bergama 4

December 8, 2022

[ ]:
```
require(Metrics)
```

1 An Analysis on Linear/Nonlinear Spatial, Temporal and Spatio-Temporal Relationships in "Wind Speed" and "Electricity Production" Data

This Project aims to analyse and compare Linear/Nonlinear Spatial, Temporal and Spatio-Temporal models using windspeed data at four different locations to make forecasts about electricity production from a nearby wind turbine.

2 Data

The data we will use contains hourly wind speed data of 4 years from four different locations and electricity hourly production data from one terminal.

Training split of the data contains the first three years starting from 1st of July, 2017.

Test split contains one year starting from 1st of July, 2020.

[6]:
```
train <- readRDS('data4train.Rda')
test <- readRDS('data4test.Rda')
colnames(train)[3:6] <- c('ne','nw','se','sw')
colnames(test)[3:6]  <- c('ne','nw','se','sw')

train[train$production==130,'production'] = 120
```

[7]:
```
dim(train)
```

1. 26304 2. 7

The data has 7 columns: date, hour, 4 locations and production.

Date column has <date> format

Hour column has integer format ranging from 0 to 23.

Locations have numeric format ranging from 0 to ~30.7

Production has numeric format ranging from 0 to 120

Below you can see a piece from the training data:

[3]:
```
head(train)
```

A data.table: 6 × 7

	date <date>	hour <int>	ne <dbl>	nw <dbl>	se <dbl>	sw <dbl>	production <dbl>
1	2017-07-01	0	6.540107	7.741893	4.760000	6.249544	57
2	2017-07-01	1	7.166667	8.727033	5.451166	7.302665	77
3	2017-07-01	2	7.793246	9.728310	6.148695	8.373715	55
4	2017-07-01	3	8.419840	10.741210	6.850642	9.456606	35
5	2017-07-01	4	8.744693	11.316219	7.012487	9.742422	35
6	2017-07-01	5	9.090841	11.925593	7.209476	10.052203	61

[6]:
```
tail(test)
```

A data.table: 6 × 7

date <date>	hour <int>	ne <dbl>	nw <dbl>	se <dbl>	sw <dbl>	production <dbl>
2021-06-30	18	8.064914	3.965807	6.657385	5.484300	28
2021-06-30	19	7.487073	4.659016	5.750321	4.226888	28
2021-06-30	20	7.225808	6.377307	5.703367	5.237055	30
2021-06-30	21	6.713198	5.863464	5.063222	6.149658	49
2021-06-30	22	6.308642	5.227768	4.900224	6.801429	60
2021-06-30	23	6.224262	5.649105	4.429714	6.606498	78

[7]:
```
dim(train) + dim(test)
```

1. 35064 2. 14

3 Plan

Our first phase will have 3 main parts: * Spatial Part * Spatial relationships are inspected on the raw data (data with 4 wind speed features) 1. We first apply a General Linear Model (GLM) to the raw data and set a basis performance level. 2. Then we do a Principal Component Analysis (PCA) on the raw data and choose the Principal Components (PC) explaining at least 90% of the variance. 3. Finally we do a Kernel PCA (with polynomial kernel of degree 3) on the raw data and choose the PCs explaining at least 90% of the variance. In this step we could see the nonlinear relationships on the simplest format. * Note that we split train data for Kernel PCA as the dimension of Kernel Matrix equals to the number of instances(rows). Even 16GB Ram is not enough for using the full train data. * Spatio-Temporal Part * For analysing spatio-temporal relationships, we add Lagged/Leaded versions of the existing features as new features to the raw data. Applying the aforementioned methods to this data, we will be able to add spatio-temporal features to our problem. 1. We first apply a GLM to the "lagged data". 2. Then we do a PCA on the lagged data and choose the PCs explaining at least 90% of the variance. 3. Finally we do a Kernel PCA (with polynomial kernel of degree 3) on the raw data and choose the PCs explaining at least 90% of the variance. * Kronecker Spatio-Temporal Part * This part will bring another approach for spatio-temporal relations. After analysing spatial and temporal relations seperately, with "Kronecker Product" we will able to get spatio-temporal principal components of the data. * Creating the "Spatial Covariance Matrix" is no different than doing PCA on raw data, however for creating the "Temporal Covariance Matrix" we will need to transform our data to another format having Lag/Lead values and no location indicators. * There sure is a difference between second and

third part. Our aim in this part is to get some findings for differences between "PCA on Lagged Data" and "Kronecker PCA".

4 Preperations

We will need to install the libraries, prepare some functions and tables that we will use before starting our first phase.

4.0.1 Libraries Needed:

[310]:
```
require(dplyr, quietly = TRUE)
require(data.table, quietly = TRUE)
require(ggplot2, quietly = TRUE)
require(glmnet, quietly = TRUE)
require(plot.matrix, quietly = TRUE)
require(kernlab, quietly = TRUE)
require(tidyr, quietly = TRUE)
require(tidyverse, quietly = TRUE)
```

4.0.2 Naming Format:

train.x.y.z for train data

test.x.y.z for test data

fit.x.y.z for model/pca results

x, y, z will be used for specifying the data format (e.g. train.lag for lagged data, fit.lag.pca for PCA model done on lagged data, etc.)

4.0.3 Tables For Storing Model Performances

results.main will have every instance of the data for every model

results.summary will only contain model name, performance result and some notes

[590]:
```
# data.tables for results

results.main = data.table(date = Sys.Date(),
                          year = integer(1),
                          month = integer(1),
                          hour = integer(1),
                          model = character(1),
                          actual = numeric(1),
                          predicted = numeric(1),
                          wmape = numeric(1))
results.main = results.main[-1,]

results.summary = data.table(model = character(),
                             wmape = numeric(),
```

```
                                notes = character())
```

4.1 Functions

[290]:
```
# Scale features of wind speed on train then apply to test (Resulting␣
 ↪windspeeds will be between 0 and 1)
scaleWind2 <- function(train,
                       test,
                       other.head = c(1,2),
                       other.tail = 7){
    # removing others
    other = c(other.head,other.tail)
    d1 = as.data.frame(train[,-..other])
    d2 = as.data.frame(test[,-..other])
    for(i in c(1:ncol(d1))){
        m = mean(d1[,i])
        s = sd(d1[,i])
        d1[,i] = (d1[,i] - m) / s
        d2[,i] = (d2[,i] - m) / s
    }
    # merge others back
    d1 = cbind(train[,..other.head],
               d1,
               train[,..other.tail])
    d2 = cbind(test[,..other.head],
               d2,
               test[,..other.tail])

    res = list(d1,d2)
    return(res)
}
```

[292]:
```
# A function for calculating Weighted Mean Absolute Percentage Error
wmape <- function(observed,predicted){
    difs = abs(observed-predicted)
    a = sum(difs)
    b = sum(observed)
    res = a/b
    invisible(res)
}
```

[293]:
```
# function for adding new features that are powers of existing features

# powers = c(integers>1)
# others are list of numbers that specify the col number of date/time/
 ↪production or other features that we dont take power of
createPowered <- function(data,
```

```
                          powers,
                          other.head = c(1,2),
                          other.tail = 7){
    # remove others
    other = c(other.head,other.tail)
    dat = data[,-..other]
    # save it for binding at last
    res = dat
    # create powered features
    for(i in powers){
        tem.dat = dat
        for(p in c(2:i)){
            tem.dat = tem.dat*dat
        }
        colnames(tem.dat) = paste(colnames(tem.dat),paste('p',i,sep=''),sep='_')
        res = cbind(res,tem.dat)
    }
    res = cbind(data[,..other.head],res,data[,..other.tail])
    return(res)
}
```

[1]:
```
# function for returning a lagged data. lags should be a list of numbers.

# highest and lowest value in lags are important for cutting the edges
# others : same as above
createLagged <- function(data,
                          lags,
                          other.head = c(1,2),
                          other.tail = 7) {
    # remove 0 if it exists
    # get min/max for removing first/last rows for lag/lead
    minlag = min(0,lags)
    maxlag = max(0,lags)
    startN = 1 - minlag
    endN = nrow(data) - maxlag
    # seperating others
    other = c(other.head, other.tail)
    tobe.head = data[(startN:endN),..other.head]
    tobe.tail = data[(startN:endN),..other.tail]
    dat = data.table() # empty data table
    # adding lagged values one by one (column by column)
    for(i in lags){
        lagdat = data[((startN+i):(endN+i)),-..other]
        if(i<0){
            lagname = paste('lag',-i,sep='')
            colnames(lagdat) = paste(colnames(lagdat),lagname,sep='_')
        } else if(i>0){
```

```
            lagname = paste('lead',i,sep='')
            colnames(lagdat) = paste(colnames(lagdat),lagname,sep='_')
        }
        dat = cbind(dat,lagdat)
    }
    dat = cbind(tobe.head,dat,tobe.tail)
    return(dat)
}
```

[295]:
```
# prepares for Temporal Covariance Methods.

# Takes lagged data(full) and removes location(feature) names and groups them␣
 ↪by only lag/lead
# feature.names : list of feature(location) names
# lags : list of integers representing lags
# finally adds the corresponding feature's name and month(of normal) to the␣
 ↪resulting data.table (for future easiness)

# NOTE: Dcast tarzi bi sey de kullanilabilirdi aslinda burada. baya hizli bisi␣
 ↪olabilir ama onlari da iyi ogrenmek lazim
laggedToTemporal <- function(data.lag,
                             feature.names,
                             lags){
    # getting month's from data.lag
    date.text <- data.lag$date
    data.lag.months = data.table(as.numeric(format(date.text,format = '%m')))
    # first lets create new column names:
    newcolnames = list()
    for(l in lags){
        if(l<0){
            x = paste('lag',-l,sep='')
        }else if(l>0){
            x = paste('lead',l,sep='')
        }else{
            x = 'normal'
        }
        newcolnames = c(newcolnames, x)
    }
    newcolnames = c(newcolnames,'month','feature')
    # create the final data.table with new column names
    res = data.table(matrix(0, nrow = 1, ncol = length(lags)+2))
    colnames(res) = as.character(newcolnames)
    res = res[-1,]
    # now split data for every feature.name
    m = names(data.lag) # lagged datanames for filtering by featurenames
    for(k in feature.names){
        data.lag.k <- as.data.frame(data.lag)[,grepl(k, m)]
```

```
        data.lag.k <- cbind(data.lag.k,data.lag.months)
        long.list.of.k <- data.table(matrix(k,nrow=nrow(data.lag.k),ncol=1))
        data.lag.k <- cbind(data.lag.k,long.list.of.k)
        colnames(data.lag.k) <- newcolnames
        res = rbind(res, data.lag.k)
    }
    return(res)
}
```

[296]:
```
# function for fitting a GLM and predicting on the test set (and printing␣
↪results)

# traindata, trainres, testdata, testres, perf(type.measure)
# for perf prefer mae
customGlm <- function(traindata,
                      trainres,
                      testdata,
                      testres,
                      perf='mae',
                      output = FALSE,
                      seed=42) {
    set.seed(seed)
    fit1 = cv.glmnet(as.matrix(traindata),
                     as.matrix(trainres),
                     type.measure = perf)
    # predicting
    pred.min = predict(fit1,
                       as.matrix(testdata),
                       s = "lambda.min")
    pred.1se = predict(fit1,
                       as.matrix(testdata),
                       s = "lambda.1se")
    if(output){
        # RMSE results
        print(paste('WMAPE on test set (min):', wmape(as.matrix(testres), pred.
↪min)))
        print(paste('WMAPEon test set (1se):', wmape(as.matrix(testres), pred.
↪1se)))
        print(paste('MAE test set (min):', mae(pred.min, as.matrix(testres))))␣
↪# may remove after some runs
        print(paste('MAE test set (1se):', mae(pred.1se, as.matrix(testres))))␣
↪# may remove after some runs
        print('---------')
    }

    invisible(pred.min)
}
```

[297]:
```
# Adds the results to both tables

addToTables <- function(test,
                        prediction,
                        model.name,
                        note,
                        output = FALSE){
    # prepare for the big data.table
    ## pull year and month
    date.text <- test$date
    year = as.numeric(format(date.text,
                             format = '%Y'))
    month = as.numeric(format(date.text,
                              format = '%m'))
    ## calculate WMAPE again
    wmape.res = wmape(test$production,prediction)
    ## create the resulting rows
    k = data.table(date = test$date,
                   year = year,
                   month = month,
                   hour = test$hour,
                   model = model.name,
                   actual = test$production,
                   predicted = prediction,
                   wmape = wmape.res)
    colnames(k) <- colnames(results.main)
    # prepare for the small data.table
    s = data.table(model = model.name,
                   wmape = wmape.res,
                   notes = note)
    colnames(s) <- colnames(results.summary)
    results.main <<- rbind(results.main,k)
    results.summary <<- rbind(results.summary,s)
    if(output){
        return(list(k,s))
    }
}
```

[298]:
```
# Removes results from tables (via modelname)
removeFromTables <- function(modelname){
    results.main <<- results.main[model != modelname]
    results.summary <<- results.summary[model != modelname]
}
```

[299]:
```
# selects data only from specific months (not really needed but used)

# n = integer or list of integers (e.g. n=1, n=c(3,4,9))
```

```
filterMonth <- function(data,
                        n){
    return(data[month(date) == n,])
}
```

[300]:
```
# choose first "n" PCs, explaining x percetange of the variance
choosePC <- function(eigenvalues,
                     percentage){
    sum.ei = sum(eigenvalues)
    tot = 0
    ctr = 0
    for(ei in eigenvalues){
        ctr = ctr + 1
        tot = tot + ei
        checker = tot/sum.ei
        if (checker > percentage) { break }
    }
    return(ctr)
}
```

4.2 Preparing Data

[301]:
```
# SCALING TO MEAN ZERO & 1 SDEV

traintest = scaleWind2(train,
                   test)

train = traintest[[1]]
test  = traintest[[2]]
```

[302]:
```
# CREATING A NEW DATA WITH SQUARES AND CUBES

train.p3 = createPowered(train,c(2,3))
test.p3 = createPowered(test,c(2,3))
```

[4]:
```
# CREATING LAGGED DATA

train.lag = createLagged(train,c(-3:3))
test.lag = createLagged(test,c(-3:3))
```

[5]:
```
# CREATING DATA FOR TEMPORAL RELATIONS

train.temporal = laggedToTemporal(train.lag,colnames(train)[3:6],lags = c(-3:3))
test.temporal = laggedToTemporal(test.lag,colnames(test)[3:6],lags = c(-3:3))
```

5 Phase 1

5.1 Part 1

1. We first apply a General Linear Model (GLM) to the raw data and set a basis performance level.
2. Then we do a Principal Component Analysis (PCA) on the raw data and choose the Principal Components (PC) explaining at least 90% of the variance.
3. Finally we do a Kernel PCA (with polynomial kernel of degree 3) on the raw data and choose the PCs explaining at least 90% of the variance. In this step we could see the nonlinear relationships on the simplest format.

5.1.1 1.a: Raw Data

[593]:
```
# Fit GLM -> Predict on Test -> Measure Performance
pred.raw <- customGlm(traindata = train[,3:6],
                      trainres = train[,7],
                      testdata = test[,3:6],
                      testres = test[,7])
addToTables(test = test,
            prediction = pred.raw,
            model.name = 'Raw Spatial (4)',
            note = 'The model for base performance. GLMNET(cv with lambda.min)␣
 ↪on spatial data')
```

[530]:
```
print(wmape(test[,7], pred.raw))
```

```
[1] 0.3998878
```

WMAPE ~= 0.3999

This is our basis performance level.

5.1.2 1.b: PCA on Raw Data

[307]:
```
# PCA on Raw Train Data - > Apply to Test Data
fit.raw.pca <- prcomp(train[,3:6])
# Choose the minimum number of PCs explaining at least 90% of the variance
n <- choosePC(fit.raw.pca$sdev,0.90)
# Do PCA with first '3' PCs
train.pca <- as.data.frame(predict(fit.raw.pca,
                           train[,3:6])[,1:n])
test.pca  <- as.data.frame(predict(fit.raw.pca,
                           test[,3:6])[,1:n])
```

[595]:
```
# Fit GLM on PCA applied train
pred.raw.pca <- customGlm(train.pca,
                          train[,7],
                          test.pca,
                          test[,7])
addToTables(test = test,
            prediction = pred.raw.pca,
```

```
            model.name = 'PCA Spatial (3)',
            note = 'PCA on spatial data. Took 3 PCs and saw a small but␣
↪expected loss of performance')
```

[309]:
```
results.summary
```

A data.table: 2 × 3

model	wmape	notes
<chr>	<dbl>	<chr>
Base Raw	0.3998878	GLM on Raw Data. (Base Model)
PCA Raw	0.4134921	GLM on PCA of Raw Data. PC: %90 Var. (3 PCs)

We have seen that our Base model has ~0.4 error.

Applying PCA would surely reduce the performance, and it did by a little.

5.1.3 1.c: Kernel PCA Raw

1. ALL PCs
2. PCs that explain the 99.9% Variance

ALL PCs

[446]:
```
k <- nrow(train)
set.seed(42)
tsplit <- sample(c(1:k), size=ceiling(k/2), replace =F)
train.part <- train[tsplit,]
```

[448]:
```
# KERNEL PCA done one partial training data

# 9 minutes (Random half of the data)
start_time <- Sys.time()
fit.raw.kernelpca.part <- kpca(as.matrix(train.part[ ,3:6]),
                           kernel = "polydot",
                           kpar = list(degree = 3,
                                       offset=1,
                                       scale=1))

end_time <- Sys.time()
print(end_time - start_time)
```

```
Time difference of 9.020707 mins
```

[453]:
```
# 10 Minutes. TOO LONG!??
start_time <- Sys.time()
train.raw.kernelpca.part.full <- predict(fit.raw.kernelpca.part, train[,3:6])
test.raw.kernelpca.part.full <- predict(fit.raw.kernelpca.part, test[,3:6])
end_time <- Sys.time()
print(end_time - start_time)
```

```
Time difference of 10.62274 mins
```

[574]:
```
ncol(train.raw.kernelpca.part.full) # number of Principal Components
```

34

[587]:
```
pred.raw.kernelpca.part.full <- customGlm(traindata = train.raw.kernelpca.part.
↪full,
                                          trainres = train[,7],
                                          testdata = test.raw.kernelpca.part.
↪full,
                                          testres = test[,7])
print(wmape(observed = test[,7], predicted = pred.raw.kernelpca.part.full))
```

```
[1] 0.3361963
```

[597]:
```
addToTables(test = test,
            prediction = pred.raw.kernelpca.part.full,
            model.name = 'KernelPCA Spatial All (34)',
            note = 'Kernel PCA on spatial data. PCA fitted on random half of␣
↪the data for RAM reasons. All (34) PCs are used. Very Good performance ')
```

With Kernel PCA we have seen a good improvement in the performance. Adding nonlinearity surely helps.

We predict that spatio-temporal models will outperform this.

Now we will try to reduce dimensions in the next part.

PCs that explain 99.9% of the variance

[483]:
```
choosePC(eig(fit.raw.kernelpca.part),0.999) # number of PCs needed for 99.9%␣
↪variance
```

13

[484]:
```
pred.raw.kernelpca.part.13 <- customGlm(traindata = train.raw.kernelpca.part.
↪full[,1:13],
                                          trainres = train[,7],
                                          testdata = test.raw.kernelpca.part.
↪full[,1:13],
                                          testres = test[,7])
print(wmape(observed = test[,7], predicted = pred.raw.kernelpca.part.13))
```

```
[1] 0.3970043
```

[600]:
```
addToTables(test = test,
            prediction = pred.raw.kernelpca.part.13,
            model.name = 'KernelPCA Spatial 99.9% (13)',
            note = 'Kernel PCA on spatial data. 13 PCs (explaining 99.9%␣
↪variance) are used but the performance is too low. CAUTION')
```

[486]:
```
results.summary
```

A data.table: 8 × 3

model <chr>	wmape <dbl>	notes <chr>
Raw Kernel PCA	0.3361963	Raw Kernel PCA. KPCA done on a random half
Raw Kernel PCA 5PC	0.6737182	Raw Kernel PCA. KPCA done on a random half
Lagged Kernel PCA	0.3990882	Lagged Kernel PCA. KPCA done on a random ha
Lagged Kernel PCA 9 PCs	0.7136027	Lagged Kernel PCA. KPCA done on a random ha
Lagged Kernel PCA	0.3990882	Lagged Kernel PCA. KPCA done on a random ha
Raw Kernel PCA	0.3361963	Raw Kernel PCA. KPCA done on a random half
Lagged Kernel PCA 48 PCs	0.4180692	Lagged Kernel PCA. KPCA done on a random ha
Raw Kernel PCA 13PC	0.3970043	Raw Kernel PCA. KPCA done on a random half

Summary of Kernel PCA: With "Kernel PCA" we were able to create new features that are polynomial combinations of existing features. This helps us to use nonlinearity in our models without increasing the computational complexity.

Using all of the Principal Components gave very good results. However using the Principal Components that explain almost all (99.9%) of the variance performed much worse.

This issue is need to be addressed.

5.1.4 SUMMARY OF PART 1:

Glmnet model of the raw data created a basis performance for our project. Weighted Mean Absolute Error of that model was ~0.40.

Applying PCA on a 4 dimensional data was not a big necessity but doing so and reducing the dimension to 3 again gave a similar result.

For addressing the nonlinearity of wind data we applied Polynomial Kernel PCA and get much better results. As Kernel Matrix grows with the number of instances(rows) we needed to take a random sample from the data for this process, but the results were promising.

For the lagged (spatiotemporal) part. We are expecting an increase in the performance on all three models. However as kernel pca created a question mark, there might be a problem at that part.

6 Part 2

1. We first apply a GLM to the "lagged data".
2. Then we do a PCA on the lagged data and choose the PCs explaining at least 90% of the variance.
3. Finally we do a Kernel PCA (with polynomial kernel of degree 3) on the raw data and choose the PCs explaining at least 90% of the variance.

[602]:
```
# 1: GLM on Lagged Data
pred.lag <- customGlm(traindata = train.lag[,3:30],
                      trainres = train.lag[,31],
                      testdata = test.lag[,3:30],
                      testres = test.lag[,31])

addToTables(test = test.lag,
```

```
            prediction = pred.lag,
            model.name = 'Raw Lagged (28)',
            note = 'Lagged model. Basis for spatiotemporal relationships. ')
```

[603]:
```
print(wmape(test.lag[,31], pred.lag))
```

```
[1] 0.3777897
```

[606]:
```
# 2: PCA on Lag and choose PCs for 95%+ variance

fit.lag.pca <- prcomp(train.lag[,3:30])
# Choose the minimum number of PCs explaining at least 90% of the variance
n <- choosePC(fit.lag.pca$sdev,0.90)
print(n)
```

```
[1] 10
```

[607]:
```
# Do PCA with first 'n' PCs
train.lag.pca <- as.data.frame(predict(fit.lag.pca,
                           train.lag[,3:30])[,1:n])
test.lag.pca  <- as.data.frame(predict(fit.lag.pca,
                           test.lag[,3:30])[,1:n])
# GLM on PCA'd Lag
pred.lag.pca <- customGlm(traindata = train.lag.pca,
                          trainres = train.lag[,31],
                          testdata = test.lag.pca,
                          testres = test.lag[,31])
addToTables(test = test.lag,
            prediction = pred.lag,
            model.name = 'PCA Lagged 90% (10)',
            note = 'PCA on lagged data. 10 PCs are used and the performance did␣
 ↪not change. Nice but as expected.')
```

[490]:
```
results.summary
```

A data.table: 9 × 3

model <chr>	wmape <dbl>	notes <chr>
Raw Kernel PCA	0.3361963	Raw Kernel PCA. KPCA done on a random half
Raw Kernel PCA 5PC	0.6737182	Raw Kernel PCA. KPCA done on a random half
Lagged Kernel PCA	0.3990882	Lagged Kernel PCA. KPCA done on a random ha
Lagged Kernel PCA 9 PCs	0.7136027	Lagged Kernel PCA. KPCA done on a random ha
Lagged Kernel PCA	0.3990882	Lagged Kernel PCA. KPCA done on a random ha
Raw Kernel PCA	0.3361963	Raw Kernel PCA. KPCA done on a random half
Lagged Kernel PCA 48 PCs	0.4180692	Lagged Kernel PCA. KPCA done on a random ha
Raw Kernel PCA 13PC	0.3970043	Raw Kernel PCA. KPCA done on a random half
Lag PCA Glm	0.3777897	GLM on PCA applied Lagged Data. (10 PCs, %9

Summary of Lag PCA: We see with reducing the dimension from 28 to 10 we can preserve the performance of our model.

This will especially be useful when dealing with more spatial or temporal features added to our model. (as in lag7, every 1 spatial feature creates 7 more dimensions)

6.1 3) Kernel PCA on Lagged data

1. All PCs
2. PCs explaining 99.9% Variance

1. All PCs

[467]:
```
# Kernel PCA done on partial Lagged data (half)
start_time <- Sys.time()
k <- nrow(train.lag)
set.seed(42)
tlagsplit <- sample(c(1:k), size=ceiling(k/2), replace =F)
train.lag.part <- train.lag[tlagsplit,]


fit.lag.kernelpca.part <- kpca(as.matrix(train.lag.part[ ,3:30]),
                               kernel = "polydot",
                               kpar = list(degree = 3,
                                           offset=1,
                                           scale=1))

end_time <- Sys.time()
print(end_time - start_time)
#
start_time <- Sys.time()
train.lag.kernelpca.part.full <- predict(fit.lag.kernelpca.part, train.lag[,3:
↪30])
test.lag.kernelpca.part.full <- predict(fit.lag.kernelpca.part, test.lag[,3:30])
end_time <- Sys.time()
print(end_time - start_time)
#
start_time <- Sys.time()
pred.lag.kernelpca.part.full <- customGlm(traindata = train.lag.kernelpca.part.
↪full,
                                          trainres = train.lag[,31],
                                          testdata = test.lag.kernelpca.part.
↪full,
                                          testres = test.lag[,31])
end_time <- Sys.time()
print(end_time - start_time)
print(wmape(observed = test.lag[,31], predicted = pred.lag.kernelpca.part.full))
```

```
Time difference of 7.983299 mins
Time difference of 13.36262 mins
Time difference of 2.585322 mins
[1] 0.3990882
```

[478]:
```
dim(pcv(fit.lag.kernelpca.part))
```

1. 13149 2. 1502

[609]:
```
addToTables(test = test.lag,
            prediction = pred.lag.kernelpca.part.full,
            model.name = 'KernelPCA Lagged All (1502)',
            note = 'Kernel PCA on lagged data. All (1502) PCs are used. Too␣
 ↪many PCs and an unexpectadly bad performance. Possible outfit and/or more␣
 ↪problems.')
```

2. 99.9% Variance:

[698]:
```
choosePC(eig(fit.lag.kernelpca.part),0.99997)
```

300

[480]:
```
start_time <- Sys.time()
pred.lag.kernelpca.part9 <- customGlm(traindata = train.lag.kernelpca.part.
 ↪full[,1:48],
                                      trainres = train.lag[,31],
                                      testdata = test.lag.kernelpca.part.
 ↪full[,1:48],
                                      testres = test.lag[,31])
end_time <- Sys.time()
print(end_time - start_time)
print(wmape(observed = test.lag[,31], predicted = pred.lag.kernelpca.part9))
```

```
Time difference of 1.666866 secs
[1] 0.4180692
```

[611]:
```
addToTables(test = test.lag,
            prediction = pred.lag.kernelpca.part9,
            model.name = 'KernelPCA Lagged 99.9% (48)',
            note = 'Kernel PCA on lagged data. 49 PCs (explaining 99.9%␣
 ↪variance) are used but the performance is even lower. Not thinking about␣
 ↪overfit but there surely is a problem.')
```

[687]:
```
results.summary[6:8,]
```

A data.table: 3 × 3

model <chr>	wmape <dbl>	notes <chr>
PCA Lagged 90% (10)	0.3777897	PCA on lagged data. 10 PCs are used and the
KernelPCA Lagged All (1502)	0.3990882	Kernel PCA on lagged data. All (1502) PCs ar
KernelPCA Lagged 99.9% (48)	0.4180692	Kernel PCA on lagged data. 49 PCs (explainir

[694]:
```
start_time <- Sys.time()
pred.lag.kernelpca.part9 <- customGlm(traindata = train.lag.kernelpca.part.
 ↪full[,1:320],
                                      trainres = train.lag[,31],
```

```
                                            testdata = test.lag.kernelpca.part.
↪full[,1:320],
                                            testres = test.lag[,31])
end_time <- Sys.time()
print(end_time - start_time)
print(wmape(observed = test.lag[,31], predicted = pred.lag.kernelpca.part9))
```

```
Time difference of 24.02167 secs
[1] 0.3448457
```

[695]:
```
start_time <- Sys.time()
pred.lag.kernelpca.part9 <- customGlm(traindata = train.lag.kernelpca.part.
↪full[,1:300],
                                      trainres = train.lag[,31],
                                      testdata = test.lag.kernelpca.part.
↪full[,1:300],
                                      testres = test.lag[,31])
end_time <- Sys.time()
print(end_time - start_time)
print(wmape(observed = test.lag[,31], predicted = pred.lag.kernelpca.part9))
```

```
Time difference of 22.47746 secs
[1] 0.3465964
```

[692]:
```
start_time <- Sys.time()
pred.lag.kernelpca.part9 <- customGlm(traindata = train.lag.kernelpca.part.
↪full[,1:451],
                                      trainres = train.lag[,31],
                                      testdata = test.lag.kernelpca.part.
↪full[,1:451],
                                      testres = test.lag[,31])
end_time <- Sys.time()
print(end_time - start_time)
print(wmape(observed = test.lag[,31], predicted = pred.lag.kernelpca.part9))
```

```
Time difference of 49.36903 secs
[1] 0.3470405
```

[699]:
```
addToTables(test = test.lag,
            prediction = pred.lag.kernelpca.part9,
            model.name = 'KernelPCA Lagged 99.997% (300)',
            note = 'Kernel PCA on lagged data. 300 PCs (explaining 99.997%␣
↪variance) are used. Explained variance changes the performance a lot. 300 is␣
↪found with monkey method.')
```

[701]:
```
table1 <- results.summary[c(1, 2, 3, 16),]
table2 <- results.summary[c(5, 6, 17),]
table3 <- results.summary[c(10, 12, 14),]
```

[702]:
```
table1
table2
table3
```

A data.table: 4 × 3

model <chr>	wmape <dbl>	notes <chr>
Raw Spatial (4)	0.3998878	The model for base performance. GLMNET(cv w
PCA Spatial (3)	0.4134921	PCA on spatial data. Took 3 PCs and saw a smal
KernelPCA Spatial All (34)	0.3361963	Kernel PCA on spatial data. PCA fitted on rando
Raw Cubed (12)	0.3449375	The cubed model. Basis for nonlinear relationship

A data.table: 3 × 3

model <chr>	wmape <dbl>	notes <chr>
Raw Lagged (28)	0.3777897	Lagged model. Basis for spatiotemporal rel
PCA Lagged 90% (10)	0.3777897	PCA on lagged data. 10 PCs are used and
KernelPCA Lagged 99.997% (300)	0.3465964	Kernel PCA on lagged data. 300 PCs (expl

A data.table: 3 × 3

model <chr>	wmape <dbl>	notes <chr>
KroneckerPCA [train, temporal] 99.9% (12)	0.3777050	Basis Kronecker PC
Nonlinear KroneckerPCA [train.p3, temporal] 99.9% (25)	0.3348935	Cubed Kronecker PC
Nonlinear KroneckerPCA [train, temporal.p3] 99.9% (29)	0.3493172	Cubed Kronecker PC

6.1.1 Summary of Kernel PCA Lag:

Kernel PCA on lagged data reduced the performance. Even when using the all Principal components we see a worse result than normal lag model.

The issue might be the too high number of principal components (1502) but using the first 48 (99.9% Var) Principal Components we get even worse performance.

This also needs to be adressed.

[Note: It is not adressed in this project but using Kernel PCA on each month to create new features were improving the performance of Lagged model. So we can say that Kernel PCA works.]

7 KRONECKER PCA ON RAW SPATIAL AND TEMPORAL DATA

All PCs

[361]:
```
kronecker.temporal <- cov(train.temporal[,1:7], train.temporal[,1:7]) #i.e. cov␣
↪on temporal data
kronecker.spatial <- cov(train[,3:6], train[,3:6]) #i.e. cov on spatial
kronecker.cov <- kronecker(kronecker.temporal, kronecker.spatial) # kronecker␣
↪(spatiotemporal)
```

[535]:
```
kronecker.vectors <- eigen(kronecker.cov)$vectors
```

[536]:
```
train.kroneckerpca <- as.matrix(train.lag[,3:30]) %*% kronecker.vectors
test.kroneckerpca  <- as.matrix(test.lag[,3:30]) %*% kronecker.vectors
```

[537]:
```
pred.kronecker <- customGlm(traindata = train.kroneckerpca,
                            trainres = train.lag[,31],
                            testdata = test.kroneckerpca,
                            testres = test.lag[,31])

print(wmape(observed = test.lag[,31], predicted = pred.kronecker))
```

```
[1] 0.381502
```

[613]:
```
addToTables(test = test.lag,
            prediction = pred.kronecker,
            model.name = 'KroneckerPCA [train, temporal] All (28) ',
            note = 'Basis Kronecker PCA. All (28) PCs are used. OK performance.␣
 ↪Should be compared to Lag PCA and we can say that they are similar.')
```

Let's use %99+ variance explaining data

[540]:
```
choosePC(eigen(kronecker.cov)$values, 0.999)
```

12

[541]:
```
kronecker.vectors12 <- eigen(kronecker.cov)$vectors[,1:12]
```

[542]:
```
train.kroneckerpca12 <- as.matrix(train.lag[,3:30]) %*% kronecker.vectors12
test.kroneckerpca12  <- as.matrix(test.lag[,3:30]) %*% kronecker.vectors12
```

[543]:
```
pred.kronecker12 <- customGlm(traindata = train.kroneckerpca12,
                            trainres = train.lag[,31],
                            testdata = test.kroneckerpca12,
                            testres = test.lag[,31])

print(wmape(observed = test.lag[,31], predicted = pred.kronecker12))
```

```
[1] 0.377705
```

[615]:
```
addToTables(test = test.lag,
            prediction = pred.kronecker12,
            model.name = 'KroneckerPCA [train, temporal] 99.9% (12)',
            note = 'Basis Kronecker PCA with less dimensions. 12 PCs␣
 ↪(explaining 99.9% variance) are used. A little improvement compared to all␣
 ↪PCs: signalling an overfit.')
```

[554]:
```
results.summary
```

A data.table: 15 × 3

model <chr>	wmape <dbl>	notes <chr>
Raw Kernel PCA	0.3361963	Raw Kernel PCA. KPCA c
Raw Kernel PCA 5PC	0.6737182	Raw Kernel PCA. KPCA c
Lagged Kernel PCA	0.3990882	Lagged Kernel PCA. KPC
Lagged Kernel PCA 9 PCs	0.7136027	Lagged Kernel PCA. KPC
Lagged Kernel PCA	0.3990882	Lagged Kernel PCA. KPC
Raw Kernel PCA	0.3361963	Raw Kernel PCA. KPCA c
Lagged Kernel PCA 48 PCs	0.4180692	Lagged Kernel PCA. KPC
Raw Kernel PCA 13PC	0.3970043	Raw Kernel PCA. KPCA c
Lag PCA Glm	0.3777897	GLM on PCA applied Lagg
Nonlinear(spatial cube) Kronecker PCA	0.3389212	Kronecker PCA with cov(t
Nonlinear(spatial cube) Kronecker PCA 25PCs	0.3348935	Kronecker PCA with cov(t
Nonlinear(temporal cube) Kronecker PCA	0.3371075	Kronecker PCA with cov(t
Nonlinear(temporal cube) Kronecker PCA 29PCs	0.3493172	Kronecker PCA with cov(t
Kronecker PCA	0.3815020	Kronecker PCA on raw spa
Kronecker PCA 12PCs	0.3777050	Kronecker PCA on raw spa

7.1 Summary of Kronecker PCA:

Using less PCs improved the results compared to all PCs.

We should compare these results with Lag PCA: They are very similar

Note that: in bigger (with more features or lags) datasets using kronecker pca would give a nicer computational performance. However it generalizes the relationship in between the spatial features, and in between the temporal features, so it might be missing some details.

Also note that The temporal covariance part can also be used universally.

8 KRONECKER PCA ON POWERED TEMPORAL AND SPATIAL DATA

8.0.1 2 Main 2 Sub parts:

1. Spatial Powered X Temporal Normal
   - All PCs
   - PCs explaining 99.9% variance
2. Spatial Normal X Temporal Powered
   - All PCs
   - PCs explaining 99.9% variance

8.0.2 1: Spatial Powered X Temporal Normal

1.a: All PCs

[499]:
```
kronecker.temporal <- cov(train.temporal[,1:7], train.temporal[,1:7])
kronecker.spatial.p3 <- cov(train.p3[,3:14], train.p3[,3:14])
```

[500]:
```
kronecker.s.p3 <- kronecker(kronecker.temporal, kronecker.spatial.p3)
```

[501]:
```
train.p3.lag <- createLagged(train.p3,lags = c(-3:3),c(1,2),15)
test.p3.lag <- createLagged(test.p3,lags = c(-3:3),c(1,2),15)
```

[516]:
```
kronecker.s.p3.vectors <- eigen(kronecker.s.p3)$vectors
```

[517]:
```
train.s.p3.kroneckerpca <- as.matrix(train.p3.lag[,3:86]) %*% kronecker.s.p3.
 ↪vectors
test.s.p3.kroneckerpca  <- as.matrix(test.p3.lag[,3:86]) %*% kronecker.s.p3.
 ↪vectors
```

[518]:
```
dim(kronecker.s.p3.vectors)
```

1. 84 2. 84

[519]:
```
pred.s.p3.kronecker <- customGlm(traindata = train.s.p3.kroneckerpca,
                                 trainres = train.lag[,31],
                                 testdata = test.s.p3.kroneckerpca,
                                 testres = test.lag[,31])
print(wmape(test.lag[,31], pred.s.p3.kronecker))
```

```
[1] 0.3389212
```

[617]:
```
addToTables(test = test.lag,
            prediction = pred.s.p3.kronecker,
            model.name = "Nonlinear KroneckerPCA [train.p3, temporal] All (84)␣
 ↪",
            note = "Cubed Kronecker PCA. Cubed spatial data and normal temporal␣
 ↪data are used. All (84) PCs are used. Promising results. Outperforms␣
 ↪KernelPCA Lag.")
```

1.b: 99.9% Variance

[507]:
```
n <- choosePC(eigen(kronecker.s.p3)$values, 0.999)
n
```

25

[508]:
```
pred.s.p3.kronecker25 <- customGlm(traindata = train.s.p3.kroneckerpca[,1:25],
                                 trainres = train.lag[,31],
                                 testdata = test.s.p3.kroneckerpca[,1:25],
                                 testres = test.lag[,31])
print(wmape(test.lag[,31], pred.s.p3.kronecker25))
```

```
[1] 0.3348935
```

[619]:
```
addToTables(test = test.lag,
            prediction = pred.s.p3.kronecker25,
            model.name = "Nonlinear KroneckerPCA [train.p3, temporal] 99.9%␣
 ↪(25)",
```

```
            note = "Cubed Kronecker PCA. Cubed spatial data and normal temporal␣
↪data are used. 25 PCs (99.9% Variance) are used. No reduce in performance,␣
↪nice! ")
```

[405]:
```
results.summary
```

A data.table: 8 × 3

model <chr>	wmape <dbl>	notes <chr>
Base Raw	0.3998878	GLM on Raw Data. (Base Model)
PCA Raw	0.4134921	GLM on PCA of Raw Data. PC: %90 Var. (
Raw Kernel PCA-1	0.3362274	Raw Kernel PCA January. KPCA done on m
Lag Glm	0.3777897	GLM on Lagged Data
Lag PCA Glm	0.3777897	GLM on PCA applied Lagged Data. (15 PCs
Lagged Kernel PCA 1 -	0.3710024	Lagged Kernel PCA January. KPCA done or
Kronecker PCA	0.3826951	Kronecker PCA on raw spatial and temporal
Nonlinear(cube) Kronecker PCA	0.3389212	Kronecker PCA with cov(train.p3) and cov(t

[406]:
```
results.summary2
```

A data.table: 29 × 3

model <chr>	wmape <dbl>	notes <chr>
Base Raw	0.3998878	GLM on Raw Data. (Base Model)
PCA Raw	0.4134921	GLM on PCA of Raw Data. PC: %90 Var. (3 PCs)
Raw Kernel PCA-1	0.3362274	Raw Kernel PCA January. KPCA done on month:
Raw Kernel PCA 2 -	0.3366250	Raw Kernel PCA January. KPCA done on month:
Raw Kernel PCA 3 -	0.3354890	Raw Kernel PCA January. KPCA done on month:
Raw Kernel PCA 4 -	0.3368440	Raw Kernel PCA January. KPCA done on month:
Raw Kernel PCA 5 -	0.3355147	Raw Kernel PCA January. KPCA done on month:
Raw Kernel PCA 6 -	0.3362304	Raw Kernel PCA January. KPCA done on month:
Raw Kernel PCA 7 -	0.3367962	Raw Kernel PCA January. KPCA done on month:
Raw Kernel PCA 8 -	0.3369167	Raw Kernel PCA January. KPCA done on month:
Raw Kernel PCA 9 -	0.3357871	Raw Kernel PCA January. KPCA done on month:
Raw Kernel PCA 10 -	0.3358707	Raw Kernel PCA January. KPCA done on month:
Raw Kernel PCA 11 -	0.3359437	Raw Kernel PCA January. KPCA done on month:
Raw Kernel PCA 12 -	0.3363214	Raw Kernel PCA January. KPCA done on month:
Lag Glm	0.3777897	GLM on Lagged Data
Lag PCA Glm	0.3777897	GLM on PCA applied Lagged Data. (15 PCs, %95)
Lagged Kernel PCA 1 -	0.3710024	Lagged Kernel PCA January. KPCA done on mont
Lagged Kernel PCA 2 -	0.3757470	Lagged Kernel PCA January. KPCA done on mont
Lagged Kernel PCA 3 -	0.3787904	Lagged Kernel PCA January. KPCA done on mont
Lagged Kernel PCA 4 -	0.3852651	Lagged Kernel PCA January. KPCA done on mont
Lagged Kernel PCA 5 -	0.3827968	Lagged Kernel PCA January. KPCA done on mont
Lagged Kernel PCA 6 -	0.3812209	Lagged Kernel PCA January. KPCA done on mont
Lagged Kernel PCA 7 -	0.3815486	Lagged Kernel PCA January. KPCA done on mont
Lagged Kernel PCA 8 -	0.3673643	Lagged Kernel PCA January. KPCA done on mont
Lagged Kernel PCA 9 -	0.3689898	Lagged Kernel PCA January. KPCA done on mont
Lagged Kernel PCA 10 -	0.3796700	Lagged Kernel PCA January. KPCA done on mont
Lagged Kernel PCA 11 -	0.3679311	Lagged Kernel PCA January. KPCA done on mont
Lagged Kernel PCA 12 -	0.3653790	Lagged Kernel PCA January. KPCA done on mont
Kronecker PCA	0.3826951	Kronecker PCA on raw spatial and temporal data.

[412]:
```
# fwrite(results.summary, "results.summary.28dec.csv")
# fwrite(results.summary2, "results.summary2.28dec.csv")
# fwrite(results.main, "results.main.28dec.csv")
```

8.0.3 2: Spatial Normal X Temporal Powered

2.a: All PCs

[510]:
```
train.temporal.p3 <- createPowered(data = train.temporal,
                                   powers = c(2,3),
                                   other.head = NULL,
                                   other.tail = c(8,9))
```

[511]:
```
kronecker.temporal.p3 <- cov(train.temporal.p3[,1:21],train.temporal.p3[,1:21])
kronecker.spatial <- cov(train[,3:6],train[,3:6])
```

[512]:
```
kronecker.t.p3 <- kronecker(kronecker.temporal.p3, kronecker.spatial)
```

[514]:
```
# we change the column orders for compatibility with kronecker product matrix
train.lag.p3 <- createPowered(train.lag, powers = c(2:3), c(1,2),31)
test.lag.p3 <- createPowered(test.lag, powers = c(2:3), c(1,2),31)
```

[520]:
```
kronecker.t.p3.vectors <- eigen(kronecker.t.p3)$vectors
```

[522]:
```
train.t.p3.kroneckerpca <- as.matrix(train.lag.p3[,3:86]) %*% kronecker.t.p3.
 ↪vectors
test.t.p3.kroneckerpca  <- as.matrix(test.lag.p3[,3:86]) %*% kronecker.t.p3.
 ↪vectors
```

[523]:
```
pred.t.p3.kronecker <- customGlm(traindata = train.t.p3.kroneckerpca,
                                 trainres = train.lag[,31],
                                 testdata = test.t.p3.kroneckerpca,
                                 testres = test.lag[,31])
print(wmape(test.lag[,31], pred.t.p3.kronecker))
```

```
[1] 0.3371075
```

[621]:
```
addToTables(test = test.lag,
            prediction = pred.t.p3.kronecker,
            model.name = "Nonlinear KroneckerPCA [train, temporal.p3] All (84)␣
 ↪",
            note = "Cubed Kronecker PCA. Normal spatial data and cubed temporal␣
 ↪data are used. All (84) PCs are used. It is an another approach of cubed␣
 ↪kronecker. Gave a very similar result.")
```

2.b: 99.9% Variance

[525]:
```
n <- choosePC(eigen(kronecker.t.p3)$values, 0.999)
n
```

```
29
```

[527]:
```
pred.t.p3.kronecker29 <- customGlm(traindata = train.t.p3.kroneckerpca[,1:29],
                                 trainres = train.lag[,31],
                                 testdata = test.t.p3.kroneckerpca[,1:29],
                                 testres = test.lag[,31])
print(wmape(test.lag[,31], pred.t.p3.kronecker29))
```

```
[1] 0.3493172
```

[623]:
```
addToTables(test = test.lag,
            prediction = pred.t.p3.kronecker29,
            model.name = "Nonlinear KroneckerPCA [train, temporal.p3] 99.9%␣
 ↪(29)",
            note = "Cubed Kronecker PCA. Normal spatial data and cubed temporal␣
 ↪data are used. 29 PCs (99.9% Variance) are used. A little loss of␣
 ↪performance which can be acceptable.")
```

[624]:
```
results.summary
```

A data.table: 14 × 3

model <chr>	wmape <dbl>	notes <chr>
Raw Spatial (4)	0.3998878	The model for base
PCA Spatial (3)	0.4134921	PCA on spatial dat
KernelPCA Spatial All (34)	0.3361963	Kernel PCA on spa
KernelPCA Spatial 99.9% (13)	0.3970043	Kernel PCA on spa
Raw Lagged (28)	0.3777897	Lagged model. Bas
PCA Lagged 90% (10)	0.3777897	PCA on lagged dat
KernelPCA Lagged All (1502)	0.3990882	Kernel PCA on lag
KernelPCA Lagged 99.9% (48)	0.4180692	Kernel PCA on lag
KroneckerPCA [train, temporal] All (28)	0.3815020	Basis Kronecker PC
KroneckerPCA [train, temporal] 99.9% (12)	0.3777050	Basis Kronecker PC
Nonlinear KroneckerPCA [train.p3, temporal] All (84)	0.3389212	Cubed Kronecker P
Nonlinear KroneckerPCA [train.p3, temporal] 99.9% (25)	0.3348935	Cubed Kronecker P
Nonlinear KroneckerPCA [train, temporal.p3] All (84)	0.3371075	Cubed Kronecker P
Nonlinear KroneckerPCA [train, temporal.p3] 99.9% (29)	0.3493172	Cubed Kronecker P

9 Finished!

10 Summary:

- Spatial Part
  1. Base Model: 0.400 WMAPE
  2. PCA on Raw: 0.413 WMAPE
  3. Kernel PCA (Partial):
     1. All PCs: 0.336 WMAPE
     2. 13 PCs (99.9% Variance) : 0.397 WMAPE. This is too low compared to All PCs and even compared to the base model. There might be a problem.
- Spatio-Temporal Part
  1. Base Lagged Model: 0.378 WMAPE
  2. PCA on Lagged with 10 PCs (99.9% Variance): 0.378 WMAPE. We can preserve the performance with highly reducing the dimension
  3. Kernel PCA (Partial):
     1. All (1502) PCs: 0.399 WMAPE. Too many principal components but the results are worse
     2. 48 PCs: 0.418 WMAPE. Choosing less PCs still reduces the performance. There is a problem
- Kronecker Spatio-Temporal Part
  1. Kronecker Base:
     1. All (28) PCs: 0.382 WMAPE
     2. 12 PCs (99.9% Variance): 0.378 WMPA. The performance increased. There was an overfitting above.
     – We may compare this to Lagged PCA. The results are similar. Both have some pros and cons
  2. Kronecker Powered:

1. Powered Spatial & Normal Temporal
    1. All (84) PCs: 0.339 WMAPE
    2. 25 PCs (99.9% Variance): 0.335 WMAPE.
2. Normal Spatial & Powered Temporal
    1. All (84) PCs: 0.337 WMAPE
    2. 29 PCs (99.9% Variance): 0.349 WMAPE.

[683]:
```
results.summary
```

A data.table: 16 × 3

model <chr>	wmape <dbl>	notes <chr>
Raw Spatial (4)	0.3998878	The model for base
PCA Spatial (3)	0.4134921	PCA on spatial dat
KernelPCA Spatial All (34)	0.3361963	Kernel PCA on spa
KernelPCA Spatial 99.9% (13)	0.3970043	Kernel PCA on spa
Raw Lagged (28)	0.3777897	Lagged model. Bas
PCA Lagged 90% (10)	0.3777897	PCA on lagged dat
KernelPCA Lagged All (1502)	0.3990882	Kernel PCA on lag
KernelPCA Lagged 99.9% (48)	0.4180692	Kernel PCA on lag
KroneckerPCA [train, temporal] All (28)	0.3815020	Basis Kronecker PC
KroneckerPCA [train, temporal] 99.9% (12)	0.3777050	Basis Kronecker PC
Nonlinear KroneckerPCA [train.p3, temporal] All (84)	0.3389212	Cubed Kronecker P
Nonlinear KroneckerPCA [train.p3, temporal] 99.9% (25)	0.3348935	Cubed Kronecker P
Nonlinear KroneckerPCA [train, temporal.p3] All (84)	0.3371075	Cubed Kronecker P
Nonlinear KroneckerPCA [train, temporal.p3] 99.9% (29)	0.3493172	Cubed Kronecker P
PCA Cubed (11)	0.3535546	PCA on cubed data
Raw Cubed (12)	0.3449375	The cubed model.

[628]:
```
results.summary.short <- results.summary[c(1,2,3,5,6,7,10,11,13),]
results.summary.short[order(wmape)]
```

A data.table: 9 × 3

model <chr>	wmape <dbl>	notes <chr>
KernelPCA Spatial All (34)	0.3361963	Kernel PCA on spatial
Nonlinear KroneckerPCA [train, temporal.p3] All (84)	0.3371075	Cubed Kronecker PCA.
Nonlinear KroneckerPCA [train.p3, temporal] All (84)	0.3389212	Cubed Kronecker PCA.
KroneckerPCA [train, temporal] 99.9% (12)	0.3777050	Basis Kronecker PCA w
Raw Lagged (28)	0.3777897	Lagged model. Basis fo
PCA Lagged 90% (10)	0.3777897	PCA on lagged data. 1
KernelPCA Lagged All (1502)	0.3990882	Kernel PCA on lagged
Raw Spatial (4)	0.3998878	The model for base per
PCA Spatial (3)	0.4134921	PCA on spatial data. T

[680]:
```
comparison.table.spatiotemporal.raw <- results.summary[c(5,6,9,10),]

comparison.table.spatial.powered <- results.summary[c(3,4,16),]

comparison.table.spatiotemporal.powered <- results.summary[c(7,8,11,12,13,14),]
```

[682]:
```
comparison.table.spatiotemporal.raw[order(wmape)]
comparison.table.spatial.powered[order(wmape)]
comparison.table.spatiotemporal.powered[order(wmape)]
```

A data.table: 4 × 3

model <chr>	wmape <dbl>	notes <chr>
KroneckerPCA [train, temporal] 99.9% (12)	0.3777050	Basis Kronecker PCA with less di
Raw Lagged (28)	0.3777897	Lagged model. Basis for spatioten
PCA Lagged 90% (10)	0.3777897	PCA on lagged data. 10 PCs are
KroneckerPCA [train, temporal] All (28)	0.3815020	Basis Kronecker PCA. All (28) PC

A data.table: 3 × 3

model <chr>	wmape <dbl>	notes <chr>
KernelPCA Spatial All (34)	0.3361963	Kernel PCA on spatial data. PCA fitted on ra
Raw Cubed (12)	0.3449375	The cubed model. Basis for nonlinear relations
KernelPCA Spatial 99.9% (13)	0.3970043	Kernel PCA on spatial data. 13 PCs (explainir

A data.table: 6 × 3

model <chr>	wmape <dbl>	notes <chr>
Nonlinear KroneckerPCA [train.p3, temporal] 99.9% (25)	0.3348935	Cubed Kronecker PC
Nonlinear KroneckerPCA [train, temporal.p3] All (84)	0.3371075	Cubed Kronecker PC
Nonlinear KroneckerPCA [train.p3, temporal] All (84)	0.3389212	Cubed Kronecker PC
Nonlinear KroneckerPCA [train, temporal.p3] 99.9% (29)	0.3493172	Cubed Kronecker PC
KernelPCA Lagged All (1502)	0.3990882	Kernel PCA on lagg
KernelPCA Lagged 99.9% (48)	0.4180692	Kernel PCA on lagg

10.0.1 Problems:

1. Raw Kernel PCA works weirdly. Using the PCs that have almost zero impact on the variance improves the performance a lot!

10.1 EXTENSION: CUBED PART

10.1.1 We Also decided to add Cubed Raw model for comparison

[630]:
```
# Cubed Raw GLM
pred.p3 <- customGlm(traindata = train.p3[,3:14],
                     trainres = train.p3[,15],
                     testdata = test.p3[,3:14],
                     testres = test.p3[,15])
```

[631]:
```
print(wmape(test.p3[,15],pred.p3))
```

```
[1] 0.3449375
```

[673]:
```
addToTables(test = test.p3,
            prediction = pred.p3,
            model.name = 'Raw Cubed (12)',
            note = 'The cubed model. Basis for nonlinear relationships')
```

[668]:
```
# PCA on Cubed (less than 98% variance reduces the performance by a lot!)

fit.p3.pca <- prcomp(train.p3[,3:14])
# Choose the minimum number of PCs explaining at least 99% of the variance
n <- choosePC(fit.p3.pca$sdev,0.99)
print(n)
# Do PCA with first 'n' PCs
train.p3.pca <- as.data.frame(predict(fit.p3.pca,
                           train.p3[,3:14])[,1:n])
test.p3.pca  <- as.data.frame(predict(fit.p3.pca,
                           test.p3[,3:14])[,1:n])
```

```
[1] 11
```

[669]:
```
pred.p3.pca <- customGlm(traindata = train.p3.pca,
                         trainres = train.p3[,15],
                         testdata = test.p3.pca,
                         testres = test.p3[,15])
print(wmape(test.p3[,15],pred.p3.pca))
```

```
[1] 0.3535546
```

[670]:
```
addToTables(test = test.p3,
            prediction = pred.p3.pca,
            model.name = 'PCA Cubed (11)',
            note = 'PCA on cubed data. Using 11 PCs would give a similar␣
 ↪performance, but is it really a dimension reduction?')
```

[684]:
```
results.summary.short[order(wmape)]
```

A data.table: 10 × 3

model <chr>	wmape <dbl>	notes <chr>
KernelPCA Spatial All (34)	0.3361963	Kernel PCA on spatia
Nonlinear KroneckerPCA [train, temporal.p3] All (84)	0.3371075	Cubed Kronecker PCA
Nonlinear KroneckerPCA [train.p3, temporal] All (84)	0.3389212	Cubed Kronecker PCA
Raw Cubed (12)	0.3449375	The cubed model. Bas
KroneckerPCA [train, temporal] 99.9% (12)	0.3777050	Basis Kronecker PCA
Raw Lagged (28)	0.3777897	Lagged model. Basis f
PCA Lagged 90% (10)	0.3777897	PCA on lagged data.
KernelPCA Lagged All (1502)	0.3990882	Kernel PCA on lagged
Raw Spatial (4)	0.3998878	The model for base pe
PCA Spatial (3)	0.4134921	PCA on spatial data.

10.2 Lets also add Lagged and Powered:

[703]:
```
train.p3.lag
```

date	hour	ne_lag3	nw_lag3	se_lag3	sw_lag3	ne_p2_
<date>	<int>	<dbl>	<dbl>	<dbl>	<dbl>	<dbl>
2017-07-01	3	0.07495450	0.00578771	-0.13545788	-0.136725959	0.00561
2017-07-01	4	0.23737605	0.21002174	0.08370014	0.109084938	0.05634
2017-07-01	5	0.39980240	0.41760091	0.30487559	0.359080946	0.15984
2017-07-01	6	0.56223248	0.62758994	0.52745232	0.611840513	0.31610
2017-07-01	7	0.64644333	0.74679758	0.57877085	0.678553228	0.41788
2017-07-01	8	0.73617432	0.87312977	0.64123292	0.750859818	0.54195
2017-07-01	9	0.83081931	1.00555009	0.71395342	0.828259686	0.69026
2017-07-01	10	0.66119017	0.87184051	0.54855342	0.620178256	0.43717
2017-07-01	11	0.49940323	0.74373531	0.39680194	0.424921535	0.24940
2017-07-01	12	0.34739373	0.62220428	0.26196051	0.246572368	0.12068
2017-07-01	13	0.29575182	0.40975192	0.26608336	0.104199383	0.08746
2017-07-01	14	0.24460568	0.19741674	0.27155551	-0.037610876	0.05983
2017-07-01	15	0.19399724	-0.01475415	0.27836545	-0.178689420	0.03763
2017-07-01	16	0.14062247	-0.08413297	0.17326471	-0.230086248	0.01977
2017-07-01	17	0.10532765	-0.12293893	0.09092375	-0.239096021	0.01109
2017-07-01	18	0.08923289	-0.12874956	0.03469349	-0.204894581	0.00796
2017-07-01	19	0.11838080	-0.15715320	0.09158141	-0.162733994	0.01401
2017-07-01	20	0.15893325	-0.17495472	0.17683130	-0.090755238	0.02525
2017-07-01	21	0.21013246	-0.18175463	0.28669136	0.007028656	0.04415
2017-07-01	22	0.35100982	-0.10719369	0.49875301	0.101790891	0.12320
2017-07-01	23	0.49414804	-0.02024829	0.71307919	0.224005625	0.24418
2017-07-02	0	0.63911750	0.07715573	0.92910427	0.368554624	0.40847
2017-07-02	1	0.38840314	-0.10493110	0.51427451	0.049064924	0.15085
2017-07-02	2	0.14406884	-0.27963776	0.10448086	-0.260742981	0.02075
2017-07-02	3	-0.09082494	-0.44361218	-0.29567962	-0.552207584	0.00824
2017-07-02	4	-0.18869373	-0.54258237	-0.36970168	-0.628810153	0.03560
2017-07-02	5	-0.27943238	-0.63850525	-0.43330655	-0.702043943	0.07808
2017-07-02	6	-0.36149819	-0.73037093	-0.48477939	-0.771010725	0.13068
2017-07-02	7	-0.37948129	-0.72323358	-0.49981568	-0.758554900	0.14400
A data.table: 26298 × 87 2017-07-02	8	-0.39714948	-0.71178975	-0.51464080	-0.744502865	0.15772
2020-06-29	15	0.19569270	-0.41929281	-0.98613515	-0.32936132	0.03829
2020-06-29	16	0.29120227	-0.45949930	-0.68102880	-0.18935168	0.08479
2020-06-29	17	0.39091125	-0.49899949	-0.34185771	-0.02277363	0.15281
2020-06-29	18	0.49422571	-0.53771454	0.01056994	0.16283775	0.24425
2020-06-29	19	0.56356049	-0.32671651	0.37324674	0.15980958	0.31760
2020-06-29	20	0.63299375	-0.11500367	0.74264456	0.17484227	0.40068
2020-06-29	21	0.70251669	0.09715636	1.11606691	0.20748412	0.49352
2020-06-29	22	0.62028392	0.02581754	0.95464672	0.47639942	0.38475
2020-06-29	23	0.53882199	-0.04552119	0.79720157	0.75550169	0.29032
2020-06-30	0	0.45822152	-0.11685983	0.64455176	1.04155833	0.20996
2020-06-30	1	0.38584455	-0.10283095	0.34806521	0.92545524	0.14887
2020-06-30	2	0.31521970	-0.08782047	0.05338312	0.80969339	0.09936
2020-06-30	3	0.24654579	-0.07185735	-0.23835979	0.69432453	0.06078
2020-06-30	4	0.17740763	-0.19662146	-0.19955405	0.64952260	0.03147
2020-06-30	5	0.11125034	-0.31776294	-0.14459312	0.61563309	0.01237
2020-06-30	6	0.04842846	-0.43415118	-0.07517315	0.59316298	0.00234
2020-06-30	7	-0.02436353	-0.49251506	-0.22349216	0.47696857	0.00059
2020-06-30	8	-0.09667964	-0.55041294	-0.37007585	0.36093482	0.00934
2020-06-30	9	-0.16844877	-0.60776318	-0.51424901	0.24509211	0.02837
2020-06-30	10	-0.06786233	-0.50238922	-0.50337854	0.25859291	0.00460

[705]:
```
start_time <- Sys.time()
pred.p3.lag <- customGlm(traindata = train.p3.lag[,3:86],
                         trainres = train.p3.lag[,87],
                         testdata = test.p3.lag[,3:86],
                         testres = test.p3.lag[,87])
end_time <- Sys.time()
print(end_time - start_time)
```

```
Time difference of 3.933087 secs
```

[706]:
```
print(wmape(test.p3.lag[,87], pred.p3.lag))
```

```
[1] 0.3332918
```

[707]:
```
addToTables(test = test.p3.lag,
            prediction = pred.p3.lag,
            model.name = 'Cubed Lagged (84)',
            note = 'Normal glmnet on Cubed & Lagged data. Runs fast, performs␣
 ↪well')
```

[708]:
```
pca.p3.lag <- prcomp(train.p3.lag[,3:86])
train.p3.lag.pca <- predict(pca.p3.lag, train.p3.lag[,3:86])
test.p3.lag.pca  <- predict(pca.p3.lag, test.p3.lag[,3:86])
```

[709]:
```
choosePC(pca.p3.lag$sdev,percentage = 0.99)
```

62

[710]:
```
pred.p3.lag.pca <- customGlm(traindata = train.p3.lag.pca[,1:62],
                             trainres = train.p3.lag[,87],
                             testdata = test.p3.lag.pca[,1:62],
                             testres = test.p3.lag[,87])
```

[711]:
```
print(wmape(test.p3.lag[,87], pred.p3.lag.pca))
```

```
[1] 0.3336829
```

[712]:
```
addToTables(test = test.p3.lag,
            prediction = pred.p3.lag.pca,
            model.name = 'PCA Cubed Lagged (62)',
            note = 'PCA on Cubed & Lagged data. Runs fast, performs well with␣
 ↪62 columns')
```

[717]:
```
results.summary
```

A data.table: 19 × 3

model <chr>	wmape <dbl>	notes <chr>
Raw Spatial (4)	0.3998878	The model for base
PCA Spatial (3)	0.4134921	PCA on spatial dat
KernelPCA Spatial All (34)	0.3361963	Kernel PCA on spa
KernelPCA Spatial 99.9% (13)	0.3970043	Kernel PCA on spa
Raw Lagged (28)	0.3777897	Lagged model. Bas
PCA Lagged 90% (10)	0.3777897	PCA on lagged dat
KernelPCA Lagged All (1502)	0.3990882	Kernel PCA on lag
KernelPCA Lagged 99.9% (48)	0.4180692	Kernel PCA on lag
KroneckerPCA [train, temporal] All (28)	0.3815020	Basis Kronecker PC
KroneckerPCA [train, temporal] 99.9% (12)	0.3777050	Basis Kronecker PC
Nonlinear KroneckerPCA [train.p3, temporal] All (84)	0.3389212	Cubed Kronecker P
Nonlinear KroneckerPCA [train.p3, temporal] 99.9% (25)	0.3348935	Cubed Kronecker P
Nonlinear KroneckerPCA [train, temporal.p3] All (84)	0.3371075	Cubed Kronecker P
Nonlinear KroneckerPCA [train, temporal.p3] 99.9% (29)	0.3493172	Cubed Kronecker P
PCA Cubed (11)	0.3535546	PCA on cubed data
Raw Cubed (12)	0.3449375	The cubed model.
KernelPCA Lagged 99.997% (300)	0.3465964	Kernel PCA on lag
Cubed Lagged (84)	0.3332918	Normal glmnet on
PCA Cubed Lagged (62)	0.3336829	PCA on Cubed &

[720]:
```
# Spatial Linear:
results.summary[c(1,2),]
# Spatial Nonlinear:
results.summary[c(3,16),]
# Spatio-Temporal Linear:
results.summary[c(6,10),]
# Spatio-Temporal Non-Linear:
results.summary[c(17,12,14,19),]
```

A data.table: 2 × 3

model <chr>	wmape <dbl>	notes <chr>
Raw Spatial (4)	0.3998878	The model for base performance. GLMNET(cv with lambda.
PCA Spatial (3)	0.4134921	PCA on spatial data. Took 3 PCs and saw a small but expec

A data.table: 2 × 3

model <chr>	wmape <dbl>	notes <chr>
KernelPCA Spatial All (34)	0.3361963	Kernel PCA on spatial data. PCA fitted on rando
Raw Cubed (12)	0.3449375	The cubed model. Basis for nonlinear relationship

A data.table: 2 × 3

model <chr>	wmape <dbl>	notes <chr>
PCA Lagged 90% (10)	0.3777897	PCA on lagged data. 10 PCs are
KroneckerPCA [train, temporal] 99.9% (12)	0.3777050	Basis Kronecker PCA with less di

A data.table: 4 × 3

model <chr>	wmape <dbl>	notes <chr>
KernelPCA Lagged 99.997% (300)	0.3465964	Kernel PCA on lagg
Nonlinear KroneckerPCA [train.p3, temporal] 99.9% (25)	0.3348935	Cubed Kronecker PC
Nonlinear KroneckerPCA [train, temporal.p3] 99.9% (29)	0.3493172	Cubed Kronecker PC
PCA Cubed Lagged (62)	0.3336829	PCA on Cubed & La

[719]:
```
d1
d2
d3
d4
```

A data.table: 2 × 3

model <chr>	wmape <dbl>	notes <chr>
Raw Spatial (4)	0.3998878	The model for base performance. GLMNET(cv with lambda.
PCA Spatial (3)	0.4134921	PCA on spatial data. Took 3 PCs and saw a small but expec

A data.table: 2 × 3

model <chr>	wmape <dbl>	notes <chr>
KernelPCA Spatial All (34)	0.3361963	Kernel PCA on spatial data. PCA fitted on rando
Raw Cubed (12)	0.3449375	The cubed model. Basis for nonlinear relationship

A data.table: 2 × 3

model <chr>	wmape <dbl>	notes <chr>
PCA Lagged 90% (10)	0.3777897	PCA on lagged data. 10 PCs are
KroneckerPCA [train, temporal] 99.9% (12)	0.3777050	Basis Kronecker PCA with less di

A data.table: 4 × 3

model <chr>	wmape <dbl>	notes <chr>
KernelPCA Lagged 99.997% (300)	0.3465964	Kernel PCA on lagg
Nonlinear KroneckerPCA [train.p3, temporal] 99.9% (25)	0.3348935	Cubed Kronecker PC
Nonlinear KroneckerPCA [train, temporal.p3] 99.9% (29)	0.3493172	Cubed Kronecker PC
PCA Cubed Lagged (62)	0.3336829	PCA on Cubed & La

11 5 Ocak After Meeting:

[774]:
```
crea
```

[775]:
```
pred.poly <- customGlm(traindata = train.poly,
                       trainres = train[,7],
                       testdata = test.poly,
                       testres = test[,7])
```

[776]:
```
print(wmape(test[,7],pred.poly))
```

```
[1] 0.3332809
```

[784]:
```
train.2lag <- createLagged(data = train,lags = c(-2:2),other.head =␣
↪c(1,2),other.tail = 7)
```

[819]:
```
# degree 3 is assumed for now
ccPoly <- function(x=as.matrix(train.lag[,3:30])){
    r = nrow(x)
    k = ncol(x)
    res = data.table(matrix(rep(1,r),r,1))
    # ASSUMES DEGREE=3
    for(i1 in c(1:k)){
        for(i2 in c(i1:k)){
            message(paste(i1, i2, sep='-'))
            for(i3 in c(i2:k)){
                t = x[,i1] * x[,i2] * x[,i3]
                res = cbind(res, t)
                colnames(res)[ncol(res)] = paste(i1, i2, i3, sep='.')
            }
        }
    }
}
```

[ ]:
```

```

[833]:
```
x = train.lag[,3:30]
r = nrow(x)
k = ncol(x)
i1 = 3
res <- data.table(matrix(rep(1,r),r,1))
for(i2 in c(i1:k)){
    for(i3 in c(i2:k)){
        t = x[,..i1] * x[,..i2] * x[,..i3]
        res = cbind(res, t)
        colnames(res)[ncol(res)] = paste(i1, i2, i3, sep='.')
    }
}
```

[834]:
```
train.lag.poly <- cbind(train.lag.poly, res[,-1])
```

[836]:
```
dim(train.lag.poly)
```

1. 26298 2. 1135

[ ]:
```

```

[ ]:
```
zztest.lag.poly  <- polym(as.matrix(test.lag[,3:30]), degree = 3, raw = TRUE,␣
↪simple=TRUE)
```

[ ]:
```
pred.lag.poly <- customGlm(traindata = train.lag.poly,
                           trainres = train.lag[,31],
                           testdata = test.lag.poly,
                           testres = test.lag[,31])
```

```
print(wmape(test.lag[,7],pred.lag.poly))
```

12 Kernel PCA yapayim ama cv'deki hatalara da bakayim

A quick test on others, too. For comparing train/test errors on all

[894]:
```
head(train.p3.lag)
```

A data.table: 6 × 87

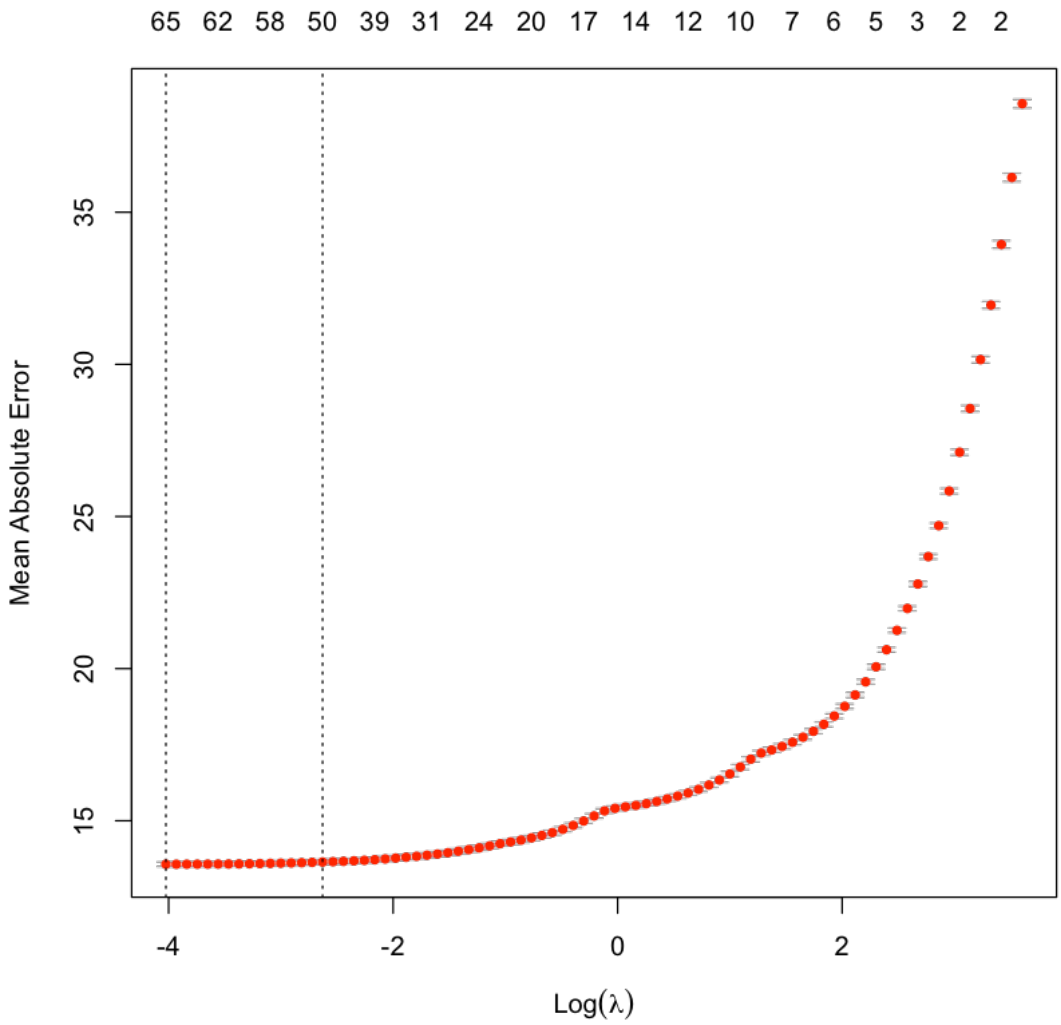
date <date>	hour <int>	ne_lag3 <dbl>	nw_lag3 <dbl>	se_lag3 <dbl>	sw_lag3 <dbl>	ne_p2_lag3 <dbl>	nw <
2017-07-01	3	0.0749545	0.00578771	-0.13545788	-0.1367260	0.005618177	3.
2017-07-01	4	0.2373761	0.21002174	0.08370014	0.1090849	0.056347390	4.
2017-07-01	5	0.3998024	0.41760091	0.30487559	0.3590809	0.159841962	1.
2017-07-01	6	0.5622325	0.62758994	0.52745232	0.6118405	0.316105365	3.
2017-07-01	7	0.6464433	0.74679758	0.57877085	0.6785532	0.417888977	5.
2017-07-01	8	0.7361743	0.87312977	0.64123292	0.7508598	0.541952628	7.

[896]:
```
fitx = cv.glmnet(as.matrix(train.p3.lag[,3:86]), as.matrix(train.p3.lag[,87]),␣
 ↪type.measure = 'mae')
fitx
plot(fitx)
```

```
Call:  cv.glmnet(x = as.matrix(train.p3.lag[, 3:86]), y = as.matrix(train.p3.
 ↪lag[,      87]), type.measure = "mae")

Measure: Mean Absolute Error

     Lambda Measure      SE Nonzero
min 0.01788   13.57 0.07502      65
1se 0.07218   13.64 0.08174      50
```



[900]:
```
pxm = predict(fitx, as.matrix(test.p3.lag[,3:86]), s = "lambda.min")
px1 = predict(fitx, as.matrix(test.p3.lag[,3:86]), s = "lambda.1se")
```

[902]:
```
mae(as.matrix(test.p3.lag[,87]), pxm)
mae(as.matrix(test.p3.lag[,87]), px1)
```

14.4585981566465

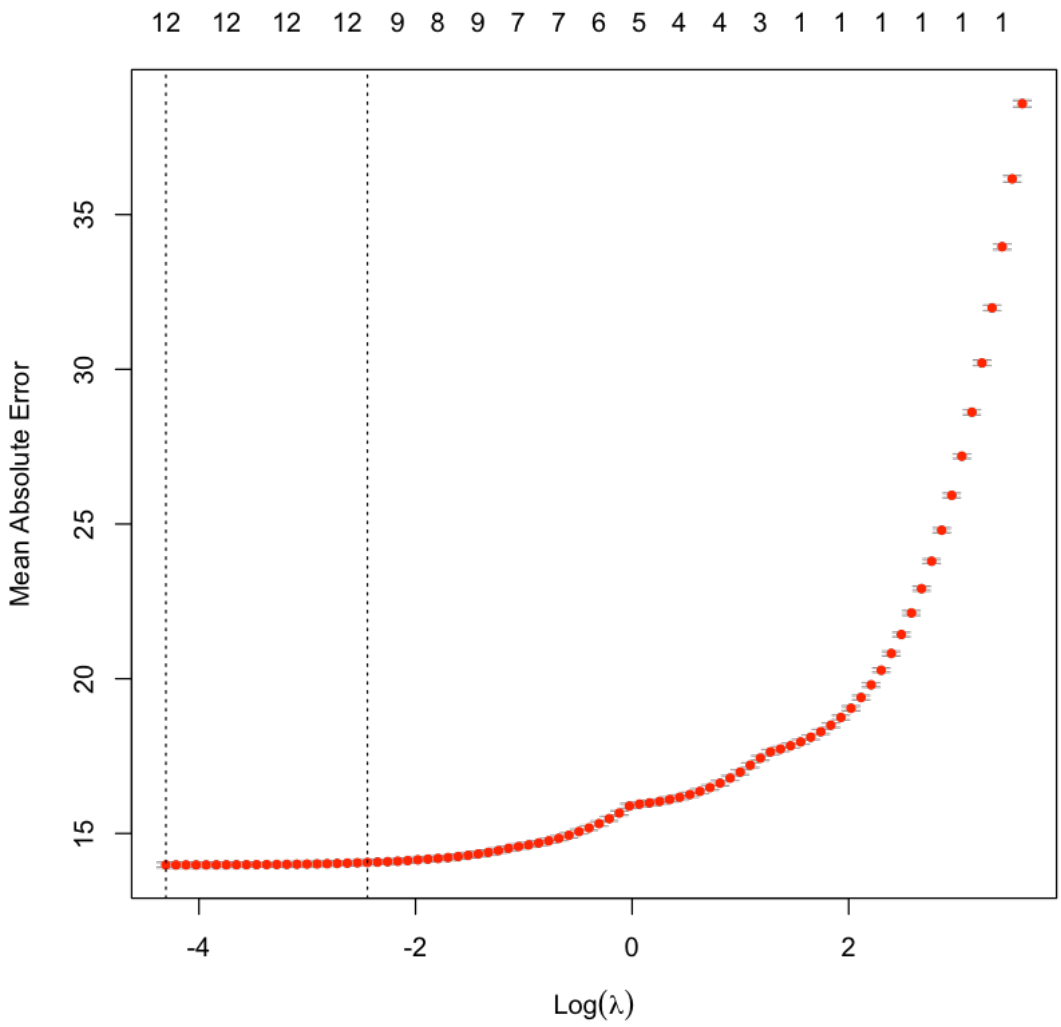
14.3791852821125

[904]:
```
fitx = cv.glmnet(as.matrix(train.p3[,3:14]), as.matrix(train.p3[,15]), type.
↪measure = 'mae')
fitx
plot(fitx)
```

```
Call:  cv.glmnet(x = as.matrix(train.p3[, 3:14]), y = as.matrix(train.p3[,
 ↪15]), type.measure = "mae")

Measure: Mean Absolute Error

     Lambda Measure      SE Nonzero
min 0.01350   13.98 0.08269      12
1se 0.08679   14.06 0.07801      11
```



[907]:
```
pxm = predict(fitx, as.matrix(test.p3[,3:14]), s = "lambda.min")
px1 = predict(fitx, as.matrix(test.p3[,3:14]), s = "lambda.1se")
mae(as.matrix(test.p3[,15]), pxm)
```

```
mae(as.matrix(test.p3[,15]), px1)
```

14.9700910881897

14.9653549778795

[ ]:

[ ]:

12.1 starts:

[ ]:
```
# train.raw.kernelpca.part.full
# test.raw.kernelpca.part.full
```

[838]:
```
set.seed(42)
fit1 = cv.glmnet(as.matrix(train.raw.kernelpca.part.full),
                 as.matrix(train[,7]),
                 type.measure = 'mae')
```

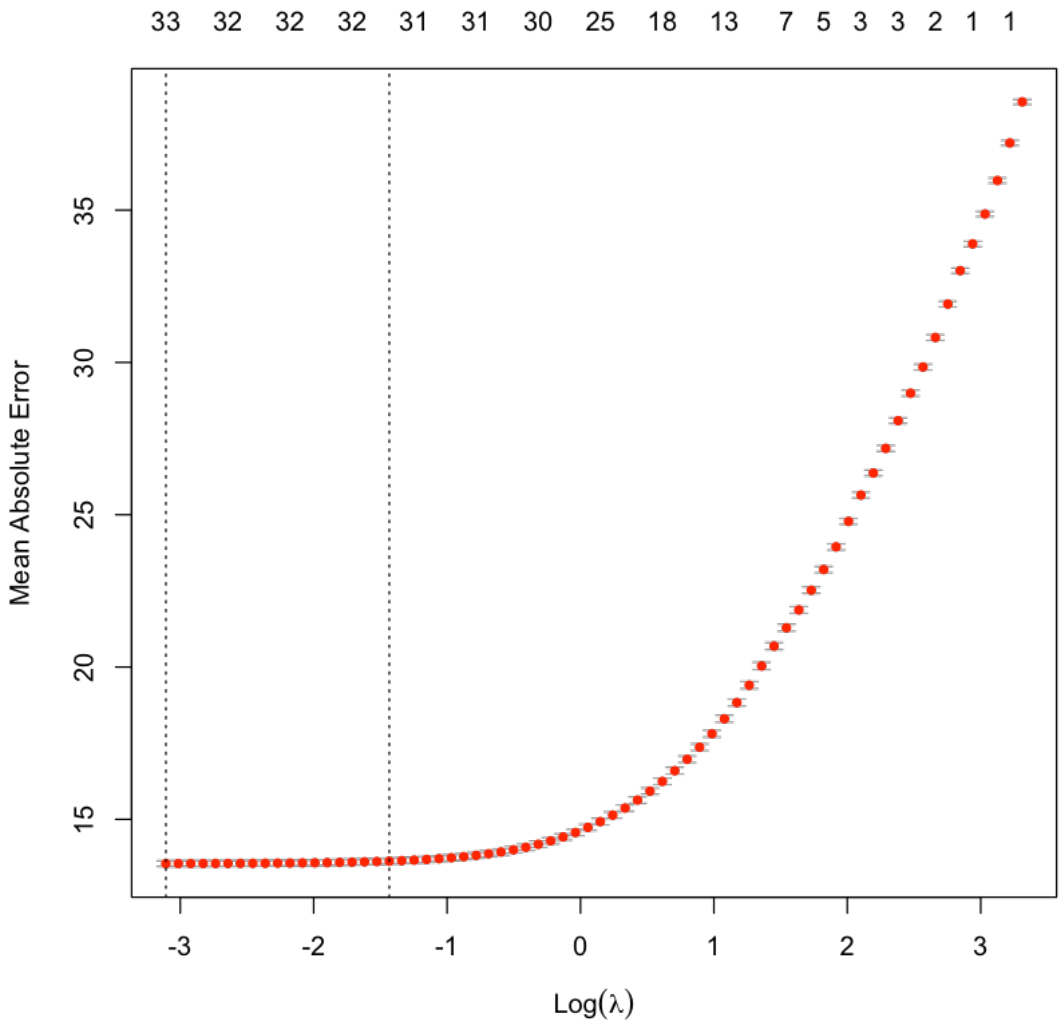
[839]:
```
fit1
```

```
Call:  cv.glmnet(x = as.matrix(train.raw.kernelpca.part.full), y = as.
↪matrix(train[,      7]), type.measure = "mae")

Measure: Mean Absolute Error

     Lambda Measure      SE Nonzero
min 0.04471   13.55 0.09563      33
1se 0.23860   13.63 0.09652      31
```

[849]:
```
plot(fit1)
```



[860]:
```
pred.raw.kernelpca.part.full <- predict(fit1, test.raw.kernelpca.part.full, s =␣
↪'lambda.1se')
```

[861]:
```
mae(as.matrix(test[,7]), as.matrix(pred.raw.kernelpca.part.full))
```

14.4606474273114

[859]:
```
mae(as.matrix(test[,7]), as.matrix(pred.raw.kernelpca.part.full))
```

14.5907268515201

[846]:
```
print(wmape(observed = test[,7], predicted = pred.raw.kernelpca.part.full))
```

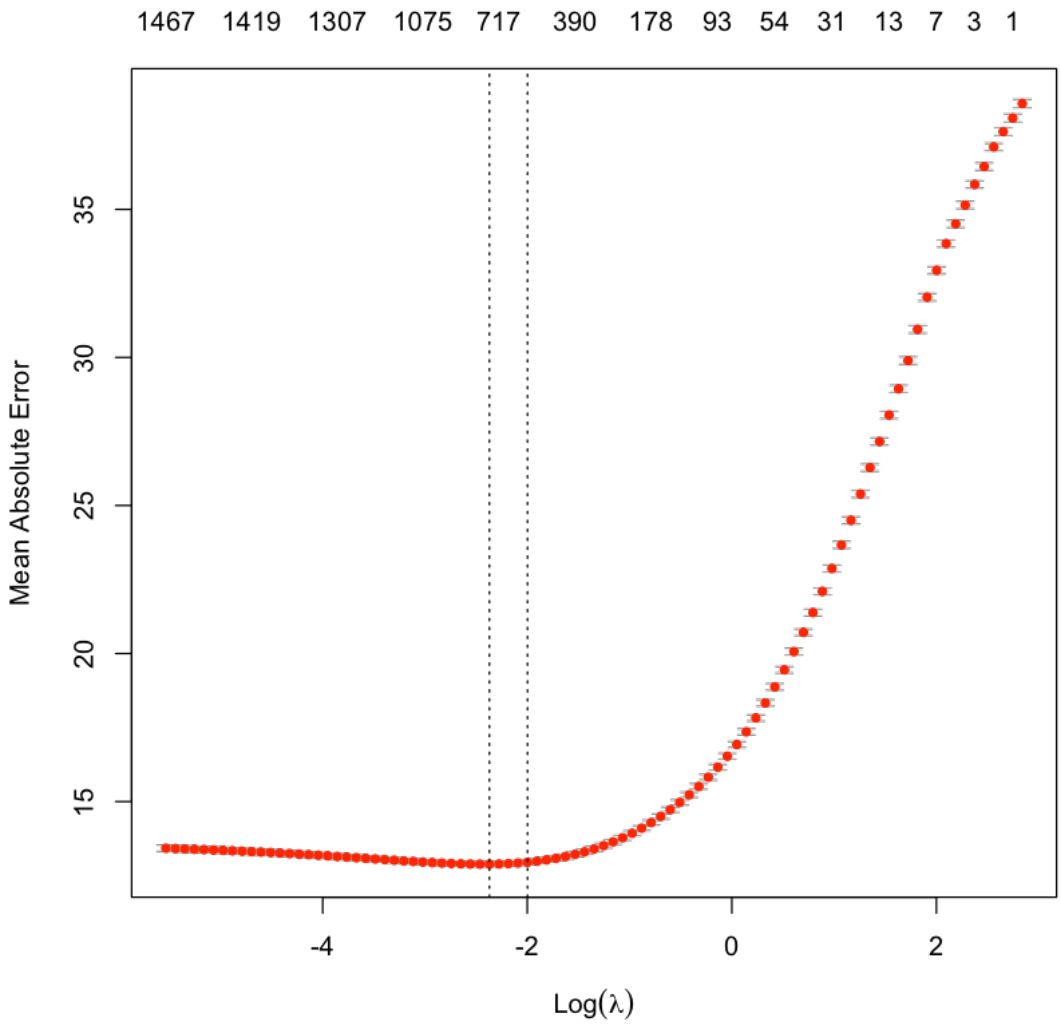
```
[1] 0.333199
```

13 Lag Kernel PCA

[863]:
```
# train.lag.kernelpca.part.full
```

[866]:
```
fit2 <- cv.glmnet(x = as.matrix(train.lag.kernelpca.part.full),
                  y = as.matrix(train.lag[,31]),
                  type.measure = 'mae')
```

[867]:
```
plot(fit2)
```



[876]:
```
fit2
```

```
Call:  cv.glmnet(x = as.matrix(train.lag.kernelpca.part.full), y = as.
 ↪matrix(train.lag[,      31]), type.measure = "mae")

Measure: Mean Absolute Error

     Lambda Measure      SE Nonzero
min 0.09348   12.89 0.07966     768
1se 0.13562   12.95 0.07978     584
```

[868]:
```
pred.lag.kernelpca.part.full.m <- predict(fit2, test.lag.kernelpca.part.full, s␣
 ↪= 'lambda.min')
pred.lag.kernelpca.part.full.1 <- predict(fit2, test.lag.kernelpca.part.full, s␣
 ↪= 'lambda.1se')
mae(as.matrix(test.lag[,31]), as.matrix(pred.lag.kernelpca.part.full.m))
mae(as.matrix(test.lag[,31]), as.matrix(pred.lag.kernelpca.part.full.1))
```

16.9731255963109

15.8620791511071

[869]:
```
print(wmape(as.matrix(test.lag[,31]), as.matrix(pred.lag.kernelpca.part.full.
 ↪m)))
print(wmape(as.matrix(test.lag[,31]), as.matrix(pred.lag.kernelpca.part.full.
 ↪1)))
```

```
[1] 0.3912554
[1] 0.3656441
```

14 hyperparameters of cv.glmnet

[878]:
```
fit2x <- cv.glmnet(x = as.matrix(train.lag.kernelpca.part.full),
                   y = as.matrix(train.lag[,31]),
                   type.measure = 'mae',
                   nfolds = 20)
```

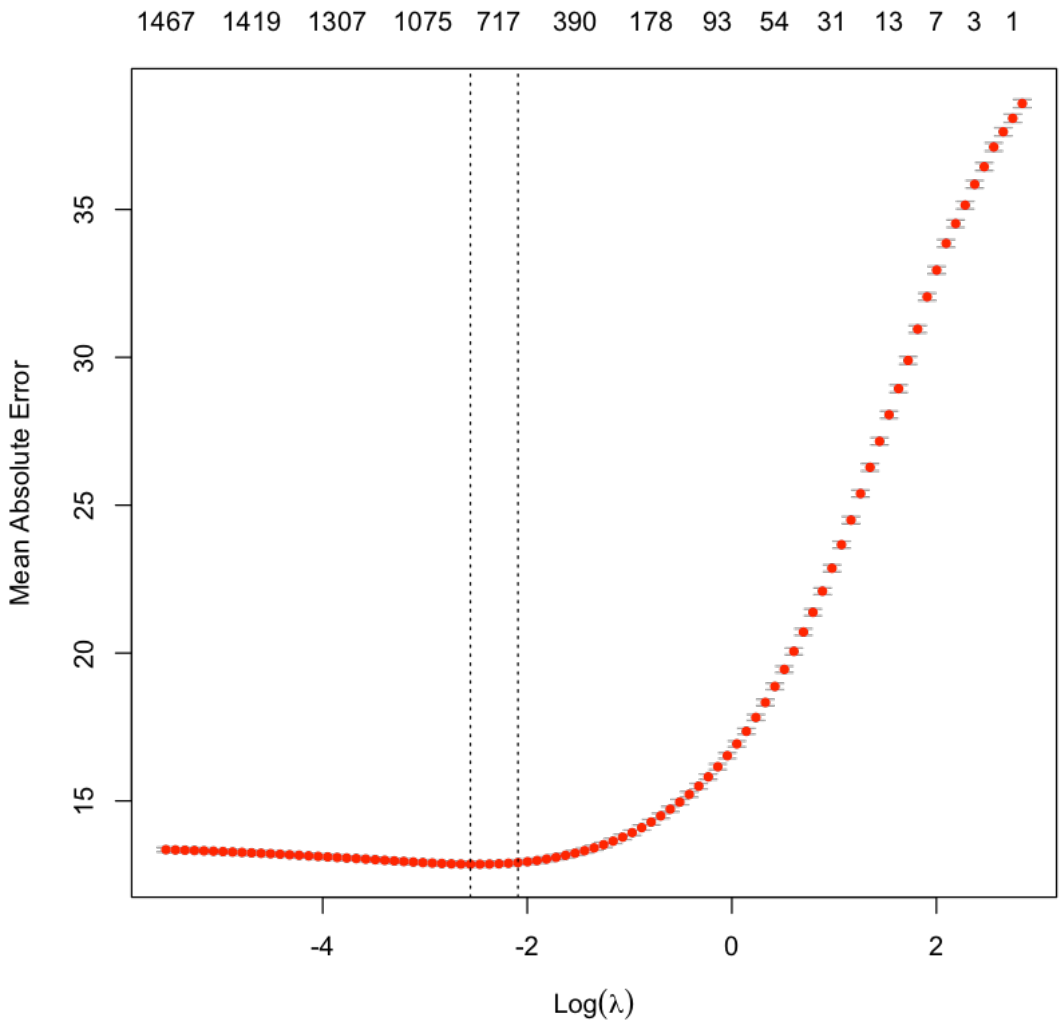
[879]:
```
fit2x
```

```
Call:  cv.glmnet(x = as.matrix(train.lag.kernelpca.part.full), y = as.
 ↪matrix(train.lag[,      31]), type.measure = "mae", nfolds = 20)

Measure: Mean Absolute Error

     Lambda Measure      SE Nonzero
min 0.07761   12.86 0.07815     858
1se 0.12357   12.91 0.07821     620
```

[880]:
```
plot(fit2x)
```



[881]:
```
pred.lag.kernelpca.part.full.mx <- predict(fit2x, test.lag.kernelpca.part.full,␣
↪s = 'lambda.min')
pred.lag.kernelpca.part.full.1x <- predict(fit2x, test.lag.kernelpca.part.full,␣
↪s = 'lambda.1se')
mae(as.matrix(test.lag[,31]), as.matrix(pred.lag.kernelpca.part.full.mx))
mae(as.matrix(test.lag[,31]), as.matrix(pred.lag.kernelpca.part.full.1x))
```

17.6108552979001

16.1113075334291

[885]:
```
fit3 <- cv.glmnet(x = as.matrix(train.lag.kernelpca.part.full[,1:500]),
                  y = as.matrix(train.lag[,31]),
```

```
                    type.measure = 'mae')
```

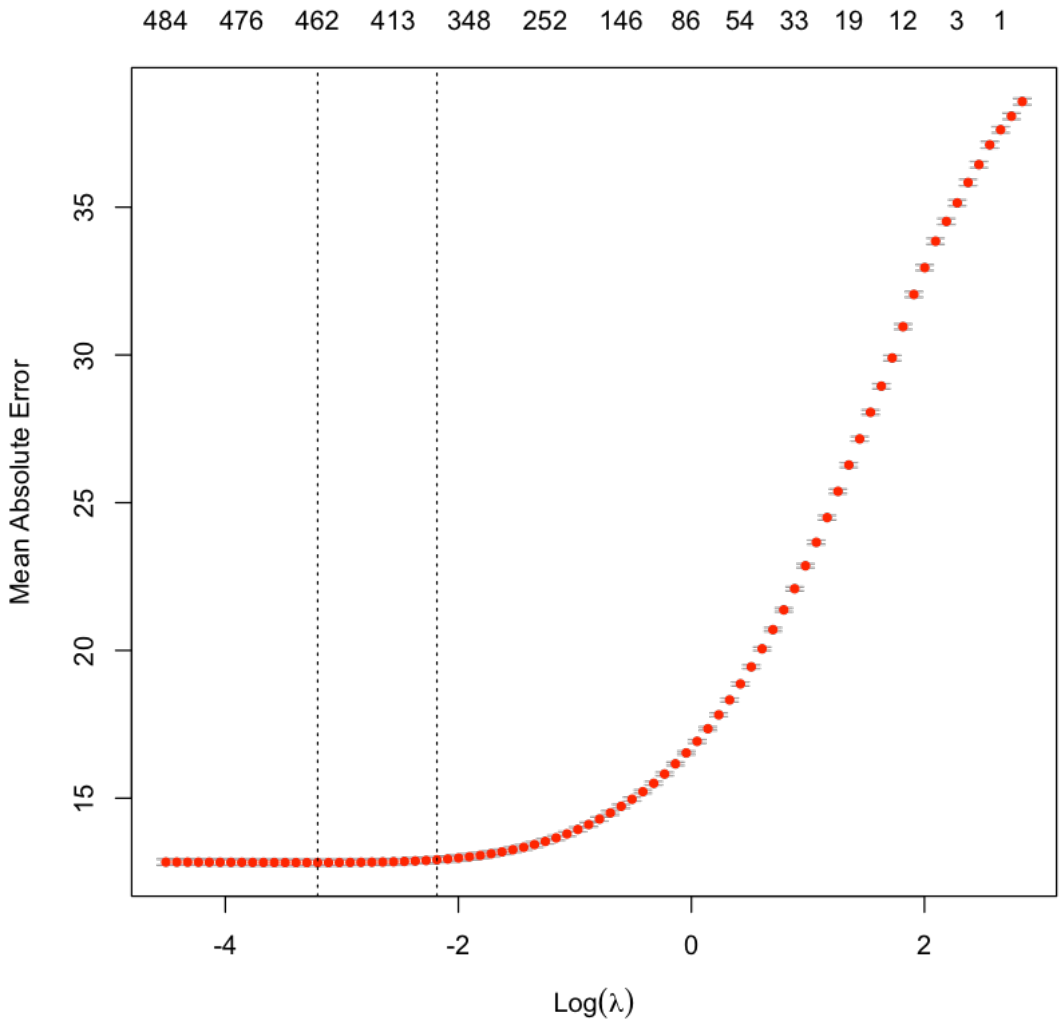
[886]:
```
fit3
```

```
Call:  cv.glmnet(x = as.matrix(train.lag.kernelpca.part.full[, 1:500]),      y =␣
↪as.matrix(train.lag[, 31]), type.measure = "mae")

Measure: Mean Absolute Error

     Lambda Measure      SE Nonzero
min 0.04046   12.82 0.09910     462
1se 0.11259   12.91 0.09579     373
```

[887]:
```
plot(fit3)
```



[889]:
```
pred.lag.kernelpca.part.full.m3 <- predict(fit3, test.lag.kernelpca.part.
↪full[,1:500], s = 'lambda.min')
pred.lag.kernelpca.part.full.13 <- predict(fit3, test.lag.kernelpca.part.
↪full[,1:500], s = 'lambda.1se')
mae(as.matrix(test.lag[,31]), as.matrix(pred.lag.kernelpca.part.full.m3))
mae(as.matrix(test.lag[,31]), as.matrix(pred.lag.kernelpca.part.full.13))
```

15.241703205679

14.7304831394317

[890]:
```
fit4 <- cv.glmnet(x = as.matrix(train.lag.kernelpca.part.full[,1:100]),
                  y = as.matrix(train.lag[,31]),
                  type.measure = 'mae')
```

[892]:
```
fit4
```

```
Call:  cv.glmnet(x = as.matrix(train.lag.kernelpca.part.full[, 1:100]),      y =␣
↪as.matrix(train.lag[, 31]), type.measure = "mae")

Measure: Mean Absolute Error

     Lambda Measure      SE Nonzero
min 0.02541   15.49 0.09916     100
1se 0.16335   15.58 0.09927      94
```

[891]:
```
pred.lag.kernelpca.part.full.m4 <- predict(fit4, test.lag.kernelpca.part.
↪full[,1:100], s = 'lambda.min')
pred.lag.kernelpca.part.full.14 <- predict(fit4, test.lag.kernelpca.part.
↪full[,1:100], s = 'lambda.1se')
mae(as.matrix(test.lag[,31]), as.matrix(pred.lag.kernelpca.part.full.m4))
mae(as.matrix(test.lag[,31]), as.matrix(pred.lag.kernelpca.part.full.14))
```

16.1416850471338

16.0722717977405

[912]:
```
k = load(file = "polydata.rda")
```

[914]:
```
dim(train.lag.poly)
```

1. 26298 2. 4061

[908]:
```
res = 0
for(i in c(1:28)){
   res = res + (i*(i+1)/2)
```

```
}
res
```

4060

[909]:
```
fit1
```

```
Call:  cv.glmnet(x = as.matrix(train.raw.kernelpca.part.full), y = as.
 ↪matrix(train[,      7]), type.measure = "mae")

Measure: Mean Absolute Error

     Lambda Measure      SE Nonzero
min 0.04471   13.55 0.09563      33
1se 0.23860   13.63 0.09652      31
```

[915]:
```
is.numeric(5)
```

TRUE

[941]:
```
values <- list(
  A = c(9, 1, 8, 4, 3),
  B = c(7,  3,  4,  4,  1,  1, 10,  8)
)

lapply(values, function(x, x-2))
```

```
Error in parse(text = x, srcfile = src):      'x'        (6    )
Traceback:
```

15 GLM (not net) only

[942]:
```
head(train.p3)
```

A data.table: 6 × 15

date <date>	hour <int>	ne <dbl>	nw <dbl>	se <dbl>	sw <dbl>	ne_p2 <dbl>	nw <dbl>
2017-07-01	0	0.0749545	0.00578771	-0.13545788	-0.1367260	0.005618177	3.
2017-07-01	1	0.2373761	0.21002174	0.08370014	0.1090849	0.056347390	4.
2017-07-01	2	0.3998024	0.41760091	0.30487559	0.3590809	0.159841962	1.
2017-07-01	3	0.5622325	0.62758994	0.52745232	0.6118405	0.316105365	3.
2017-07-01	4	0.6464433	0.74679758	0.57877085	0.6785532	0.417888977	5.
2017-07-01	5	0.7361743	0.87312977	0.64123292	0.7508598	0.541952628	7.

[943]:
```
fit.glm.p3 <- glm(formula = production~., data = train.p3[,3:15])
```

[945]:
```
mean(abs(fit.glm.p3$residuals))
```

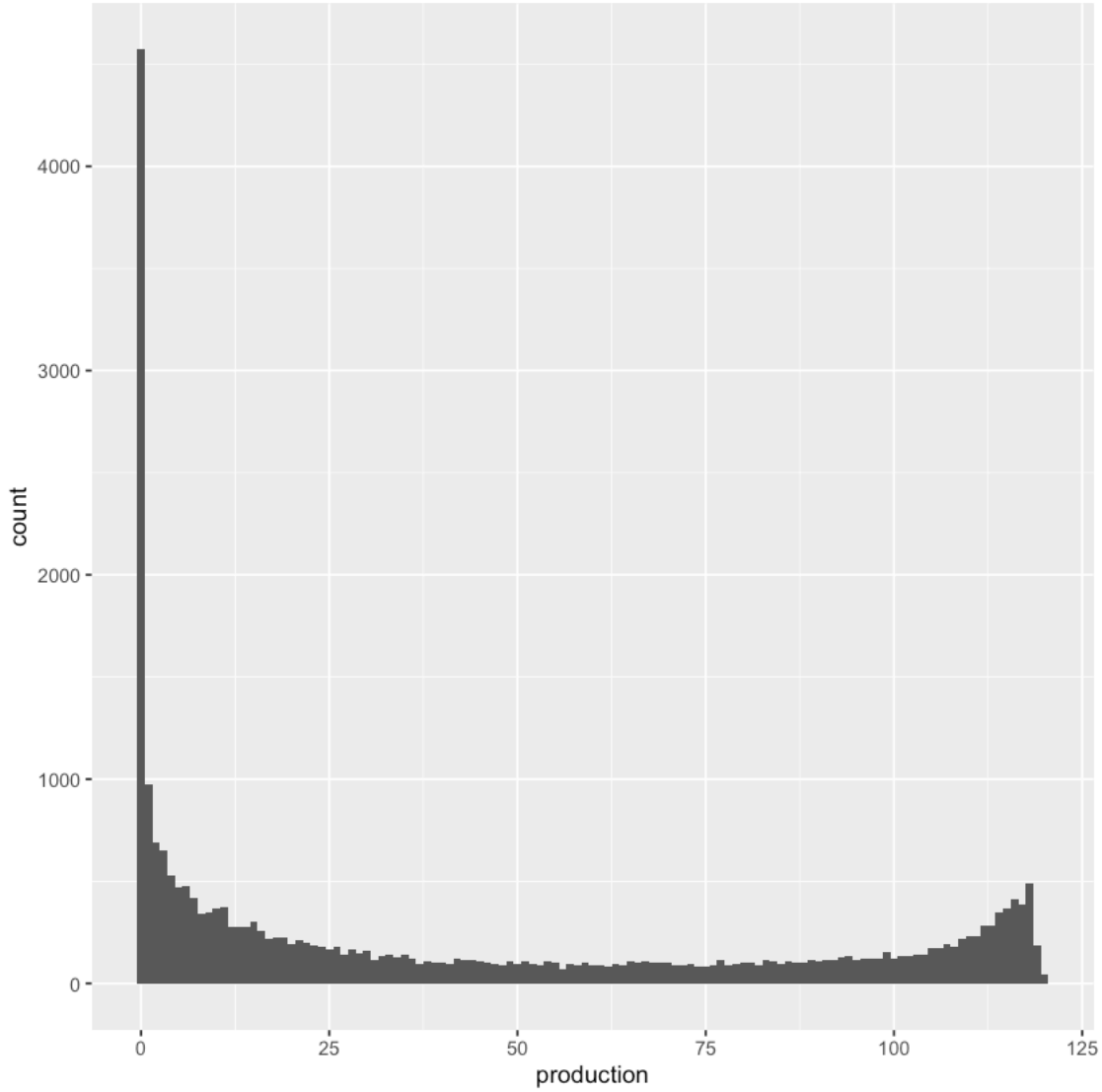
13.964886175408

[946]:
```
pred.glm.p3 <- predict(fit.glm.p3, test.p3[,3:15])
```

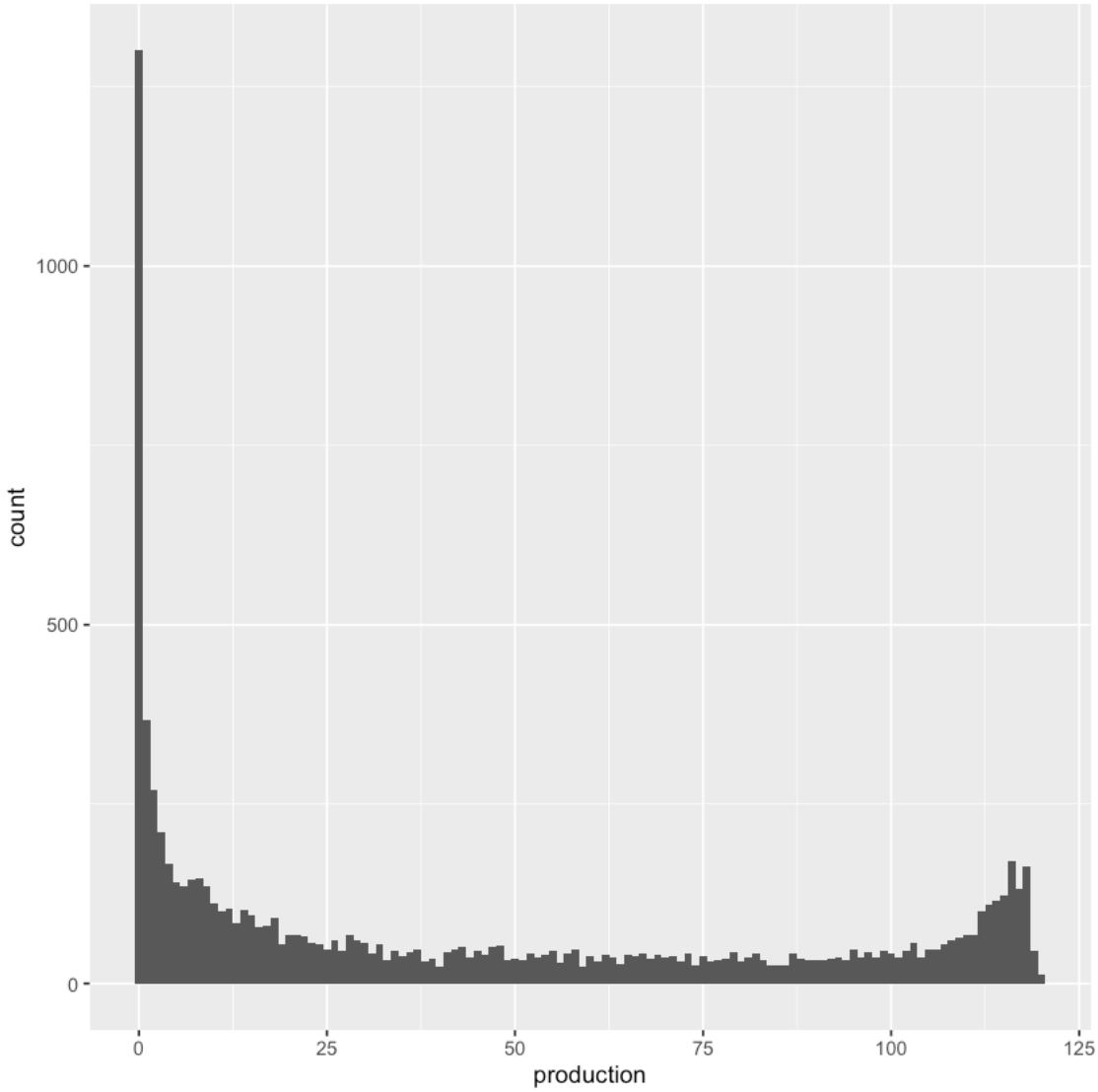
[949]:
```
mean(abs(test.p3$production - pred.glm.p3))
```

15.0084313741627

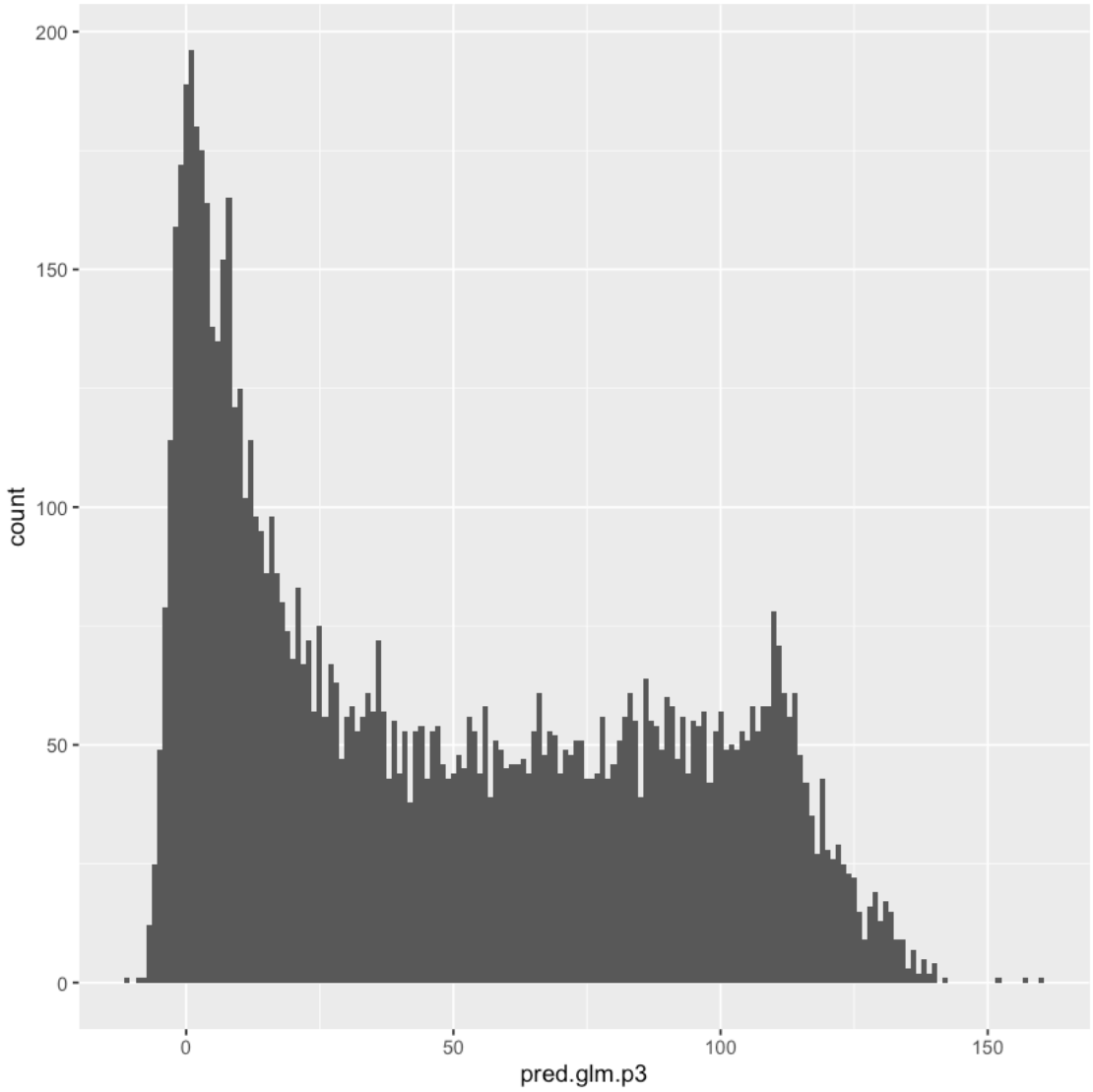
[953]:
```
ggplot(train) + geom_histogram(aes(production),binwidth = 1)
```



[954]:
```
ggplot(test) + geom_histogram(aes(production),binwidth = 1)
```



[957]:
```
ggplot() + geom_histogram(aes(pred.glm.p3), binwidth = 1)
```



[8]:
```
head(train)
```

A data.table: 6 × 7

	date <date>	hour <int>	ne <dbl>	nw <dbl>	se <dbl>	sw <dbl>	production <dbl>
1	2017-07-01	0	6.540107	7.741893	4.760000	6.249544	57
2	2017-07-01	1	7.166667	8.727033	5.451166	7.302665	77
3	2017-07-01	2	7.793246	9.728310	6.148695	8.373715	55
4	2017-07-01	3	8.419840	10.741210	6.850642	9.456606	35
5	2017-07-01	4	8.744693	11.316219	7.012487	9.742422	35
6	2017-07-01	5	9.090841	11.925593	7.209476	10.052203	61

[9]:
```
cov(train[1:100,3:6])
```

A matrix: 4 × 4 of type dbl

	ne	nw	se	sw
ne	15.04719	15.49542	12.89695	14.08022
nw	15.49542	17.54608	13.06738	15.42398
se	12.89695	13.06738	11.49754	12.25437
sw	14.08022	15.42398	12.25437	14.11792

[ ]: