any coefficients to zero and can not prove an interpretable model. To solve this problem, we used the next model, Lasso Regression.

1.2.2 Lasso Regression

Lasso Regression uses penalty term and it can make coefficients perform a continuous shrinking to zero. We did variable selection first, and the results show that all variables are significant. Then we did Lasso regression under Poisson distribution. We also did cross-validation to select best lambda which is 1.571184 in Lasso Regression. As can be observed, Lasso regression has a better performance and Lasso Regression provides MSE of 267,148 which is much lower than 341,420 in Ridge Regression.

1.2.3 Linear Regression

A multiple linear regression was carried out using the training set. The model had an R-squared value of 0.7094 which is reasonably well. However, the normal QQ plot indicates that residuals are not normally distributed, violating a key assumption of linear regression. Additionally, when we apply our testing set to the model, it has a relatively high MSE of 317,334.

1.2.4 K Nearest Neighbour

The next approach that we implemented was the K-Nearest Neighbour algorithm. After plotting RMSE against number of neighbours, the best tuning parameter, k=5 was chosen. However, the model still yielded a large value of MSE, 939,034 when tested against the testing set. Thus, we conclude that K-Nearest Neighbour is a very poor model to predict the number of future bike shares.

1.2.5 Poisson Regression

Poisson Regression is a form of generalised linear model used to model count data in statistics. Poisson Regression is one of our candidates because we are dealing with count data. Although it provides a better MSE of 264,081, it proved unsuitable because the difference between the mean and variance is extremely large (mean is 1143, variance is 1177460). As the key assumption of Poisson Regression is violated, we decided to try alternative methods.

1.2.6 Negative Binomial Regression

Negative Binomial Regression relaxes the the constraining assumption that mean equals to variance in the Poisson model. Using similar techniques applied in Poisson Regression, the MSE yielded is 329,233, indicating that Negative Binomial is does not fit as good as Poisson.

1.2.7 Summary of Different Approaches and their MSE

Linear Regression	317,334
K-Nearest Neighbours	939,034
Poisson Regression	264,081
Negative Binomial Regression	329,420
Ridge Regression	341,420
Lasso Regression	267,148

1.3 Summary of the final approach

The final approach we used is Random Forest which has a better performance compared to the previous models. Regression tree minimises the RSS of the recursive binary split in the following

way,

$$\sum_{j=1}^{J} \sum_{i \in R_j} \left(y_i - \widehat{y}_{R_j}\right)^2$$

where $\widehat{y}_{R_j}$ is the mean response for the training observation within j-th box. We conducted a K-fold cross-validation to find the deviance by using the cost-complexity parameter k = 5. After we did decision tree pruning, we got the MSE of 304298.8.

We decided the number of variables randomly sampled equals 4. The Bagging algorithm achieved on testing set with the lowest MSE of 54428.97 among those models and the Boosting algorithm based on Gaussian distribution also got a relatively low MSE of 47816.16.

As shown in the Figure 3, Bagging got a better fitting result than Boosting.

In Bagging, we used Extract variable importance measurement. MSE and residual sum of squares are measured to compute the variable importance. The results show that Time and Weekend are the two most significant variables.

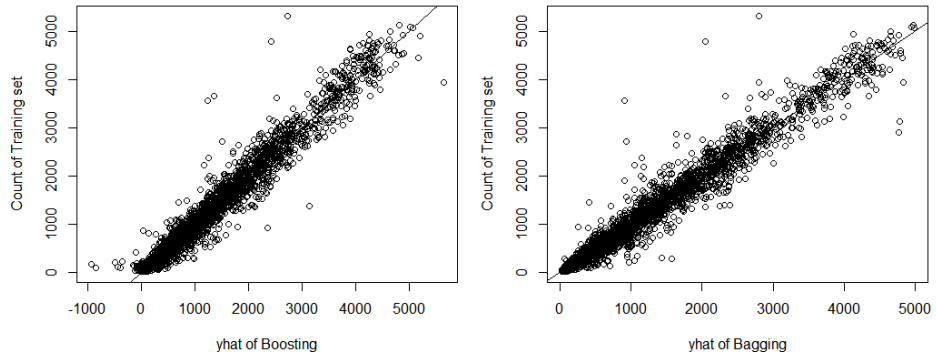

Figure 3: Boosting vs Bagging

1.4 Conclusion

This study predicted the sharing bike usage in London by using different regression models. By comparing MSE of those models, we decide to use Bootstrap Aggregation as our final choice.

As a result of categorizing data, we can see that the number of sharing bike used is different depending on time, weather, weekend, etc. The result of Bootstrap Aggregation also suggests that Time and Weekend are the two most significant features.

Based on these results, Bootstrap Aggregation is useful to predict the usage of sharing bike in order to improve the quality of public transportation.

2 Estimation of graphical models using lasso-related approaches

2.1 Introduction of graphical lasso and node-wise lasso

2.1.1 Graph, nodes and edges

A graph is a specific mathematical object comprising of a set of vertices(nodes) and edges. Related pairs of nodes are linked by edges. If the edges in a graph have directional arrows, then it is a directed graph, otherwise it is an undirected graph. This report only discusses undirected graphical models. Below is an example of undirected graph.

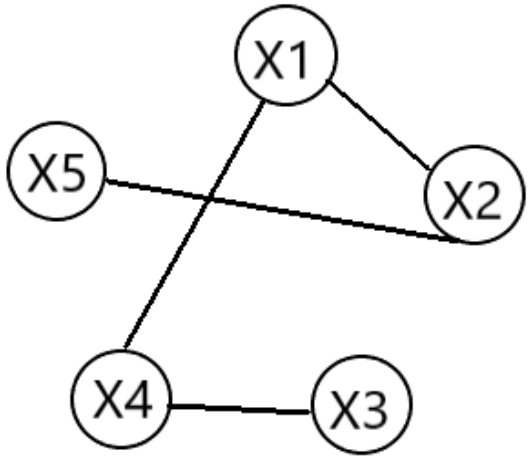

Figure 4: An example of undirected graph

According to Figure 4, the graph has 5 vertices(nodes) and 4 edges. X1 and X2 are connected by an edge, indicating that they are related.

Graphical models are used to measure the conditional dependence structure among p random variables, representing p nodes in a graph, by recovering the edge set. For example, denote

$$\mathrm{c_{jl}} = \mathrm{Cov}\left(X_j, X_l | X_k, 1 \leq k \leq p, k \neq j, l\right) \text{ for } 1 \leq j, l \leq p$$

representing the covariance between X_j and X_l conditioned on the remaining variables. Then node j and node l are connected by an edge if and only if $c_{jl} \neq 0$. Let $G = (V, E)$ represents an undirected graph with vertex set $V = 1, \ldots, p$ and edge set is determined by the covariance between two random variables.

$$\mathrm{E} = \{(\mathrm{j}, \mathrm{l}) : \mathrm{c_{jl}} \neq 0, 1 \leq \mathrm{j}, 1 \leq \mathrm{p}, \mathrm{j} \neq 1\}$$

2.1.2 Node-wise lasso

Node-wise lasso approach is rather straightforward. More specifically, for node $j \in V$, build regression model of X_j with the remaining variables $X_l, l \neq j$,

$$\mathrm{X_j} = \sum_{1 \leq l \leq p, l \neq j} \beta_{jl} X_l + \varepsilon_{\mathrm{jl}}$$

With lasso penalty function,

$$\sum_{i=1}^{n} \left(x_{ij} - \sum_{1 \leq l \leq p, l \neq j} \beta_{jl} X_{il} \right) + \lambda \sum_{l} |\beta_{jl}|$$

Lasso approach enables a sparse solution in finding the linked nodes in the graph. The estimator $\widehat{\beta}_{jl}$ is used to determine if node j and node l are connected. The following two rules are used in the report.

Node-wise lasso 1

$$\hat{E}_1 = \{(j,l) : \beta_{jl} \neq 0 \text{ and } \beta_{lj} \neq 0, 1 \leq j, 1 \leq p, j \neq l\}$$

Node-wise lasso 2

$$\hat{E}_2 = \{(j,l) : \beta_{jl} \neq 0 \text{ or } \beta_{lj} \neq 0, 1 \leq j, l \leq p, j \neq l\}$$

Since $\widehat{E}$ is a symmetric matrix, the first rule is stricter than the second, and it is expected that fewer edges are discovered under node-wise lasso 1 approach.

2.1.3 Graphical lasso

The graphical lasso is under the assumption that $\boldsymbol{X}$ is multivariate Gaussian with covariance matrix $\boldsymbol{\Sigma} \in \mathbb{R}^{pxp}$. We denote $\boldsymbol{\Theta} = \boldsymbol{\Sigma}^{-1}$, thus

$$\mathrm{c_{jl}} = 0 \text{ is equivalent to } \Theta_{\mathrm{jl}} = 0$$

And the edge set can be interpreted as

$$\widehat{E}_3 = \left\{ (j,l) : \widehat{\Theta}_{jl} \neq 0, 1 \leq j, l \leq p, j \neq l \right\}$$

2.2 The selection of optimal tuning parameters

This report uses two indices in choosing the optimal tuning parameter, one of which is the overall accuracy rate, and the other one is F1 score. These two measurements are based on confusion matrix.

Table 1: Confusion matrix

		Predicted	
		0	1
True	0	True Negative	False Positive (Type-I Error)
	1	False Negative (Type-II Error)	True Positive

2.2.1 Accuracy rate

$$\text{ACC} = \frac{\text{TN} + \text{TP}}{\text{TN} + \text{FP} + \text{FN} + \text{TP}}$$

Accuracy rate describes the proportion of correct classifications. It somehow reveals the overall performance of a prediction. However, it is not reliable when the data set is unbalanced.

2.2.2 F1 score

F1 score is the harmonic mean of true positive rate (TPR) and positive predictive value (PPV). As a complementary to accuracy rate, it also reflects the overall performance of a predictor.

$$\text{TPR} = \frac{\text{TP}}{\text{TP} + \text{FN}}$$

$$\text{PPV} = \frac{\text{TP}}{\text{TP} + \text{FP}}$$

$$\text{F1} = \frac{2}{\frac{1}{\text{TPR}} + \frac{1}{\text{PPV}}} = 2 * \frac{\text{TPR} * \text{PPV}}{\text{TPR} + \text{PPV}}$$

2.3 Simulation settings and numerical results under dierent sparsity structures

Generate original edge set $\mathbf{\Theta} = \mathbf{B} + \delta \mathbf{I}_p \in \mathbb{R}^{pxp}$, where the off-diagonals of symmetric matrix B are generated randomly and independently with probability 0.1 to be 0.5 and probability 0.9 to be 0, representing if node j and l are connected by an edge. $\mathbf{I}_p$ is identity matrix and $\delta > 0$ so that $\mathbf{\Theta}$ is positive definite and thus invertible. $\mathbf{\Theta}$ reflects the true edge set in E.

Since node-wise lasso approach and graphical lasso approach are under the assumption that $\mathbf{X}$ is subject to multivariate Gaussian distribution. We generate n random samples from a multivariate Gaussian distribution with mean zero and covariance matrix $\mathbf{\Sigma} = \mathbf{\Theta^{-1}}$.

We use the criteria introduced in Section 2.2 to determine an optimal tuning parameter for each method and replicate the experiment 50 times. Finally, we have the AUC value, FPR value, optimal λ value, etc for each experiment.

2.3.1 Low dimensional, large sample size

In this case we choose n=1000 and p=10. The average optimal tuning parameters for three different approaches under 50 repetitive experiments are 0.0883, 0.0641 and 0.0995 respectively. According

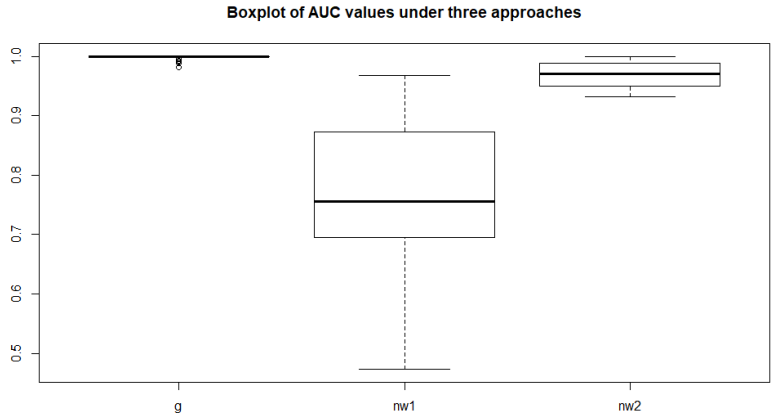

Figure 5: AUC when n=1000, p=10

to Figure 5, graphical lasso outperforms the other two approaches in terms of AUC value, with nearly all experiments' AUC values equalling 1, meaning that it can recover almost all edges in the graph, whilst node-wise 2 still has an overall good performance, with AUC values generally close to 1. However, AUC values under node-wise 1 are rather dispersed. In addition, the average optimal tuning parameters for three methods are very small. We assume that the constraints of node-wise 1 is relatively strict under circumstances when dimensionality is far smaller than sample size.

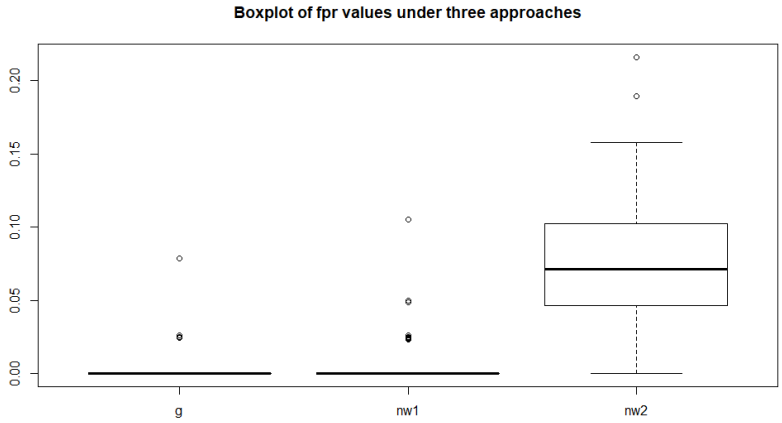

Figure 6: FPR when n=1000, p=10

Figure 6 shows the False Positive Rate (FPR) under three methods. FPR examines Type-I er-

ror. Node-wise 2 has relatively higher FPR than the other two approaches, and we believe the reason is, intuitively, this method estimates node i and j to be connected if either coefficient is non-zero according to lasso regression, and this criterion tends to classify an edge to exist. Therefore, node-wise 2 has higher Type-I error in this scenario.

2.3.2 High dimensional, small sample size

In this case we choose n=50 and p=100. The average optimal tuning parameters for three different approaches under 50 repetitive experiments are 0.5867, 0.3887 and 0.6572 respectively. In this

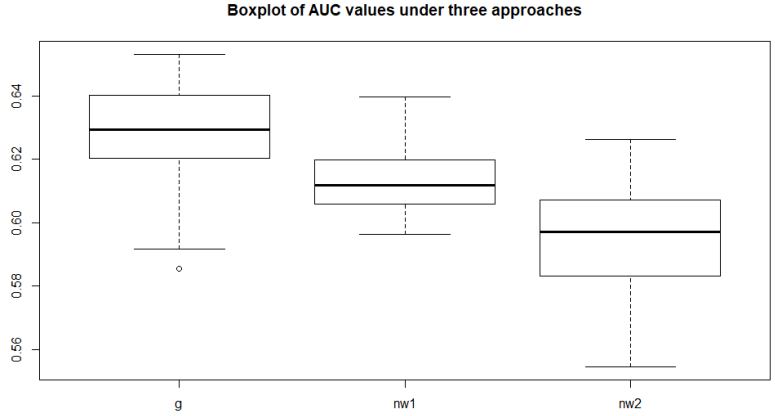

Figure 7: AUC when n=50, p=100

scenario(Figure 7), graphical lasso outperforms the other two methods in terms of AUC values, with a median AUC value of 0.63. Node-wise 2 gives the worst performance. Besides, the average optimal λ for node-wise 2 is the largest. Intuitively, since dimensionality is large, penalty term tends to be larger in order to get a sparse result with relatively small sample size. We assume that the constraints of node-wise 2 is relatively loose under circumstances when dimensionality is far larger than sample size.

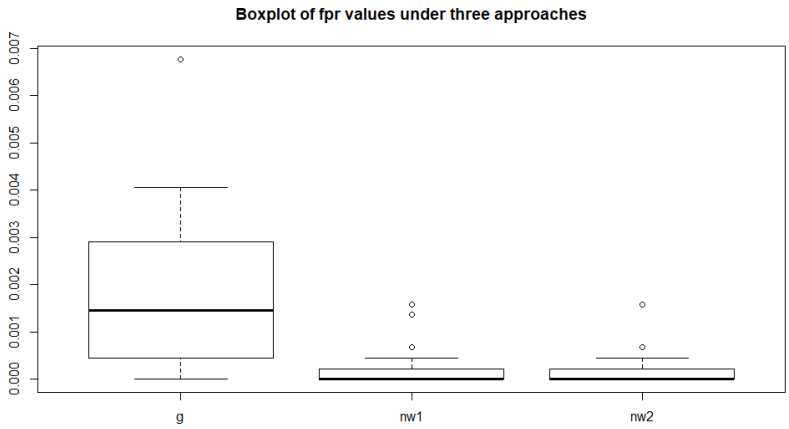

Figure 8: FPR when n=50, p=100

In this case(Figure 8), graphical lasso has relatively higher FPR than the other two approaches.

2.3.3 Compatible dimensionality and sample size

In this case we choose n=p=50. The average optimal tuning parameters for three different approaches under 50 repetitive experiments are 0.5625, 0.3579 and 0.5780 respectively. In this sce-

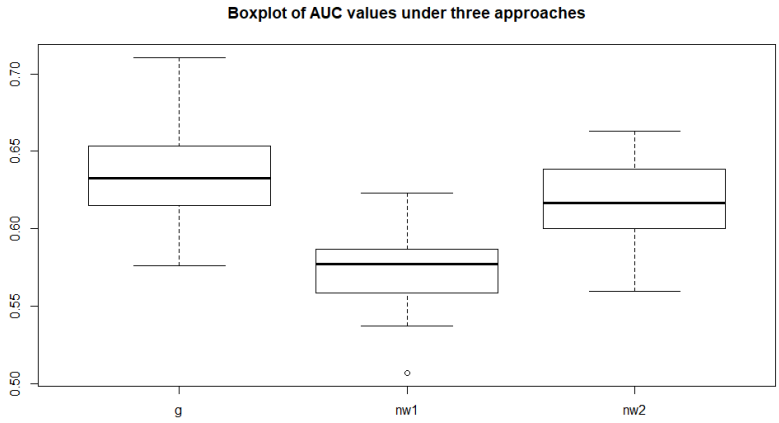

Figure 9: AUC when n=50, p=50

nario(Figure 9), graphical lasso still has the best performance regarding AUC value and node-wise 1 has relatively poor results. The optimal tuning parameters are much larger than that of Section 2.3.1, but similar to that of Section 2.3.2. Thus, we assume that the choice of tuning parameter is more sensitive to dimensionality instead of sample size.

Similar to Section 1.3.2, graphical lasso has relatively higher FPR than the other two approaches

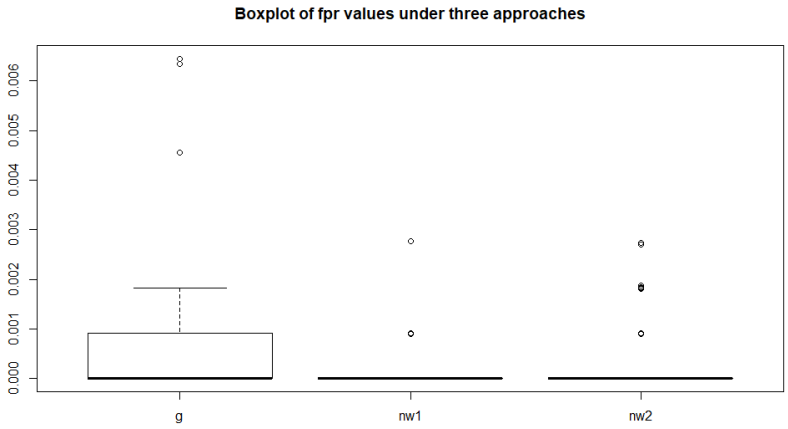

Figure 10: FPR when n=50, p=50

in this scenario(Figure 10).

2.4 Summary

This section summarises our findings from three perspectives. First we will talk about the choices of optimal tuning parameters, then we compare their computation time focusing on the efficiency of different algorithms, and finally we present our finding regarding the asymmetry problem with graphical lasso method.

2.4.1 Optimal tuning parameter

Replicating the data generation and analysis process for 50 times under different scenarios, the best tuning parameter is obtained by averaging the estimated best parameter in each loop. The results are shown in table above.

Table 2: Comparison of tuning parameter

	n<<p	n>>p	n=p
Graphical Lasso	0.5867	0.0883	0.5625
Node-wise 1	0.3887	0.0641	0.3579
Node-wise 2	0.6572	0.0995	0.5780

2.4.2 Running time comparison

This section delivers time comparison among graphical lasso, node-wise lasso 1 and 2. The table above shows the average time spend of each method under different scenarios(unit: milliseconds). For the same sample data, graphical lasso is more efficient in all scenarios than the other two

approaches. For n¿¿p and n=p case, time spend of node-wise lasso is about 10 times higher than graphical lasso, and 3 times higher for n¡¡p.

Table 3: Comparison of computation time

	n<<p	n>>p	n=p
Graphical Lasso	35.88	1.10	4.33
Node-wise 1	99.54	11.62	48.50
Node-wise 2	99.54	11.78	48.37

2.4.3 The lack of symmetry in graphical lasso

We notice that the estimated inverse covariance matrix provided by 'glasso' is not always symmetric. We try for a graphical lasso with n=30, p=40 and $\rho = 10^-5$. To quantify the lack of symmetric, consider the Manhattan distance between the upper triangle and lower triangle of the estimated inverse covariance matrix, the result is 253773, which implies that asymmetry of inverse covariance matrix does exist.

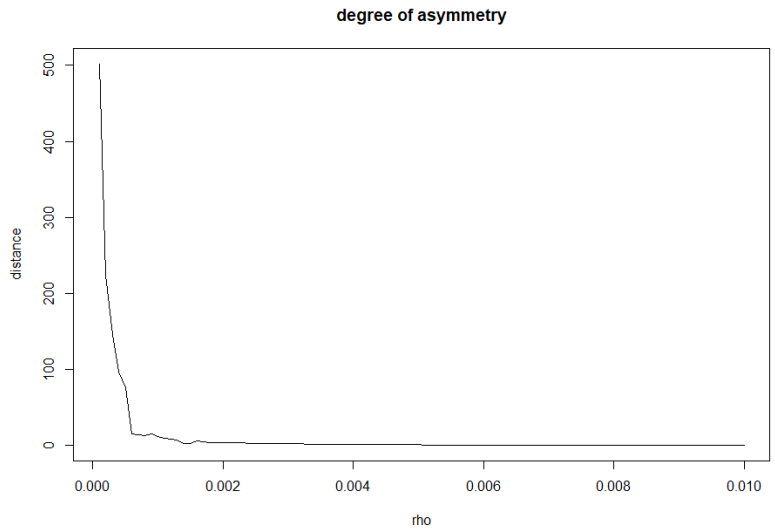

Figure 11: degree of asymmetry when n=30, p=40

As is mentioned by Rolfs and Rajaratnam(2013)[1], convergence of covariance matrix to a specified tolerance does not necessarily imply convergence of inverse covariance matrix to a specified tolerance, since 'glasso' package does not calculate the inverse covariance matrix directly from the estimated covariance matrix. This problem exacerbates when ρ is small or when n$\ll p$.

[1]Rolfs, Benjamin Rajaratnam, Bala. (2011). A note on the lack of symmetry in the graphical lasso.

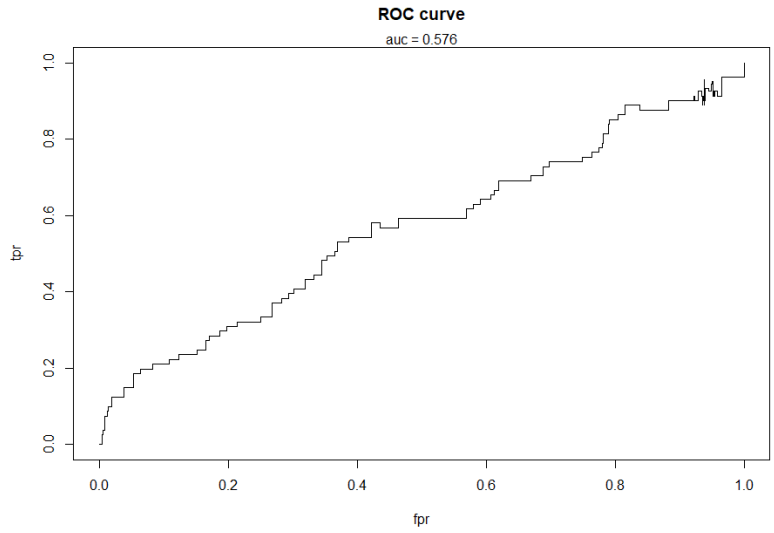

Figure 12: ROC curve when n=30, p=40

Compare the degree of asymmetry by different ρ under scenario n=30, p=40. The degree of asymmetry is extremely high for small $\rho(\rho < 10^{-}3)$. Thus, when estimating graphical lasso with small ρ, it is better to inverse the estimated covariance function manually rather than use the estimated inverse covariance function directly. The asymmetry is also the reason for the wired ROC curve of graphical lasso model, under small ρ, the estimate becomes not reliable enough.

3 Appendix

3.1 Bike Sharing in London

3.1.1 Data Resource

The data are collected from 3 sources from 1/1/2015 to 31/12/2016 published on "Kaggle" - https://www.kaggle.com/hmavrodiev/london-bike-sharing-dataset.

https://cycling.data.tfl.gov.uk/ 'Contains OS data © Crown copyright and database rights 2016' and Geomni UK Map data © and database rights [2019] 'Powered by TfL Open Data' - Bike sharing usage dataset

https://freemeteo.com - weather data

https://www.gov.uk/bank-holidays

3.1.2 R code

```
library(tidyverse)
library(lubridate)
library(stats)
library(glmnet)
library(FNN)
library(MASS)
library(caret)
library(dplyr)
library(tree)
library(randomForest)
library(gbm)

bike_raw <- as.tibble(read.csv('london_merged.csv'))

bike <- bike_raw %>%
  mutate(weather_code = as.factor(weather_code)) %>%
  mutate(is_holiday = as.factor(is_holiday)) %>%
  mutate(is_weekend = as.factor(is_weekend)) %>%
  mutate(timestamp = as.POSIXct(timestamp, format =  "%Y-%m-%d␣%H:%M:%OS")) %>%
  mutate(date = as.Date(timestamp)) %>%
  mutate(time = as.factor(hour(timestamp))) %>%
  mutate(day = day(date)) %>%
  mutate(month = month(date)) %>%
  mutate(year = year(date)) %>%
  arrange(t1, t2) %>%
  filter(is.na(month) == F) %>%
  mutate(yday = yday(date)) %>%
  mutate(sin = sin(2*pi*yday/(365))) %>%
  mutate(cos = cos(2*pi*yday/(365))) %>%
```

```
  dplyr::select(-c(timestamp,yday,year,month,day,date))
bike
bike_y <- bike$cnt
## devide testing set and training set
set.seed(443)
index <- seq(1, 17413)
test_index <- sample(index, 3483)
train_index <- index[-test_index]
bike_y_test = bike_y[test_index]
bike_y = bike_y[-test_index]

## linear regression
summary(reg_bike <- lm(cnt~., data = bike[train_index,]))
yhat_lm <- predict(reg_bike, bike[test_index,])
summary((yhat_lm - bike$cnt[test_index])^2) #MSE 315967
par(mfrow=c(2,2))
plot(reg_bike)

## Ridge regression
bike_x <- model.matrix(cnt~., bike[train_index,])[,-1] #remove B0
fit.ridge <-glmnet(bike_x,bike_y , alpha=0)
plot(fit.ridge, xvar="lambda", label= TRUE)
plot(fit.ridge, xvar="dev", label= TRUE)
cv.ridge <-cv.glmnet(bike_x,bike_y, alpha=0)
plot(cv.ridge)
coef(cv.ridge)
coef(glmnet(bike_x,bike_y ,alpha=0, lambda=cv.ridge$lambda.min))
opt_lambda <- cv.ridge$lambda.min
opt_lambda
fit <- cv.ridge$glmnet.fit
summary(fit)
bike_x_test <- model.matrix(cnt~., bike[test_index,])[,-1]
yhat_ridge <- predict(cv.ridge, bike_x_test,type='response')
summary((yhat_ridge - bike_y_test)^2)



## lasso, first do the variables selection
bike_x <- model.matrix(cnt~., bike[train_index,])[,-1] #remove B0
bike_x
lasso_cv <- cv.glmnet(bike_x, bike_y,family=c('poisson'))
lasso_cv
coef(lasso_bike <- glmnet(bike_x, bike_y, lambda = lasso_cv$lambda.1se,family=c('poisson')))
bike_x_test <- model.matrix(cnt~., bike[test_index,])[,-1]
yhat_lasso <- predict(lasso_bike,bike_x_test,type='response')
summary((yhat_lasso - bike$cnt[test_index])^2)
coef(lasso_cv)
coef(glmnet(bike_x,bike_y, lambda=lasso_cv$lambda.min))

## Validation set approach to select best lambda in Lasso
```

```
set.seed(1)
train <-sample(nrow(bike_x), 3483, replace=FALSE)
lasso.train <-glmnet(bike_x[train,], bike_y[train])
pred.test <-predict(lasso.train, bike_x[-train,])
rmse <-sqrt(apply((bike_y[-train]-pred.test)^2,2,mean))
plot(log(lasso.train$lambda), rmse, type="b", xlab="Log(lambda)")
lambda.best <-lasso.train$lambda[order(rmse)[1]]
lambda.best

# poisson regression
count.fits <- function(model,newdata,y=NULL){
  fit.mu <- newdata
  fit.mu$muhat <- predict(model,newdata,type="response")
  if(!is.null(y)){
    ind.tmp <- seq(ncol(fit.mu)+1,ncol(fit.mu)+length(y))
    fit.mu <- cbind(fit.mu,matrix(NA,nrow(fit.mu),length(y)))
    for(j in seq(nrow(fit.mu))){
      if(class(model)[1]=="glm")
        fit.mu[j,ind.tmp] <- dpois(y,lambda=fit.mu$muhat[j])
      if(class(model)[1]=="negbin"){
        fit.mu[j,ind.tmp] <- dnbinom(y,size=model$theta,mu=fit.mu$muhat[j])
      }
    }
    colnames(fit.mu)[ind.tmp] <- paste("P(Y=",y,")",sep="")
  }
  fit.mu
}
summary(pois_bike <- glm(cnt~., data=bike[train_index,],family='poisson'))
yhat <- count.fits(pois_bike,bike[test_index,])
summary((yhat$muhat - bike$cnt[test_index])^2)
plot(pois_bike)

## Negative binomial regression
neg_bike <- glm.nb(cnt~., data=bike[train_index,])
summary(neg_bike)
Yhat_neg = predict(neg_bike, newdata=bike[test_index,],type="response")
mean((bike_y_test-Yhat_neg)^2)

## knn regression
# KNN Plot model accuracy vs different values of k=5
set.seed(123)
model <- train(
  cnt ~., data = bike[train_index,], method = "knn",
  trControl = trainControl("cv", number = 10),
  preProcess = c("center","scale"),
  tuneLength = 20
)
plot(model)
model$bestTune
```

```
yhat_knn <- knn.reg(bike_x, bike_x_test, bike_y, k = 5)
summary((yhat_knn$pred - bike$cnt[test_index])^2) #high mse



#######################
bike_month <- bike %>%
  group_by(month) %>%
  mutate(monthuse = mean(cnt)) %>%
  select(month, monthuse) %>%
  unique()
plot(bike_month$month, bike_month$monthuse)

bike_hour <- bike %>%
  group_by(time,is_holiday) %>%
  mutate(hourave = mean(cnt)) %>%
  select(hourave, time,is_holiday) %>%
  group_by() %>%
  mutate(time=as.numeric(time)) %>%
  unique()
plot(bike_hour$time, bike_hour$hourave, type = 'p')

bike_day <- bike %>%
  group_by(day) %>%
  mutate(dayavg = mean(cnt)) %>%
  select(day, dayavg)
plot(bike_day$day, bike_day$dayavg)
ggplot(bike_hour,aes(time,hourave,color=is_holiday))+
  geom_line()

##Regressiontree
attach(bike)
tree.bike<-tree(cnt~. , data=bike, subset=train_index)
summary(tree.bike)
plot(tree.bike)
text(tree.bike, pretty=0)

cv.bike <- cv.tree(tree.bike)
plot(cv.bike$size, cv.bike$dev, type = "b")

#best=6

#Prune
prune.bike <- prune.tree(tree.bike, best = 5)
plot(prune.bike)
text(prune.bike, pretty = 0)

yhat <- predict(tree.bike, newdata = bike[-train_index,])
bike.test <- bike[-train_index, "cnt"]
plot(yhat, bike[-train_index,]$cnt)
```

```
abline(0, 1)

mean((yhat - bike[-train_index,]$cnt) ^ 2)



############
# Bagging #
############
bag.bike<-randomForest(cnt~. , data= bike, subset=train_index, mtry=4, importance=TRUE)
bag.bike

yhat.bag <-predict(bag.bike, newdata = bike[-train_index,])
plot(yhat.bag, bike[-train_index,]$cnt)
abline(0,1)

mean((yhat.bag-bike[-train_index,]$cnt) ^2 )

importance(bag.bike)
varImpPlot(bag.bike)

# time and is_weekend is most important

############
# Boosting #
############

boost.bike <- gbm( cnt ~ ., data = bike[train_index,], distribution = "gaussian", n.trees =

summary(boost.bike)
par(mfrow = c(1, 2))
plot(boost.bike, i="time")
plot(boost.bike, i='is_weekend')

yhat.boost = predict(boost.bike, newdata = bike[-train_index, ],
                     n.trees = 5000)
mean((yhat.boost - bike[-train_index,]$cnt) ^ 2)
```

3.2 Estimation of graphical models using lasso-related approaches

```
library(MASS)
library(glmnet)
library(tidyverse)
# define a function to generate covariance matrix with dimension p
simu <- function(p,delta,seed=20191118){
  set.seed(seed)
  # create p^2 uniform random numbers
  rand <- runif(p*p)
  # with prob 0.1 to be 0.5, prob 0.9 to be 0
```

```
  rand <- ifelse(rand >= .1,0,.5)
  # create p*p matrix
  b <- matrix(rand,p,p)
  # transform b to be symmetric
  b[lower.tri(b)] <- t(b)[lower.tri(b)]
  # set diag to be delta
  diag(b) <- delta
  return(b)
}
# generate n random samples
simu2 <- function(n,p,delta,seed=20191118){
  a <- simu(p,delta,seed=seed)
  # standardise b to have unit diagonals
  theta <- cov2cor(a)
  # solve for the covariance matrix for Gaussian distribution
  sigma <- solve(theta)
  # simulate from multivariate normal distribution
  data <- mvrnorm(n,rep(0,p),sigma)
  return(data)
}

# node-wise lasso
# calc estimated edge matrix
node_wise<-function(n,p,delta,lambda=1,seed=20191118){
  dat <- simu2(n,p,delta,seed=seed)
  coef_<-rep(0,p)
  for (i in 1:p) {
    # the value of lambda to be continued...
    coefficients<-coef(glmnet(dat[,-i],dat[,i], lambda=lambda,intercept = F))
    coefficients<-as.vector(coefficients)
    swap<-coefficients[i]
    coefficients[i]<-coefficients[1]
    coefficients[1]<-swap
    coef_<-rbind(coef_,coefficients)
  }
  coef_ <- coef_[-1,]
  diag(coef_) <- 1
  return(coef_)
}

# a more efficient way to transform edge
edge<-function(n,p,delta,type,lambda=1,seed=20191118){
  node<-node_wise(n,p,delta,lambda=lambda,seed=seed)
  upper<-node[upper.tri(node)]
  lower<-node[lower.tri(node)]
  node<-matrix(0,p,p)
  if(type=='1'){
    # node-wise lasso 1: both are non-zero
    upper[(upper!=0)&(lower!=0)]<-1
    node[upper.tri(node, diag=FALSE)] <- upper
```

```
    node[lower.tri(node)] <- t(node)[lower.tri(node)]
  }
  if(type=='2'){
    # node-wise lasso 2: either is non-zero
    upper[(upper!=0)|(lower!=0)]<-1
    node[upper.tri(node, diag=FALSE)] <- upper
    node[lower.tri(node)] <- t(node)[lower.tri(node)]
  }
  diag(node)<-1
  return(node)
}

library(glasso)
# calculate edge of graphic lasso model
graphic <- function(n,p,delta,rho,seed=20191118){
  dat <- simu2(n,p,delta,seed=seed)
  cov <- cov(dat)
  glasso <- glasso(cov,rho)
  wi <- glasso$wi
  wi[!wi==0] <- 1
  return(wi)
}

# compute TPR and FPR
# input for data and reference is matrix
tfpr <- function(data, reference){
  dim <- ncol(data)
  data[!data==0] <- 1
  reference[!reference==0] <- 1
  # flatten the matrix to vector
  data_flat <- as.vector(data)
  ref_flat <- as.vector(reference)
  # extract the diagnal elements
  diag_index <- (dim)*seq(0,dim-1)+seq(1,dim)
  data_flat <- data_flat[-diag_index]
  ref_flat <- ref_flat[-diag_index]
  # calculate TPR, TPR=TP/P
  tpr <- length(ref_flat[(data_flat==1)&(ref_flat==1)])/length(ref_flat[ref_flat==1])
  # calculate FPR, FPR=FP/N
  fpr <- length(ref_flat[(data_flat==1)&(ref_flat==0)])/length(ref_flat[ref_flat==0])
  # calculate overall prediction accuracy
  overall <- length(ref_flat[data_flat==ref_flat])/length(ref_flat)
  # calculate precision
  ppv <- length(ref_flat[(data_flat==1)&(ref_flat==1)])/length(data_flat[data_flat==1])
  # calculate F1 score
  f1 <- 2*ppv*tpr/(ppv+tpr)
  out <- c(tpr=tpr,fpr=fpr,overall=overall,ppv=ppv,f1=f1)
  return(out)
}
```

```
# calc auc
calc_auc <- function(data){
  data <- data[order(data[,2]),]
  auc <- 0
  for(i in 2:nrow(data)){
    auc = auc + (data[i,2]-data[i-1,2]) * data[i,1]
  }
  return(auc)
}

# plot roc curve
# the range of lambda should be reconsidered
roc <- function(n,p,delta,type,shrink,k=100,seed=20191118,plot=T){
  roc_curve <- matrix(NA, k+2, 6)
  ref <- simu(p,delta,seed=seed)
  for (i in seq(1:k)){
    lambda <- i^3/shrink
    if(type=='g')
      res <- graphic(n,p,delta,rho = lambda,seed=seed)
    else
      res <- edge(n,p,delta,lambda = lambda,type=type,seed=seed)
    roc_curve[i,1:5] <- tfpr(res,ref)
    roc_curve[i,6] <- lambda
  }
  roc_curve[k+1,] <- c(1,1,NA,NA,NA,0)
  roc_curve[k+2,] <- c(0,0,NA,NA,NA,Inf)
  roc_curve <- as.data.frame(roc_curve) %>%
    arrange(V2,V1)
  colnames(roc_curve) <- c('tpr','fpr','overall','ppv','f1','lambda')
  auc <- calc_auc(roc_curve[,1:2])
  if (plot == T){
    plot(roc_curve[,2],roc_curve[,1],type='s',xlab = 'fpr',ylab = 'tpr',
    main = 'ROC␣curve',xlim = c(0,1),ylim = c(0,1))
    mtext(paste('auc␣=', round(auc,3)),3)
  }
  roc_curve <- roc_curve %>%
    arrange(lambda)
    return(list(value=roc_curve,auc=auc))
}

# find the best tuning parameter for each model
# select tuning parameter based on accuracy rate and F1 score
tunning <- function(list){
  value <- list$value
  value <- value[-1,]
  value <- value[-nrow(value),]
  lambda_overall <- value[which.max(value$overall),]
  lambda_f1 <- value[which.max(value$f1),]
  return(list(overall=lambda_overall,f1=lambda_f1))
}
```

```
# replicate for 50 times
rep_50 <- function(n,p,delta,type,shrink,k=100){
  overall <- data.frame(tpr=NA,fpr=NA,overall=NA,ppv=NA,f1=NA,lambda=NA)
  f1 <- data.frame(tpr=NA,fpr=NA,overall=NA,ppv=NA,f1=NA,lambda=NA)
  auc <- vector()
  for (i in seq(1:50)){
    original<-simu(p,delta,seed=20191117+i)
    value <- roc(n,p,delta,type,shrink,k=k,plot=F,seed=20191117+i)
    lambda <- tunning(value)
    overall[i,] <- lambda$overall
    f1[i,] <- lambda$f1
    auc[i] <- value$auc
  }
  return(list(overall=overall,f1=f1,auc=auc))
}

# try n=1000,p=10
rep11 <- rep_50(1000,10,4,type='g',10000)
rep12 <- rep_50(1000,10,4,type='1',10000)
rep13 <- rep_50(1000,10,4,type='2',10000)
# save(rep11,rep12,rep13,file='rep1.RData')
load('rep1.RData')

auc1=data.frame(g=rep11$auc,nw1=rep12$auc,nw2=rep13$auc)
boxplot(auc1,main='Boxplot␣of␣AUC␣values␣under␣three␣approaches')

fpr1=data.frame(g=rep11$overall['fpr'],nw1=rep12$overall['fpr'],nw2=rep13$overall['fpr'])
colnames(fpr1)<-c('g','nw1','nw2')
boxplot(fpr1,main='Boxplot␣of␣fpr␣values␣under␣three␣approaches')

lambda1<-data.frame(g=rep11$overall['lambda'],nw1=rep12$overall['lambda'],
                    nw2=rep13$overall['lambda'])
summary(lambda1)

# try n=p=50
rep21 <- rep_50(50,50,4,type = 'g',10000)
rep22 <- rep_50(50,50,4,type = '1',10000)
rep23 <- rep_50(50,50,4,type = '2',10000)
# save(rep21,rep22,rep23,file='rep2.RData')
load('rep2.RData')

auc2=data.frame(g=rep21$auc,nw1=rep22$auc,nw2=rep23$auc)
boxplot(auc2,main='Boxplot␣of␣AUC␣values␣under␣three␣approaches')

fpr2=data.frame(g=rep21$overall['fpr'],nw1=rep22$overall['fpr'],nw2=rep23$overall['fpr'])
colnames(fpr2)<-c('g','nw1','nw2')
boxplot(fpr2,main='Boxplot␣of␣fpr␣values␣under␣three␣approaches')

lambda2<-data.frame(g=rep21$overall['lambda'],nw1=rep22$overall['lambda'],
```

```
                    nw2=rep23$overall['lambda'])
summary(lambda2)


# try n=30, p=40
#rep31 <- rep_50(30,40,4,type = 'g',4000000,k=1500)
#rep32 <- rep_50(30,40,4,type = '1',100000,k=300)
#rep33 <- rep_50(30,40,4,type = '2',100000,k=300)
# save(rep31,rep32,rep33,file='rep3.RData')
# load('rep3.RData')

#try n=50, p=100
rep31 <- rep_50(50,100,4,type = 'g',4000000,k=1500)
rep32 <- rep_50(50,100,4,type = '1',100000,k=300)
rep33 <- rep_50(50,100,4,type = '2',100000,k=300)
save(rep31,rep32,rep33,file='rep31.RData')
load('rep31.RData')

auc3=data.frame(g=rep31$auc,nw1=rep32$auc,nw2=rep33$auc)
boxplot(auc3,main='Boxplot of AUC values under three approaches')

fpr3=data.frame(g=rep31$overall['fpr'],nw1=rep32$overall['fpr'],nw2=rep33$overall['fpr'])
colnames(fpr3)<-c('g','nw1','nw2')
boxplot(fpr3,main='Boxplot of fpr values under three approaches')

lambda3<-data.frame(g=rep31$overall['lambda'],nw1=rep32$overall['lambda'],
                    nw2=rep33$overall['lambda'])
summary(lambda3)


# compare the time spend between models
library(microbenchmark)
# for n=1000, p=10
microbenchmark(edge(1000,10,4,type='1'))
microbenchmark(edge(1000,10,4,type='2'))
microbenchmark(graphic(1000,10,4,0.1))
# for n=50,p=50
microbenchmark(edge(50,50,4,type='1'))
microbenchmark(edge(50,50,4,type='2'))
microbenchmark(graphic(50,50,4,0.1))
# for n=30,p=40
microbenchmark(edge(30,40,4,type='1'))
microbenchmark(edge(30,40,4,type='2'))
microbenchmark(graphic(30,40,4,0.1))
# for n=50,p=100
microbenchmark(edge(50,100,4,type='1'))
microbenchmark(edge(50,100,4,type='2'))
microbenchmark(graphic(50,100,4,0.1))
# easily see that graphical lasso is far more efficient
```