R Codes to paper

**Spatial bootstrapped microeconometrics:**

**forecasting for out-of-sample geo-locations in big data**

**Katarzyna Kopczewska**

[**kkopczewska@wne.uw.edu.pl**](mailto:kkopczewska@wne.uw.edu.pl)

**Faculty of Economic Sciences, University of Warsaw, Poland**

This material presents two sections with R codes, tested in R 4.2.1 version:

* codes to run an algorithm itself to estimate and select the best bootstrap model – they work quickly and efficient with reasonable parameters (2000 observations, 50 iterations, 100 out-of-sample points); they use the dataset of point data and shapefile map available at Github <https://github.com/kkopczewska/bootstrapping>
* codes to replicate all figures from the paper – they are not necessary to estimate your best bootstrap model, but are to show transparently research methods used in this paper; it takes time to run all estimations; thus, one can download the files with the results of simulations from <https://app.sugarsync.com/iris/wf/D1836703_09668593_7523736> and read them to R, what significantly increases replication time.

The R codes were written in two standards: *sp* class and *sf* class. The major difference between those classes (in this set of codes) lays in reading the contour maps and handling spatial objects, running tessellation and overlying points on tessellation tiles. The rest is the same.

This set of codes includes five major sections:

1. sp class in brief – to use the bootstrap procedure for own data (as long as sp class is supported)
2. sf class in brief – to use the bootstrap procedure for own data
3. sp class in details – to understand all steps which were done in writing the paper
4. sf class in details – to understand all steps which were done in writing the paper
5. codes to replicate all figures from the paper – for the reliability of this research

Spis treści

[# sp class – model in brief - R codes to run your own bootstrap model 4](#_Toc115360137)

[# starter – reading and preparing data 4](#_Toc115360138)

[# bootstrap simulation to find the best model 4](#_Toc115360139)

[# selection of the best model with PAM 6](#_Toc115360140)

[# validation for out-of-sample 6](#_Toc115360141)

[# sf class – model in brief - R codes to run your own bootstrap model 8](#_Toc115360142)

[# starter – reading and preparing data 8](#_Toc115360143)

[# bootstrap simulation to find the best model 8](#_Toc115360144)

[# selection of the best model with PAM 10](#_Toc115360145)

[# validation for out-of-sample 10](#_Toc115360146)

[# sp class – full analytics - R codes to run bootstrap models as in paper 12](#_Toc115360147)

[# starter – reading and preparing data 12](#_Toc115360148)

[# estimation of the full model (for comparison) 16](#_Toc115360149)

[# bootstrap simulation to find the best model 17](#_Toc115360150)

[# selection of the best model with PAM 21](#_Toc115360151)

[# validation for out-of-sample 22](#_Toc115360152)

[# OLS predictions 22](#_Toc115360153)

[# SEM predictions 23](#_Toc115360154)

[# SAR predictions 25](#_Toc115360155)

[# SDM predictions 27](#_Toc115360156)

[# all RAMSE together – for models and for forecasts 28](#_Toc115360157)

[# full sample SDM - to get RAMSE of forecast from the full-sample model 28](#_Toc115360158)

[# sf class R codes to run bootstrap model 31](#_Toc115360159)

[# starter – reading and preparing data 31](#_Toc115360160)

[# estimation of the full model (for comparison) 35](#_Toc115360161)

[# bootstrap simulation to find the best model 36](#_Toc115360162)

[# selection of the best model with PAM 40](#_Toc115360163)

[# validation for out-of-sample 41](#_Toc115360164)

[# OLS predictions 41](#_Toc115360165)

[# SEM predictions 42](#_Toc115360166)

[# SAR predictions 43](#_Toc115360167)

[# SDM predictions 45](#_Toc115360168)

[# all RAMSE together – for models and for forecasts 46](#_Toc115360169)

[# full sample SDM - to get RAMSE of forecast from the full-sample model 46](#_Toc115360170)

[# FIGURES FROM THE TEXT 49](#_Toc115360171)

[# Figure 1 49](#_Toc115360172)

[# Figure 2a 50](#_Toc115360173)

[# Figure 2b 52](#_Toc115360174)

[# Figure 3 53](#_Toc115360175)

[# Fig.3a 56](#_Toc115360176)

[# Fig.3b 56](#_Toc115360177)

[# Figure 4 58](#_Toc115360178)

[# Figure 5 61](#_Toc115360179)

[# Figure 6 62](#_Toc115360180)

[# Fig. 6a 63](#_Toc115360181)

[# Fig. 6b 65](#_Toc115360182)

[# Fig.6c 66](#_Toc115360183)

[# Fig. 6d 66](#_Toc115360184)

[# Figure 7 68](#_Toc115360185)

[# Fig.7a 73](#_Toc115360186)

[# Fig.7b 74](#_Toc115360187)

[# Fig.7c 74](#_Toc115360188)

[# Fig.7d 75](#_Toc115360189)

[# Figure 8 77](#_Toc115360190)

[# Figure 9 79](#_Toc115360191)

[# Fig.9a 81](#_Toc115360192)

[# Fig.9b 82](#_Toc115360193)

[# Figure 10 84](#_Toc115360194)

# 

# # sp class – model in brief - R codes to run your own bootstrap model

**# please change lines in RED**

## # starter – reading and preparing data

**# loading packages**

set.seed(357)

library(spdep) # nb2listw(), make.sym.nb(), knn2nb(), knearneigh(), moran.test() – W spatial weights matrix

library(rgdal) # readOGR() – reading spatial data

library(maptools) # for sp conversion – handling spatial data

library(sp) # spTransform(), SpatialPoints(), CRS(), proj4string(), over(), - spatial data operations

library(doBy) # doBy() - sorting

library(spatialreg) # errorsarlm(), lagsarlm(), predict.sarlm() – spatial econometrics

library(sampling) # for strata() – stratified sampling

library(cluster) # for pam() - clustering

library(spatstat) # for as.owin(), dirichlet(), ppp() - tesselation

**# Setting path of Working Directory**

#setwd("D:/My all/dataset") # change it according to your settings

**# reading contour map with rgdal:: here NTS2 regions of Poland**

reg<-readOGR(".", "wojewodztwa") # 16 regions

reg<- spTransform(reg, CRS("+proj=longlat +datum=NAD83")) # changing the projections into NAD83

reg.merc<-spTransform(reg, CRS("+proj=merc +datum=NAD83")) # planar projections

reg.sel<-reg[reg@data$jpt\_nazwa\_=="lubelskie",] # one region only

reg.sel.merc<-reg.merc[reg.merc@data$jpt\_nazwa\_=="lubelskie",] # one region only

**# reading geolocated point data – ca.37.000 firms in Lubelskie region**

dane<-read.csv("point\_data.csv", sep=",", dec=".", header=TRUE)

summary(dane)

names(dane)

head(dane)

dane1.in<-dane[1:30000,] # split your data for training data

dane1.out<-dane[30001:37374,] # and test data

## # bootstrap simulation to find the best model

############################################

**# parameters of simulation – change here your parameters – now very limited just to run quickly the model**

n.col<-10 # number of iterations

n.row<-800 # number of observations in a sample

n.new<-10 # number of new points in the forecast

n.var<-5 # set the number of explanatory variables included in the model equation

n.knn<-5 # set the number of knn for W

n.dep<-27 # which column is dependent y variable

n.x<-58 # which column are x coordinates

n.y<-59 # which column are y coordinates

############################################

**# set your equation**

eq<-roa~empl+prod+constr+serv+dist # model structure

############################################

crds.in<-dane1.in[,n.x:n.y] # separate the locations (x,y) of points used in training

crds.out<-dane1.out[,n.x:n.y] # separate the locations (x,y) of points used in preditions

###############################################

**# cluster your data – xy coordinates - to get irregular shapes for sampling**

groups<-kmeans(crds.in, n.row/100)

dane1.in$kmean<-groups$cluster

###############################################

**# selector selects the rows of observations for every iteration and saves in the matrix**

**# later it will be used to recover, on which data the best model was estimated**

**# strata() from sampling:: package samples observations from groups (clusters)**

**# method="srswor" stands for simple random sampling without replacement (in a given column)**

selector<-matrix(0, nrow=n.row, ncol=n.col)

for(i in 1:n.col){

vec<-sample(1:dim(dane1.in)[1], n.row, replace=FALSE)

x<-strata(dane1.in, "kmean", size=rep(100, times=n.row/100), method="srswor") # from sampling::

selector[,i]<-x$ID\_unit}

###################################################

**# objects to store the results of iterations**

coef.boot<-matrix(0, nrow=n.col, ncol=n.var\*2+1) # for coefficients

error.boot<-matrix(0, nrow=n.col, ncol=n.var\*2+1) # for standard errors

fitted.boot<-matrix(0, nrow=n.row, ncol=n.col) # for fitted values

y.boot<-matrix(0, nrow=n.row, ncol=n.col) # retriving original values of y

quality.boot<-matrix(0, nrow=n.col, ncol=5) # AIC.ols, AIC.spatial, time.W, time.model, spatial.coef

colnames(quality.boot)<-c("AIC.ols", "AIC.spatial", "time.W", "time.model", "spatial.coef")

**# big loop – estimation of selected model (SEM, SAR, SDM) on the same subsets with time measurement**

for(i in 1:n.col){

danex<-dane1.in[selector[,i],]

**# W matrix**

crds<-as.matrix(crds.in[selector[,i], ])

start.time <- Sys.time()

pkt.k.sym.listw<-nb2listw(make.sym.nb(knn2nb(knearneigh(crds, k=n.knn))))

end.time <- Sys.time()

time.W<- difftime(end.time, start.time, units="secs")

**# choose one model – add hash # for the unwanted**

start.time <- Sys.time()

model<-errorsarlm(eq, data=danex, pkt.k.sym.listw, method="LU")

#model<-lagsarlm(eq, data=danex, pkt.k.sym.listw, method="LU")

#model<-lagsarlm(eq, data=danex, pkt.k.sym.listw, method="LU", type="mixed")

end.time <- Sys.time()

time.model<- difftime(end.time, start.time, units="secs")

**# saving the results into appropriate objects**

coef.boot[i,1:length(model$coefficients)]<-model$coefficients

error.boot[i, 1:length(model$coefficients)]<-model$rest.se

fitted.boot[,i]<-model$fitted.values

y.boot[,i]<-danex[,n.dep] # saving y values for each iteration in roa matrix

quality.boot[i,1]<-model$AIC\_lm.model # AIC.ols

quality.boot[i,2]<-AIC(model) # AIC.spatial

quality.boot[i,3]<-time.W

quality.boot[i,4]<-time.model

quality.boot[i,5]<-ifelse(is.null(model$rho)==TRUE, model$lambda, model$rho)

}

## # selection of the best model with PAM

clust.mod<-pam(cbind(coef.boot[,1:length(model$coefficients)], quality.boot[,5]),1) #cluster::pam(), for n<65.536

summary(clust.mod) # summary of clustering

clust.mod$medoids # coefficients of selected best model

clust.mod$id.med # which iteration (model) is most representative

## # validation for out-of-sample

**# tessellation of observations from the selected best model**

region.owin<-as.owin(reg.sel.merc) # spatstat:: requires planar coordinates, ppp class object

points<-data.frame(x=crds.in[selector[,clust.mod$id.med],1], y=crds.in[selector[,clust.mod$id.med],2])

points.sp<-SpatialPoints(points) # new points in sp class - spherical

proj4string(points.sp)<-CRS("+proj=longlat +datum=NAD83") # spherical projections

points.sp<-spTransform(points.sp, CRS("+proj=merc +datum=NAD83")) # planar projections

region.ppp<-ppp(x=points.sp@coords[,1], y=points.sp@coords[,2], window=region.owin) # points of ppp class

region.tes<-dirichlet(region.ppp) # Dirichlet / Voronoi tesselation

tes.poly<-as(region.tes, "SpatialPolygons") # converting tessellation into sp

proj4string(tes.poly)<-CRS("+proj=merc +datum=NAD83") # giving projections

tes.poly<-spTransform(tes.poly, CRS("+proj=merc +datum=NAD83")) # planar projections

**# restoring the medoid model – choose proper specification**

dane.x<-dane1.in[selector[,clust.mod$id.med],] # data which were used in estimation of the best model

crds.x<-as.matrix(points) # we create W

pkt.k.sym.listw<-nb2listw(make.sym.nb(knn2nb(knearneigh(crds.x, k=n.knn))))

RAMSE.best<-(sum((y.boot[ , clust.mod$id.med] - fitted.boot[ , clust.mod$id.med])^2)/n.row)^(0.5)

RAMSE.best

model.best<-errorsarlm(eq, data=dane.x, pkt.k.sym.listw, method="LU")

#model.best<-lagsarlm(eq, data=dane.x, pkt.k.sym.listw, method="LU")

#model.best<-lagsarlm(eq, data=dane.x, pkt.k.sym.listw, method="LU", type="mixed")

moran.test(model.best$residuals, pkt.k.sym.listw)

**# selection of out-of-sample points for predictions**

points.pred<-SpatialPoints(crds.out[1:n.new, ]) # selection of a new points

proj4string(points.pred)<-CRS("+proj=longlat +datum=NAD83") # spherical projections

points.pred<-spTransform(points.pred, CRS("+proj=merc +datum=NAD83")) # planar projections

a1<-over(points.pred, tes.poly) # assigning points to tessellation tiles

**# completing the draw when NA occurs**

**# determining the number of new points to be drawn (as from-to)**

a2<-n.new+1 # from …

a3<-which(is.na(a1))

a4<-a2+length(a3)-1 # to …

points.pred2<-SpatialPoints(crds.out[a2:a4, ]) # new points

proj4string(points.pred2)<-CRS("+proj=longlat +datum=NAD83") # spherical

points.pred2<-spTransform(points.pred2, CRS("+proj=merc +datum=NAD83")) # planar

a5<-over(points.pred2, tes.poly) # putting new points on the tiles

a1[which(is.na(a1))]<-a5 # overwriting with new points

forecasts1<-matrix(0, nrow=n.new, ncol=5)

colnames(forecasts1)<-c("predicted y","real y","crds x","crds y", "diff")

**# loop for forecasts for new points - separate match for each point**

for(i in 1:n.new){ # n.new

dane.x.new<-dane.x

xxx<-dane1.out[i, ]

dane.x.new[a1[i],]<-xxx

rownames(dane.x.new)<-1:dim(dane.x.new)[1]

**# prediction for out-of-sample calibrated best model**

pred<-predict(model.best, newdata=dane.x.new, listw=pkt.k.sym.listw, legacy.mixed=TRUE)

forecasts1[i,1]<- pred[a1[i]] # predicted value of y for a new point

forecasts1[i,2]<- xxx[, n.dep] # empirical value of y for a new point

forecasts1[i,3]<-xxx[, n.x] # x coordinates

forecasts1[i,4]<-xxx[, n.y] # y coordinates

}

**# calculating RAMSE for predictions**

forecasts1[,5]<-(forecasts1[,1]-forecasts1[,2])^2

RAMSE.pred<-(mean(forecasts1[,5]))^0.5

**# plotting tessellation + points of predictions**

plot(region.tes, main=" ") # tessellation plot

plot(region.ppp, add=TRUE, pch=".", col="darkblue", cex=2) # points added

points.pred<-SpatialPoints(forecasts1[, 3:4]) # selection of a new point

proj4string(points.pred)<-CRS("+proj=longlat +datum=NAD83") # spherical projection

points.pred<-spTransform(points.pred, CRS("+proj=merc +datum=NAD83"))

plot(points.pred, add=TRUE, bg="red", pch=21) # adding out-of-sample points to tessellation figure

**# outputing**

head(coef.boot) # bootstrapped coefficients

head(error.boot) # bootstrapped standard errors

head(quality.boot) # quality statistics & computation time

head(forecasts1) # quality of predictions

RAMSE.pred

RAMSE.best

summary(model.best)

sum(quality.boot[,3:4]) # full time in sec to tun bootstrap

# 

# # sf class – model in brief - R codes to run your own bootstrap model

**# please change lines in RED**

## # starter – reading and preparing data

**# loading packages**

set.seed(246)

library(sf) # st\_read(), st\_transform(), st\_distance()

library(ggplot2)

library(doBy) # for doBy()

library(spdep) # for nb2listw(), make.sym.nb(), knn2nb(), knearneigh(), moran.test()

library(spatialreg) # errorsarlm(), lagsarlm(), predict.sarlm()

library(sampling) # for strata()

library(cluster) # for pam()

**# Setting path of Working Directory**

#setwd("D:/My all/dataset") # change it according to your settings

**# reading contour map info sf class - NTS2 regions of Poland**

**# note that in this set of codes sf objects are written with CAPITAL LETTERS, while sp objects with small letters**

WOJ<-st\_read("wojewodztwa.shp")

WOJ<-st\_transform(WOJ, 4326) # change of projection into WGS84

WOJ.lub<-WOJ[WOJ$jpt\_nazwa\_=="lubelskie",] # fragment of the regional map, to match the point data

**# reading geolocated point data – ca.37.000 firms in Lubelskie region**

dane<-read.csv("point\_data.csv", sep=",", dec=".", header=TRUE)

**# summary of data**

summary(dane)

names(dane)

head(dane)

dane1.in<-dane[1:30000,] # split your data for training data

dane1.out<-dane[30001:37374,] # and test data

## # bootstrap simulation to find the best model

############################################

**# parameters of simulation – change here your parameters – now very limited just to run quickly the model**

n.col<-10 # number of iterations

n.row<-800 # number of observations in a sample

n.new<-10 # number of new points in the forecast

n.var<-5 # set the number of explanatory variables included in model equation

n.knn<-5 # set the number of knn for W

n.dep<-27 # which column is dependent y variable

n.x<-58 # which column are x coordinates

n.y<-59 # which column are y coordinates

############################################

**# set your equation**

eq<-roa~empl+prod+constr+serv+dist # model structure

############################################

crds.in<-dane1.in[,n.x:n.y] # separate the locations (x,y) of points used in training

crds.out<-dane1.out[,n.x:n.y] # separate the locations (x,y) of points used in preditions

###############################################

**# cluster your data- xy geocoordinates - to get irregular shapes for sampling**

groups<-kmeans(crds.in, n.row/100)

dane1.in$kmean<-groups$cluster

###############################################

**# selector selects the rows of observations for every iteration and saves in matrix**

**# later it will be used to recover, on which data the best model was estimated**

**# strata() from sampling:: package samples observations from groups (clusters)**

**# method="srswor" stands for simple random sampling without replacement (in given column)**

selector<-matrix(0, nrow=n.row, ncol=n.col)

for(i in 1:n.col){

vec<-sample(1:dim(dane1.in)[1], n.row, replace=FALSE)

x<-strata(dane1.in, "kmean", size=rep(100, times=n.row/100), method="srswor") # from sampling::

selector[,i]<-x$ID\_unit}

###################################################

**# objects to store results of iterations**

coef.boot<-matrix(0, nrow=n.col, ncol=n.var\*2+1) # for coefficients

error.boot<-matrix(0, nrow=n.col, ncol=n.var\*2+1) # for standard errors

fitted.boot<-matrix(0, nrow=n.row, ncol=n.col) # for fitted values

y.boot<-matrix(0, nrow=n.row, ncol=n.col) # retriving original values of y

quality.boot<-matrix(0, nrow=n.col, ncol=5) # AIC.ols, AIC.spatial, time.W, time.model, spatial.coef

colnames(quality.boot)<-c("AIC.ols", "AIC.spatial", "time.W", "time.model", "spatial.coef")

**# big loop – estimation of all models (OLS, SEM, SAR, SDM) on the same subsets with time measurement**

for(i in 1:n.col){

danex<-dane1.in[selector[,i],]

**# W matrix**

crds<-as.matrix(crds.in[selector[,i], ])

start.time <- Sys.time()

pkt.k.sym.listw<-nb2listw(make.sym.nb(knn2nb(knearneigh(crds, k=n.knn))))

end.time <- Sys.time()

time.W<- difftime(end.time, start.time, units="secs")

**# choose one model – add hash # for unwanted**

start.time <- Sys.time()

model<-errorsarlm(eq, data=danex, pkt.k.sym.listw, method="LU")

#model<-lagsarlm(eq, data=danex, pkt.k.sym.listw, method="LU")

#model<-lagsarlm(eq, data=danex, pkt.k.sym.listw, method="LU", type="mixed")

end.time <- Sys.time()

time.model<- difftime(end.time, start.time, units="secs")

**# saving the results into appropriate objects**

coef.boot[i,1:length(model$coefficients)]<-model$coefficients

error.boot[i, 1:length(model$coefficients)]<-model$rest.se

fitted.boot[,i]<-model$fitted.values

y.boot[,i]<-danex[,n.dep] # saving y values for each iteration in roa matrix

quality.boot[i,1]<-model$AIC\_lm.model # AIC.ols

quality.boot[i,2]<-AIC(model) # AIC.spatial

quality.boot[i,3]<-time.W

quality.boot[i,4]<-time.model

quality.boot[i,5]<-ifelse(is.null(model$rho)==TRUE, model$lambda, model$rho)

}

## # selection of the best model with PAM

clust.mod<-pam(cbind(coef.boot[,1:length(model$coefficients)], quality.boot[,5]),1) #cluster::pam(), for n<65.536

summary(clust.mod)

clust.mod$medoids # coefficients of selected best model

clust.mod$id.med # which iteration (model) is most representative

## # validation for out-of-sample

**# tessellation of observations from the selected best model**

points<-data.frame(x=dane1.in[selector[, clust.mod$id.med], n.x], y=dane1.in[selector[, clust.mod$id.med], n.y])

points.sf<-st\_as\_sf(points, coords=c("x", "y"), crs=4326, agr ="constant") # points in sf class

points.sfc<-st\_geometry(points.sf)

box.sfc<-st\_geometry(WOJ.lub)

points.union<-st\_union(points.sfc)

tess<-st\_voronoi(points.union, box.sfc)

tess.clip<-st\_intersection(st\_cast(tess), st\_union(box.sfc))

**# restoring the medoid model – choose proper specification**

dane.x<-dane1.in[selector[,clust.mod$id.med],] # data which were used in estimation of the best model

crds.x<-as.matrix(points) # we create W

pkt.k.sym.listw<-nb2listw(make.sym.nb(knn2nb(knearneigh(crds.x, k=n.knn))))

RAMSE.best<-(sum((y.boot[ , clust.mod$id.med] - fitted.boot[ , clust.mod$id.med])^2)/n.row)^(0.5)

RAMSE.best

model.best<-errorsarlm(eq, data=dane.x, pkt.k.sym.listw, method="LU")

#model.best<-lagsarlm(eq, data=dane.x, pkt.k.sym.listw, method="LU")

#model.best<-lagsarlm(eq, data=dane.x, pkt.k.sym.listw, method="LU", type="mixed")

moran.test(model.best$residuals, pkt.k.sym.listw)

**# selection of out-of-sample points for predictions**

points.pred<-dane1.out[1:n.new, c(1:2, n.x:n.y)] # selection of a new points

points.pred.sf<-st\_as\_sf(points.pred, coords=c("xxe", "yye"), crs=4326, agr ="constant") # points in sf class

tess.sf<-st\_sf(tess.clip)

a1<-st\_intersects(points.pred.sf, tess.sf) # new version of over()

forecasts1<-matrix(0, nrow=n.new, ncol=5)

colnames(forecasts1)<-c("predicted y","real y","crds x","crds y", "(predY-realY)^2")

**# loop for forecasts for new points - separate match for each point**

for(i in 1:n.new){ # n.new

dane.x.new<-dane.x

xxx<-dane1.out[i, ]

dane.x.new[unlist(a1[i]),]<-xxx

rownames(dane.x.new)<-1:dim(dane.x.new)[1]

**# prediction for out-of-sample calibrated selected model**

pred<-predict(model.best, newdata=dane.x.new, listw=pkt.k.sym.listw, legacy.mixed=TRUE)

forecasts1[i,1]<- pred[unlist(a1[i])] # predicted value of y for a new point

forecasts1[i,2]<- xxx[, n.dep] # empirical value of y for a new point

forecasts1[i,3]<-xxx[, n.x] # x coordinates

forecasts1[i,4]<-xxx[, n.y] # y coordinates

}

forecasts1[,5]<-(forecasts1[,1]-forecasts1[,2])^2

RAMSE.pred<-(mean(forecasts1[,5]))^0.5

**# outputing**

head(coef.boot) # bootstrapped coefficients

head(error.boot) # bootstrapped standard errors

head(quality.boot) # quality statistics & computation time

head(forecasts1) # quality of predictions

RAMSE.pred

RAMSE.best

summary(model.best)

sum(quality.boot[,3:4]) # full time in sec to tun bootstrap

# plotting tessellation + points of predictions

plot(tess.clip, main=" ") # tessellation plot

points(points[,1:2], add=TRUE, pch=".", col="darkblue", cex=2) # points added

points(points.pred[,3:4], bg="red", pch=21) # adding out-of-sample points to tessellation figure

# # sp class – full analytics - R codes to run bootstrap models as in paper

## 

## # starter – reading and preparing data

**# loading packages**

set.seed(357)

library(spdep) # for nb2listw(), make.sym.nb(), knn2nb(), knearneigh(), moran.test()

library(rgdal) # readOGR()

library(maptools) # for sp conversion

library(sp) # spTransform(), SpatialPoints(), CRS(), proj4string(), over(),

library(doBy) # for doBy()

library(spatialreg) # errorsarlm(), lagsarlm(), predict.sarlm()

library(sampling) # for strata()

library(RColorBrewer) # for brewer.pal()

library(cluster) # for pam()

library(spatstat) # for as.owin(), dirichlet(), ppp()

library(classInt) # for legend in Fig.2b

**# Setting path of Working Directory**

#setwd("D:/My all/dataset") # change it according to your settings

**# reading contour map with rgdal:: - NTS2 regions of Poland**

**# note that in this set of codes sf objects are written with CAPITAL LETTERS, while sp objects with small letters**

woj<-readOGR(".", "wojewodztwa") # 16 regions

woj<- spTransform(woj, CRS("+proj=longlat +datum=NAD83")) # changing the projections into NAD83

**# reading geolocated point data – ca.37.000 firms in Lubelskie region**

dane<-read.csv("geoloc\_data\_firms.csv", sep=";", dec=",", header=TRUE)

summary(dane)

names(dane)

head(dane)

**# This is what originally was included in REGON dataset. In fact, one uses real geo-location of firm and sector of its activity.**

**# ID ADDRESS**

#1 206455 24-100 PuÂławy PuÂławy ul. Jana Tadeusza Ostrowskiego 2

#2 130287 21-080 GarbĂłw GarbĂłw Drugi 226

#3 34039 23-413 Obsza Babice 163

#4 94804 23-235 Annopol Bliskowice 38

#5 13456 21-530 Piszczac Piszczac-Kolonia 35

#6 173065 21-412 AdamĂłw Czarna 43

**# STREET STREET.NO ZIP CITY\_POST CITY**

#1 ul. Jana Tadeusza Ostrowskiego 2 24-100 PuÂławy PuÂławy

#2 <NA> 226 21-080 GarbĂłw GarbĂłw Drugi

#3 <NA> 163 23-413 Obsza Babice

#4 <NA> 38 23-235 Annopol Bliskowice

#5 <NA> 35 21-530 Piszczac Piszczac-Kolonia

#6 <NA> 43 21-412 AdamĂłw Czarna

**# region2 poviat region3 subreg coords.x1**

#1 powiat puÂławski powiat puławski PuÂławy 4 22.00263

#2 powiat lubelski powiat lubelski gmina GarbĂłw 3 22.34528

#3 powiat biÂłgorajski powiat biłgorajski gmina Obsza 2 22.96182

#4 powiat kraÂśnicki powiat kraśnicki gmina Annopol 4 21.83905

#5 powiat bialski powiat bialski gmina Piszczac 1 23.37778

#6 powiat Âłukowski powiat łukowski gmina AdamĂłw 4 22.25986

**# coords.x2 LEGAL\_FORM1 LEGAL\_FORM2 OWNERSHIP PKD7 SEC\_PKD7 GR\_EMPL empl**

#1 51.40935 9 99 214 4511Z G 1 5

#2 51.35417 9 99 214 6622Z K 1 5

#3 50.31523 9 99 214 0111Z A 1 5

#4 50.95910 9 99 214 0150Z A 1 5

#5 51.96064 9 99 214 0143Z A 1 5

#6 51.74347 9 99 214 0150Z A 1 5

**# Wider econometric estimation needs more data, that is why they are created below.**

**# generating data on profitability**

**# this will play the role of dependent variable**

param<-data.frame(SEC\_PKD7=c("A", "B", "C", "D" ,"E", "F", "G", "H", "I", "J", "K" ,"L", "M", "N", "O", "P", "Q", "R", "S"), SEC\_agg=c("agri", "prod", "prod", "prod" ,"prod", "constr", "serv", "serv", "serv", "serv", "serv" ,"serv", "serv", "serv", "serv", "serv", "serv", "serv", "serv"), roa\_ind=c(2,2.5,3,3.5,4,4.5,5,5.5,6,6.5,7,7.5,8,8.5,9,9.5,10,10.5,11))

dane1<-merge(dane, param, by="SEC\_PKD7")

param2<-data.frame(SEC\_agg=c("agri", "prod", "constr", "serv"), roa\_sec=c(2,3.5,5,8))

dane1<-merge(dane1, param2, by="SEC\_agg")

dane1$roa\_geo<-ifelse(dane1$poviat=="powiat Lublin", 1.5,0)

dane1$roa\_param<-dane1$roa\_sec+dane1$roa\_geo # one can do: roa\_ind+roa\_geo

for(i in 1:dim(dane1)[1]){

dane1$roa[i]<-rnorm(1, dane1$roa\_param[i], 0.045)}

**# sectoral dummies**

dane1$sA<-ifelse(dane1$SEC\_PKD7=="A",1,0)

dane1$sB<-ifelse(dane1$SEC\_PKD7=="B",1,0)

dane1$sC<-ifelse(dane1$SEC\_PKD7=="C",1,0)

dane1$sD<-ifelse(dane1$SEC\_PKD7=="D",1,0)

dane1$sE<-ifelse(dane1$SEC\_PKD7=="E",1,0)

dane1$sF<-ifelse(dane1$SEC\_PKD7=="F",1,0)

dane1$sG<-ifelse(dane1$SEC\_PKD7=="G",1,0)

dane1$sH<-ifelse(dane1$SEC\_PKD7=="H",1,0)

dane1$sI<-ifelse(dane1$SEC\_PKD7=="I",1,0)

dane1$sJ<-ifelse(dane1$SEC\_PKD7=="J",1,0)

dane1$sK<-ifelse(dane1$SEC\_PKD7=="K",1,0)

dane1$sL<-ifelse(dane1$SEC\_PKD7=="L",1,0)

dane1$sM<-ifelse(dane1$SEC\_PKD7=="M",1,0)

dane1$sN<-ifelse(dane1$SEC\_PKD7=="N",1,0)

dane1$sO<-ifelse(dane1$SEC\_PKD7=="O",1,0)

dane1$sP<-ifelse(dane1$SEC\_PKD7=="P",1,0)

dane1$sQ<-ifelse(dane1$SEC\_PKD7=="Q",1,0)

dane1$sR<-ifelse(dane1$SEC\_PKD7=="R",1,0)

dane1$sS<-ifelse(dane1$SEC\_PKD7=="S",1,0)

dane1$agri<-ifelse(dane1$SEC\_agg=="agri",1,0)

dane1$prod<-ifelse(dane1$SEC\_agg=="prod",1,0)

dane1$constr<-ifelse(dane1$SEC\_agg=="constr",1,0)

dane1$serv<-ifelse(dane1$SEC\_agg=="serv",1,0)

**# calculating the distances – from all firms to centre of Lublin city (core location in region)**

coords<-as.matrix(data.frame(x=dane1$coords.x1, y=dane1$coords.x2))

core<-c(22.5666700, 51.2500000) # location of CBD in Lublin

dane1$dist<-spDistsN1(coords, core, longlat=TRUE)

**# random sorting of dataset**

dane1$los<-runif(dim(dane1)[1], 0,1)

dane1<-orderBy(~los, data=dane1)

**# random jitter correction of variables (as the values are very similar)**

dane1$los2<-rnorm(dim(dane1)[1], 0,0.0005)

dane1$empl<-dane1$empl+dane1$los2

dane1$los3<-rnorm(dim(dane1)[1], 0,0.0005)

dane1$prod<-dane1$prod+dane1$los3

dane1$los4<-rnorm(dim(dane1)[1], 0,0.0005)

dane1$constr<-dane1$constr+dane1$los4

dane1$los5<-rnorm(dim(dane1)[1], 0,0.0005)

dane1$serv<-dane1$serv+dane1$los5

dane1$los6<-rnorm(dim(dane1)[1], 0,0.0005)

dane1$dist<-dane1$dist+dane1$los6

**# split of data and correction of location by epsilon in in-sample and out-of-sample datasets**

epsilon.x<-rnorm(dim(dane1)[1], mean=0, sd=0.015)

epsilon.y<-rnorm(dim(dane1)[1], mean=0, sd=0.015)

dane1$xxe<-dane1[,14]+epsilon.x

dane1$yye<-dane1[,15]+epsilon.y

dane1.in<-dane1[1:30000,]

dane1.out<-dane1[30001:37374,]

dane1.out$los<-runif(dim(dane1.out)[1], 0,1)

dane1.out<-orderBy(~los, data=dane1.out)

######## END OF DATA PREPARATION ###################

**SEC\_agg SEC\_PKD7 ID ADDRESS**

10677 agri A 111442 21-136 Firlej Sobolew 39

26985 serv G 88333 22-335 ÂŻĂłÂłkiewka-Osada ÂŻĂłÂłkiew 59

8248 agri A 315492 20-468 Lublin Lublin ul. Leona Kruczkowskiego 8

24187 prod C 212006 24-170 KurĂłw KurĂłw ul. Tadeusza KoÂściuszki 69

12485 agri A 318950 20-611 Lublin Lublin ul. BolesÂława ÂŚmiaÂłego 7

14323 agri A 233018 08-550 KÂłoczew Bramka 78

**STREET STREET.NO ZIP CITY\_POST CITY**

10677 <NA> 39 21-136 Firlej Sobolew

26985 <NA> 59 22-335 ÂŻĂłÂłkiewka-Osada ÂŻĂłÂłkiew

8248 ul. Leona Kruczkowskiego 8 20-468 Lublin Lublin

24187 ul. Tadeusza KoÂściuszki 69 24-170 KurĂłw KurĂłw

12485 ul. BolesÂława ÂŚmiaÂłego 7 20-611 Lublin Lublin

14323 <NA> 78 08-550 KÂłoczew Bramka

**region2 poviat region3 subreg**

10677 powiat lubartowski powiat lubartowski gmina Firlej 3

26985 powiat krasnostawski powiat krasnostawski gmina ÂŻĂłÂłkiewka 2

8248 Lublin powiat Lublin Lublin 3

24187 powiat puÂławski powiat puławski gmina KurĂłw 4

12485 Lublin powiat Lublin Lublin 3

14323 powiat rycki powiat rycki gmina KÂłoczew 4

**coords.x1 coords.x2 LEGAL\_FORM1 LEGAL\_FORM2 OWNERSHIP PKD7 GR\_EMPL**

10677 22.47410 51.51694 9 99 214 0150Z 1

26985 22.85254 50.88325 9 99 214 4711Z 1

8248 22.57219 51.21366 9 99 214 0150Z 1

24187 22.18994 51.39580 9 99 214 1392Z 1

12485 22.53744 51.23922 9 99 214 0150Z 1

14323 22.02138 51.72837 9 99 214 0141Z 1

**empl roa\_ind roa\_sec roa\_geo roa\_param roa sA sB sC sD sE sF sG**

10677 5.000519 2 2.0 0.0 2.0 1.983343 1 0 0 0 0 0 0

26985 5.000790 5 8.0 0.0 8.0 8.042440 0 0 0 0 0 0 1

8248 4.999808 2 2.0 1.5 3.5 3.503759 1 0 0 0 0 0 0

24187 5.000745 3 3.5 0.0 3.5 3.508910 0 0 1 0 0 0 0

12485 5.000063 2 2.0 1.5 3.5 3.481518 1 0 0 0 0 0 0

14323 4.998978 2 2.0 0.0 2.0 1.987931 1 0 0 0 0 0 0

**sH sI sJ sK sL sM sN sO sP sQ sR sS agri prod constr**

10677 0 0 0 0 0 0 0 0 0 0 0 0 1 -0.0003436035 0.0004591658

26985 0 0 0 0 0 0 0 0 0 0 0 0 0 -0.0011445054 0.0001216126

8248 0 0 0 0 0 0 0 0 0 0 0 0 1 -0.0000505683 0.0003509886

24187 0 0 0 0 0 0 0 0 0 0 0 0 0 0.9995378867 -0.0006009576

12485 0 0 0 0 0 0 0 0 0 0 0 0 1 0.0000105884 0.0002562051

14323 0 0 0 0 0 0 0 0 0 0 0 0 1 -0.0004311137 0.0005874865

**serv dist los los2 los3**

10677 -1.028717e-04 30.389969 5.410193e-05 5.188503e-04 -0.0003436035

26985 9.993726e-01 45.456709 6.981613e-05 7.902950e-04 -0.0011445054

8248 9.137862e-05 4.061494 9.868899e-05 -1.923529e-04 -0.0000505683

24187 -7.336136e-05 30.868211 1.219867e-04 7.454510e-04 -0.0004621133

12485 -1.830129e-04 2.367233 1.656434e-04 6.329818e-05 0.0000105884

14323 -4.945271e-04 65.322113 2.137653e-04 -1.021590e-03 -0.0004311137

**los4 los5 los6**

10677 0.0004591658 -1.028717e-04 -2.936005e-04

26985 0.0001216126 -6.274238e-04 4.881613e-04

8248 0.0003509886 9.137862e-05 2.654125e-04

24187 -0.0006009576 -7.336136e-05 7.234496e-04

12485 0.0002562051 -1.830129e-04 -1.090864e-05

14323 0.0005874865 -4.945271e-04 -5.323068e-04

## # estimation of the full model (for comparison)

**# This code runs full sample models (OLS, SEM, SAR, SDM) and saves the results info files**

**# full OLS, SEM, SAR & SDM models**

eq<-roa~empl+prod+constr+serv+dist # equation

danex<-dane1.in # dataset

crds<-as.matrix(dane1.in[ ,57:58]) # 57:58 are spatial coordinates

**# spatial weights matrix k nearest neighbours knn=5**

pkt.k.sym.listw<-nb2listw(make.sym.nb(knn2nb(knearneigh(crds, k=5))))

**# estimation of models**

model.ols.full<-lm(eq, data=danex)

model.sem.full<-errorsarlm(eq, data=danex, pkt.k.sym.listw, method="LU")

model.sar.full<-lagsarlm(eq, data=danex, pkt.k.sym.listw, method="LU")

model.sdm.full<-lagsarlm(eq, data=danex, pkt.k.sym.listw, method="LU", type="mixed")

**# summary of models**

summary(model.ols.full)

summary(model.sem.full, Nagelkerke=TRUE)

summary(model.sar.full, Nagelkerke=TRUE)

summary(model.sdm.full, Nagelkerke=TRUE)

**# saving full regression results in txt files**

m<-summary(model.ols.full)

save(m, file="model\_ols\_full.RData")

m<-summary(model.sem.full)

save(m, file="model\_sem\_full.RData")

m<-summary(model.sar.full)

save(m, file="model\_sar\_full.RData")

m<-summary(model.sdm.full)

save(m, file="model\_sdm\_full.RData")

**# 🡪 and reading them back if needed**

model.ols.full<-get(load("model\_ols\_full.RData"))

model.sem.full<-get(load("model\_sem\_full.RData"))

model.sar.full<-get(load("model\_sar\_full.RData"))

model.sdm.full<-get(load("model\_sdm\_full.RData"))

## # bootstrap simulation to find the best model

# This code runs bootstrap simulation – each of four models (OLS, SEM, SAR, SDM) is estimated many times (as n.col = number of iterations). Each iteration uses a different data subset. All results are saved into objects.

**# This may take ca. 10 minutes with parameters as specified below**

############################################

**# parameters of simulation – change here your parameters – now very limited just to run quickly the model**

n.col<-50 # number of iterations

n.row<-800 # number of observations in a sample

nnew<-10 # number of new points in the forecast

############################################

**# full procedure – bootstrapped models**

############################################

**# selector matrix – with k-means clustering**

**# assigning points to clusters with k-means – to get irregular shapes for sampling**

**# visualisation of result – see Fig.5**

groups<-kmeans(dane1.in[,57:58], n.row/100)

dane1.in$kmean<-groups$cluster

plot(dane1.in[,57:58], col=1:max(groups$cluster))

**# selector selects the rows of observations for every iteration and saves in matrix**

**# later it will be used to recover, on which data the best model was estimated**

**# strata() from sampling:: package samples observations from groups (clusters)**

**# method="srswor" stands for simple random sampling without replacement (in given column)**

selector<-matrix(0, nrow=n.row, ncol=n.col)

for(i in 1:n.col){

vec<-sample(1:dim(dane1.in)[1], n.row, replace=FALSE)

x<-strata(dane1.in, "kmean", size=rep(100, times=n.row/100), method="srswor") # from sampling::

selector[,i]<-x$ID\_unit}

selector[1:5, 1:5] # matrix with randomly selected numbers, n.row x n.col

# [,1] [,2] [,3] [,4] [,5]

#[1,] 63 615 2 133 312

#[2,] 317 685 537 157 566

#[3,] 977 1265 593 527 569

#[4,] 1057 1432 864 634 935

#[5,] 1062 1448 1212 663 1205

#####################################

**# all models – objects to store results of iterations**

coef.ols<-matrix(0, nrow=n.col, ncol=6) # for coefficients

coef.sem<-matrix(0, nrow=n.col, ncol=6)

coef.sar<-matrix(0, nrow=n.col, ncol=6)

coef.sdm<-matrix(0, nrow=n.col, ncol=11)

error.ols<-matrix(0, nrow=n.col, ncol=6) # for standard errors

error.sem<-matrix(0, nrow=n.col, ncol=6)

error.sar<-matrix(0, nrow=n.col, ncol=6)

error.sdm<-matrix(0, nrow=n.col, ncol=11)

sig.ols<-matrix(0, nrow=n.col, ncol=6)

fitted.ols<-matrix(0, nrow=n.row, ncol=n.col) # for fitted values

fitted.sem<-matrix(0, nrow=n.row, ncol=n.col)

fitted.sar<-matrix(0, nrow=n.row, ncol=n.col)

fitted.sdm<-matrix(0, nrow=n.row, ncol=n.col)

quality<-matrix(0, nrow=n.col, ncol=5) # R2, AIC.ols, AIC.sem, AIC.sar, AIC.sdm

time<-matrix(0, nrow=n.col, ncol=5) # time.ols, time.W, time.sem, time.sar, time.sdm

roa<-matrix(0, nrow=n.row, ncol=n.col) # retriving original values of y

spatial<-matrix(0, nrow=n.col, ncol=3) # lambda.sem, rho.sar, rho.sdm

eq<-roa~empl+prod+constr+serv+dist # model structure

**# big loop – estimation of all models (OLS, SEM, SAR, SDM) on the same subsets with time measurement**

for(i in 1:n.col){

danex<-dane1.in[selector[,i],]

roa[,i]<-danex$roa # saving y values for each iteration in roa matrix

**# W matrix**

crds<-as.matrix(dane1.in[selector[,i], 57:58]) # spatial weights matrix W

start.time <- Sys.time()

pkt.k.sym.listw<-nb2listw(make.sym.nb(knn2nb(knearneigh(crds, k=5)))) # knn=5

end.time <- Sys.time()

time.W<- difftime(end.time, start.time, units="secs")

**# OLS model**

start.time <- Sys.time()

model.ols<-lm(eq, data=danex)

end.time <- Sys.time()

time.ols<- difftime(end.time, start.time, units="secs")

**# SEM model**

start.time <- Sys.time()

model.sem<-errorsarlm(eq, data=danex, pkt.k.sym.listw, method="LU")

end.time <- Sys.time()

time.sem<- difftime(end.time, start.time, units="secs")

**# SAR model**

start.time <- Sys.time()

model.sar<-lagsarlm(eq, data=danex, pkt.k.sym.listw, method="LU")

end.time <- Sys.time()

time.sar<- difftime(end.time, start.time, units="secs")

**# SDM model**

start.time <- Sys.time()

model.sdm<-lagsarlm(eq, data=danex, pkt.k.sym.listw, method="LU", type="mixed")

end.time <- Sys.time()

time.sdm<- difftime(end.time, start.time, units="secs")

**# saving the results into appropriate objects**

coef.ols[i,]<-summary(model.ols)$coefficients[,1]

error.ols[i,]<-summary(model.ols)$coefficients[,2]

sig.ols[i,]<-summary(model.ols)$coefficients[,4]

fitted.ols[,i]<-model.ols$fitted.values

coef.sem[i,]<-model.sem$coefficients

error.sem[i,]<-model.sem$rest.se

fitted.sem[,i]<-model.sem$fitted.values

coef.sar[i,]<-model.sar$coefficients

error.sar[i,]<-model.sar$rest.se

fitted.sar[,i]<-model.sar$fitted.values

coef.sdm[i,]<-model.sdm$coefficients

error.sdm[i,]<-model.sdm$rest.se

fitted.sdm[,i]<-model.sdm$fitted.values

quality[i,1]<-summary(model.ols)$r.squared # R2

quality[i,2]<-AIC(model.ols) # AIC.ols

quality[i,3]<-AIC(model.sem) # AIC.sem

quality[i,4]<-AIC(model.sar) # AIC.sar

quality[i,5]<-AIC(model.sdm) # AIC.sdm

time[i,1]<-time.ols

time[i,2]<-time.W

time[i,3]<-time.sem

time[i,4]<-time.sar

time[i,5]<-time.sdm

spatial[i,1]<-model.sem$lambda

spatial[i,2]<-model.sar$rho

spatial[i,3]<-model.sdm$rho

}

# each model (OLS, SAR, SEM, SDM) was estimated 50 times

head(coef.ols)

# [,1] [,2] [,3] [,4] [,5] [,6]

#[1,] 2.437616 0.0020284324 1.578931 3.006216 6.174388 -0.006400216

#[2,] 2.398009 0.0005306455 1.735549 3.049162 6.140863 -0.005760920

#[3,] 2.479301 -0.0003706600 1.737262 3.067651 6.173159 -0.006832407

#[4,] 2.492294 -0.0008471774 1.560847 2.978469 6.166155 -0.006828261

#[5,] 2.360972 0.0012591673 1.652750 2.967866 6.123976 -0.005223608

#[6,] 2.456224 -0.0006022282 1.658252 3.080859 6.150532 -0.006436162

head(time)

# [,1] [,2] [,3] [,4] [,5]

#[1,] 0.004276991 0.2657180 2.189982 3.173867 3.211553

#[2,] 0.002266884 0.2149570 2.127379 3.553725 3.363792

#[3,] 0.002393007 0.2408359 2.188978 3.327949 3.306635

#[4,] 0.002336979 0.2149439 2.088798 3.439352 3.406646

#[5,] 0.002411127 0.2539051 2.148157 3.269229 3.482678

#[6,] 0.002171993 0.2128470 2.105742 3.725344 3.992017

head(quality)

# [,1] [,2] [,3] [,4] [,5]

#[1,] 0.9880292 415.5217 -1025.8992 185.4271 -1042.1972

#[2,] 0.9887564 338.7113 -793.2285 163.7461 -803.4621

#[3,] 0.9873993 433.9909 -933.8936 214.2263 -939.4232

#[4,] 0.9879185 450.1025 -1029.9045 230.8638 -1033.9413

#[5,] 0.9892614 324.7827 -643.6447 188.6026 -647.3813

#[6,] 0.9877178 418.1233 -1003.2167 207.2063 -1007.3604

## # selection of the best model with PAM

# This code analyses bootstrap models and looks for the most central (medoid) one

# for clustering, one uses both beta coefficients and spatial coefficients (added with cbind() function)

# One always assumes one cluster only

**## PAM for OLS**

c1.ols<-pam(coef.ols,1) #cluster::pam(), works for n<65536, clustering all data into one cluster

summary(c1.ols)

c1.ols$clustering # clustering vector – only 1 values

c1.ols$medoids # coefficients of selected best model

c1.ols$id.med # which iteration (model) is most representative

#hopkins(coef.ols, n=nrow(coef.ols)-1) # Hopkins statistics to check clusterability, takes long…

**## PAM for SEM**

c1.sem<-pam(cbind(coef.sem, spatial[,1]),1) #cluster::pam(), works for n<65536

summary(c1.sem)

c1.sem$clustering # clustering vector – only 1 values

c1.sem$medoids # coefficients of selected best model

c1.sem$id.med # which iteration (model) is most representative

#hopkins(coef.sem, n=nrow(coef.sem)-1) # takes long…

**## PAM for SAR**

c1.sar<-pam(cbind(coef.sar, spatial[,2]),1) #cluster::pam(), works for n<65536

summary(c1.sar)

c1.sar$clustering

c1.sar$medoids

c1.sar$id.med

#hopkins(coef.sar, n=nrow(coef.sar)-1) #

**## PAM for SDM**

c1.sdm<-pam(cbind(coef.sdm, spatial[,3]),1) #cluster::pam(), works for n<65536

summary(c1.sdm)

c1.sdm$clustering

c1.sdm$medoids

c1.sdm$id.med

#hopkins(coef.sem, n=nrow(coef.sem)-1)

## # validation for out-of-sample

# For each model (OLS, SEM, SAR, SDM) procedure is as follows:

# One takes the best PAM model and observations used in its estimation

# On those observations, one runs tessellation (Voronoi polygons)

# On the best model, one runs the prediction – for a single new point in each prediction round

# For forecasts of all new points, one calculates RAMSE

**# preparing ppp objects for spatstat**

woj<-readOGR(".", "wojewodztwa") # 16 units (NTS2 regions)

woj<-spTransform(woj, CRS("+proj=longlat +datum=NAD83")) # spherical projections

woj<-spTransform(woj, CRS("+proj=merc +datum=NAD83")) # planar projections

region<-woj[woj@data$jpt\_nazwa\_=="lubelskie",] # one region only

region.owin<-as.owin(region) # spatstat:: requires planar coordinates

### # OLS predictions

**# tessellation of observations from the selected best model**

points<-data.frame(x=dane1.in[selector[,c1.ols$id.med],57], y=dane1.in[selector[,c1.ols$id.med],58])

points.sp<-SpatialPoints(points) # new points in sp class - spherical

proj4string(points.sp)<-CRS("+proj=longlat +datum=NAD83") # spherical

points.sp<-spTransform(points.sp, CRS("+proj=merc +datum=NAD83")) # planar

region.ppp<-ppp(x=points.sp@coords[,1], y=points.sp@coords[,2], window=region.owin) # points of ppp class

region.tes<-dirichlet(region.ppp) # Dirichlet / Voronoi tesselation

tes.poly<-as(region.tes, "SpatialPolygons") # converting tessellation into sp

proj4string(tes.poly)<-CRS("+proj=merc +datum=NAD83") # giving projections

tes.poly<-spTransform(tes.poly, CRS("+proj=merc +datum=NAD83")) # planar projections

plot(region.tes, main=" ") # tessellation plot

plot(region.ppp, add=TRUE, pch=".", col="darkblue", cex=2) # adding points in blue

**# restoring the medoid model**

dane.x<-dane1.in[selector[,c1.ols$id.med],] # we choose the data which were used in estimation of the best model

crds<-as.matrix(dane.x[, 57:58]) # we create W

RAMSE.med.ols<-(sum((dane1.in[selector[,c1.ols$id.med], 26]-fitted.ols[,c1.ols$id.med])^2)/n.row)^(0.5)

RAMSE.med.ols

**# knn object # to use in Moran test in case of OLS**

pkt.k.sym.listw<-nb2listw(make.sym.nb(knn2nb(knearneigh(crds, k=5))))

eq<-roa~empl+prod+constr+serv+dist # regression equation

model.ols<-lm(eq, data=dane.x) # ols model

moran.test(model.ols$residuals, pkt.k.sym.listw) # Moran test for OLS residuals for spatial autocorrelation

**# selection of out-of-sample points for predictions**

nnew<-100 # number of new points in the forecast

points.pred<-SpatialPoints(dane1.out[1:nnew, 57:58]) # selection of a new point

proj4string(points.pred)<-CRS("+proj=longlat +datum=NAD83") # spherical projection

points.pred<-spTransform(points.pred, CRS("+proj=merc +datum=NAD83")) # planar projection of points

a1<-over(points.pred, tes.poly) # assigning points to tessellation tiles

**# completing the draw when NA occurs**

# determining the number of new points to be drawn (as from-to)

a2<-nnew+1 # from …

a3<-which(is.na(a1))

a4<-a2+length(a3)-1 # to …

points.pred2<-SpatialPoints(dane1.out[a2:a4, 57:58]) # new points

proj4string(points.pred2)<-CRS("+proj=longlat +datum=NAD83") # spherical

points.pred2<-spTransform(points.pred2, CRS("+proj=merc +datum=NAD83"))

a5<-over(points.pred2, tes.poly) # putting new points on the tile

a1[which(is.na(a1))]<-a5 # overwriting with new points

**# object for new points**

forecasts1<-matrix(0, nrow=nnew, ncol=5)

colnames(forecasts1)<-c("predicted y","real y","crds x","crds y", "diff")

**# loop for forecasts for new points**

**# there is a separate match for each point**

for(i in 1:nnew){

# point by point - assigning new data to the old data set

dane.x.new<-dane.x

xxx<-dane1.out[i, ]

dane.x.new[a1[i],]<-xxx

rownames(dane.x.new)<-1:dim(dane.x.new)[1]

# prediction for out-of-sample calibrated SDM model

pred<-predict(model.ols, newdata=dane.x.new)

pred[a1[i]] # prediction for a new point

xxx[,26] # empirical y value of the new point

forecasts1[i,1]<- pred[a1[i]] # predicted value of y

forecasts1[i,2]<- xxx[, 26] # empirical value of y

forecasts1[i,3]<-xxx[, 57] # x coordinates

forecasts1[i,4]<-xxx[, 58] # y coordinates

}

forecasts1[,5]<-(forecasts1[,1]-forecasts1[,2])^2

RAMSE.ols<-(mean(forecasts1[,5]))^0.5

head(forecasts1)

RAMSE.ols # quality of predictions

##########################

### # SEM predictions

**# tessellation of observations from the selected best model**

points<-data.frame(x=dane1.in[selector[,c1.sem$id.med],57], y=dane1.in[selector[,c1.sem$id.med],58])

points.sp<-SpatialPoints(points) # new points in sp class - spherical

proj4string(points.sp)<-CRS("+proj=longlat +datum=NAD83") # spherical projections

points.sp<-spTransform(points.sp, CRS("+proj=merc +datum=NAD83")) # planar projections

region.ppp<-ppp(x=points.sp@coords[,1], y=points.sp@coords[,2], window=region.owin) # points of ppp class

region.tes<-dirichlet(region.ppp) # Dirichlet / Voronoi tesselation

tes.poly<-as(region.tes, "SpatialPolygons") # converting tessellation into sp

proj4string(tes.poly)<-CRS("+proj=merc +datum=NAD83") # giving projections

tes.poly<-spTransform(tes.poly, CRS("+proj=merc +datum=NAD83")) # planar projections

plot(region.tes, main=" ") # tessellation plot

plot(region.ppp, add=TRUE, pch=".", col="darkblue", cex=2) # points added

**# restoring the medoid model**

dane.x<-dane1.in[selector[,c1.sem$id.med],] # we choose the data which were used in estimation of the best model

crds<-as.matrix(dane.x[, 57:58]) # we create W

RAMSE.med.sem<-(sum((dane1.in[selector[,c1.sem$id.med], 26]-fitted.sem[,c1.sem$id.med])^2)/n.row)^(0.5)

RAMSE.med.sem

pkt.k.sym.listw<-nb2listw(make.sym.nb(knn2nb(knearneigh(crds, k=5, longlat = NULL)))) # knn=5

eq<-roa~empl+prod+constr+serv+dist

model.sem<-errorsarlm(eq, data=dane.x, pkt.k.sym.listw, method="LU")

moran.test(model.sem$residuals, pkt.k.sym.listw)

**# selection of out-of-sample points for predictions**

nnew<-100 # number of new points in the forecast

points.pred<-SpatialPoints(dane1.out[1:nnew, 57:58]) # selection of a new points

proj4string(points.pred)<-CRS("+proj=longlat +datum=NAD83") # spherical projections

points.pred<-spTransform(points.pred, CRS("+proj=merc +datum=NAD83")) # planar projections

a1<-over(points.pred, tes.poly) # assigning points to tessellation tiles

**# completing the draw when NA occurs**

**# determining the number of new points to be drawn (as from-to)**

a2<-nnew+1 # from …

a3<-which(is.na(a1))

a4<-a2+length(a3)-1 # to …

points.pred2<-SpatialPoints(dane1.out[a2:a4, 57:58]) # new points

proj4string(points.pred2)<-CRS("+proj=longlat +datum=NAD83") # spherical

points.pred2<-spTransform(points.pred2, CRS("+proj=merc +datum=NAD83")) # planar

a5<-over(points.pred2, tes.poly) # putting new points on the tiles

a1[which(is.na(a1))]<-a5 # overwriting with new points

forecasts1<-matrix(0, nrow=nnew, ncol=5)

colnames(forecasts1)<-c("predicted y","real y","crds x","crds y", "diff")

**# loop for forecasts for new points**

**# there is a separate match for each point**

for(i in 1:nnew){ # nnew

# point by point - assigning new data to the old data set

dane.x.new<-dane.x

xxx<-dane1.out[i, ]

dane.x.new[a1[i],]<-xxx

rownames(dane.x.new)<-1:dim(dane.x.new)[1]

**# prediction for out-of-sample calibrated SDM model**

pred<-predict(model.sem, newdata=dane.x.new, listw=pkt.k.sym.listw, legacy.mixed=TRUE)

pred[a1[i]] # prediction for a new point

xxx[,26] # empirical y value of the new point

forecasts1[i,1]<- pred[a1[i]] # predicted value of y

forecasts1[i,2]<- xxx[, 26] # empirical value of y

forecasts1[i,3]<-xxx[, 57] # x coordinates

forecasts1[i,4]<-xxx[, 58] # y coordinates

}

forecasts1[,5]<-(forecasts1[,1]-forecasts1[,2])^2

RAMSE.sem<-(mean(forecasts1[,5]))^0.5

head(forecasts1)

RAMSE.sem

#############################

### # SAR predictions

# tessellation of observations from the selected best model

points<-data.frame(x=dane1.in[selector[,c1.sar$id.med],57], y=dane1.in[selector[,c1.sar$id.med],58])

points.sp<-SpatialPoints(points) # new points in sp class - spherical

proj4string(points.sp)<-CRS("+proj=longlat +datum=NAD83") # spherical

points.sp<-spTransform(points.sp, CRS("+proj=merc +datum=NAD83")) # planar

region.ppp<-ppp(x=points.sp@coords[,1], y=points.sp@coords[,2], window=region.owin) # points of ppp class

region.tes<-dirichlet(region.ppp) # Dirichlet tesselation

tes.poly<-as(region.tes, "SpatialPolygons")

proj4string(tes.poly)<-CRS("+proj=merc +datum=NAD83")

tes.poly<-spTransform(tes.poly, CRS("+proj=merc +datum=NAD83")) #planar

plot(region.tes, main=" ") # tessellation plot

plot(region.ppp, add=TRUE, pch=".", col="darkblue", cex=2)

nnew<-100 # number of new points in the forecast

forecasts1<-matrix(0, nrow=nnew, ncol=5)

colnames(forecasts1)<-c("predicted y","real y","crds x","crds y", "diff")

**# restoring the medoid model**

dane.x<-dane1.in[selector[,c1.sar$id.med],] # we choose the data which were used in estimation of the best model

crds<-as.matrix(dane.x[, 57:58]) # we create W

RAMSE.med.sar<-(sum((dane1.in[selector[,c1.sar$id.med], 26]-fitted.sar[,c1.sar$id.med])^2)/n.row)^(0.5)

RAMSE.med.sar

pkt.k.sym.listw<-nb2listw(make.sym.nb(knn2nb(knearneigh(crds, k=5, longlat = NULL)))) # knn=5

eq<-roa~empl+prod+constr+serv+dist

model.sar<-lagsarlm(eq, data=dane.x, pkt.k.sym.listw, method="LU")

moran.test(model.sar$residuals, pkt.k.sym.listw)

**# selection of out-of-sample points for predictions**

points.pred<-SpatialPoints(dane1.out[1:nnew, 57:58]) # selection of a new point

proj4string(points.pred)<-CRS("+proj=longlat +datum=NAD83") # spherical projection

points.pred<-spTransform(points.pred, CRS("+proj=merc +datum=NAD83"))

a1<-over(points.pred, tes.poly) # assigning points to tessellation tiles

**# completing the draw when NA occurs**

**# determining the number of new points to be drawn (as from-to)**

a2<-nnew+1 # from …

a3<-which(is.na(a1))

a4<-a2+length(a3)-1 # to …

points.pred2<-SpatialPoints(dane1.out[a2:a4, 57:58]) # new points

proj4string(points.pred2)<-CRS("+proj=longlat +datum=NAD83") # spherical

points.pred2<-spTransform(points.pred2, CRS("+proj=merc +datum=NAD83"))

a5<-over(points.pred2, tes.poly) # putting new points on the tile

a1[which(is.na(a1))]<-a5 # overwriting with new points

# loop for forecasts for new points # there is a separate match for each point

for(i in 1:nnew){

# point by point - assigning new data to the old data set

dane.x.new<-dane.x

xxx<-dane1.out[i, ]

dane.x.new[a1[i],]<-xxx

rownames(dane.x.new)<-1:dim(dane.x.new)[1]

# prediction for out-of-sample calibrated SDM model

pred<-predict(model.sar, newdata=dane.x.new, listw=pkt.k.sym.listw, legacy.mixed=TRUE)

pred[a1[i]] # prediction for a new point

xxx[,26] # empirical y value of the new point

forecasts1[i,1]<- pred[a1[i]] # predicted value of y

forecasts1[i,2]<- xxx[, 26] # empirical value of y

forecasts1[i,3]<-xxx[, 57] # x coordinates

forecasts1[i,4]<-xxx[, 58] # y coordinates

}

forecasts1[,5]<-(forecasts1[,1]-forecasts1[,2])^2

RAMSE.sar<-(mean(forecasts1[,5]))^0.5

head(forecasts1)

RAMSE.sar

#########################################

### # SDM predictions

# tessellation of observations from the selected best model

points<-data.frame(x=dane1.in[selector[,c1.sdm$id.med], 57], y=dane1.in[selector[,c1.sdm$id.med], 58])

points.sp<-SpatialPoints(points) # new points in sp class - spherical

proj4string(points.sp)<-CRS("+proj=longlat +datum=NAD83") # spherical

points.sp<-spTransform(points.sp, CRS("+proj=merc +datum=NAD83")) # planar

region.ppp<-ppp(x=points.sp@coords[,1], y=points.sp@coords[,2], window=region.owin) # points of ppp class

region.tes<-dirichlet(region.ppp) # Dirichlet tesselation

tes.poly<-as(region.tes, "SpatialPolygons")

proj4string(tes.poly)<-CRS("+proj=merc +datum=NAD83")

tes.poly<-spTransform(tes.poly, CRS("+proj=merc +datum=NAD83")) #planar

plot(region.tes, main=" ") # tessellation plot

plot(region.ppp, add=TRUE, pch=".", col="darkblue", cex=2)

nnew<-100 # number of new points in the forecast

forecasts1<-matrix(0, nrow=nnew, ncol=5)

colnames(forecasts1)<-c("predicted y","real y","crds x","crds y", "diff")

**# restoring the medoid model**

dane.x<-dane1.in[selector[,c1.sdm$id.med],] # we choose the data which were used in estimation of the best model

crds<-as.matrix(dane.x[, 57:58]) # we create W

RAMSE.med.sdm<-(sum((dane1.in[selector[,c1.sdm$id.med], 26]-fitted.sdm[,c1.sdm$id.med])^2)/n.row)^(0.5)

RAMSE.med.sdm

pkt.k.sym.listw<-nb2listw(make.sym.nb(knn2nb(knearneigh(crds, k=5, longlat = NULL)))) # knn=5

eq<-roa~empl+prod+constr+serv+dist

model.sdm<-lagsarlm(eq, data=dane.x, pkt.k.sym.listw, method="LU", type="mixed")

moran.test(model.sdm$residuals, pkt.k.sym.listw)

**# selection of out-of-sample points for predictions**

points.pred<-SpatialPoints(dane1.out[1:nnew, 57:58]) # selection of a new point

proj4string(points.pred)<-CRS("+proj=longlat +datum=NAD83") # spherical projection

points.pred<-spTransform(points.pred, CRS("+proj=merc +datum=NAD83"))

a1<-over(points.pred, tes.poly) # assigning points to tessellation tiles

# completing the draw when NA occurs # determining the number of new points to be drawn (as from-to)

a2<-nnew+1 # from …

a3<-which(is.na(a1))

a4<-a2+length(a3)-1 # to …

points.pred2<-SpatialPoints(dane1.out[a2:a4, 57:58]) # new points

proj4string(points.pred2)<-CRS("+proj=longlat +datum=NAD83") # spherical

points.pred2<-spTransform(points.pred2, CRS("+proj=merc +datum=NAD83"))

a5<-over(points.pred2, tes.poly) # putting new points on the tile

a1[which(is.na(a1))]<-a5 # overwriting with new points

# loop for forecasts for new points # there is a separate match for each point

for(i in 1:nnew){

# point by point - assigning new data to the old data set

dane.x.new<-dane.x

xxx<-dane1.out[i, ]

dane.x.new[a1[i],]<-xxx

rownames(dane.x.new)<-1:dim(dane.x.new)[1]

# prediction for out-of-sample calibrated SDM model

pred<-predict(model.sdm, newdata=dane.x.new, listw=pkt.k.sym.listw, legacy.mixed=TRUE)

pred[a1[i]] # prediction for a new point

xxx[,26] # empirical y value of the new point

forecasts1[i,1]<- pred[a1[i]] # predicted value of y

forecasts1[i,2]<- xxx[, 26] # empirical value of y

forecasts1[i,3]<-xxx[, 57] # x coordinates

forecasts1[i,4]<-xxx[, 58] # y coordinates

}

forecasts1[,5]<-(forecasts1[,1]-forecasts1[,2])^2

RAMSE.sdm<-(mean(forecasts1[,5]))^0.5

head(forecasts1)

RAMSE.sdm

#########################################################

### # all RAMSE together – for models and for forecasts

RAMSE.ols

RAMSE.sem

RAMSE.sar

RAMSE.sdm

RAMSE.med.ols

RAMSE.med.sem

RAMSE.med.sar

RAMSE.med.sdm

###########################################

### # full sample SDM - to get RAMSE of forecast from the full-sample model

# tessellation of observations from the selected best model

points<-data.frame(x=dane1.in[ , 57], y=dane1.in[ , 58])

points.sp<-SpatialPoints(points) # new points in sp class - spherical

proj4string(points.sp)<-CRS("+proj=longlat +datum=NAD83") # spherical

points.sp<-spTransform(points.sp, CRS("+proj=merc +datum=NAD83")) # planar

region.ppp<-ppp(x=points.sp@coords[,1], y=points.sp@coords[,2], window=region.owin) # points of ppp class

region.tes<-dirichlet(region.ppp) # Dirichlet tesselation

tes.poly<-as(region.tes, "SpatialPolygons")

proj4string(tes.poly)<-CRS("+proj=merc +datum=NAD83")

tes.poly<-spTransform(tes.poly, CRS("+proj=merc +datum=NAD83")) #planar

plot(region.tes, main=" ") # tessellation plot

plot(region.ppp, add=TRUE, pch=".", col="darkblue", cex=2)

nnew<-100 # number of new points in the forecast

forecasts1<-matrix(0, nrow=nnew, ncol=5)

colnames(forecasts1)<-c("predicted y","real y","crds x","crds y", "diff")

pkt.k.sym.listw<-nb2listw(make.sym.nb(knn2nb(knearneigh(crds, k=5, longlat = NULL)))) # knn=5

eq<-roa~empl+prod+constr+serv+dist

model.sdm.full<-lagsarlm(eq, data=dane.x, pkt.k.sym.listw, method="LU", type="mixed")

moran.test(model.sdm.full$residuals, pkt.k.sym.listw)

**# selection of out-of-sample points for predictions**

points.pred<-SpatialPoints(dane1.out[1:nnew, 57:58]) # selection of a new point

proj4string(points.pred)<-CRS("+proj=longlat +datum=NAD83") # spherical projection

points.pred<-spTransform(points.pred, CRS("+proj=merc +datum=NAD83"))

a1<-over(points.pred, tes.poly) # assigning points to tessellation tiles

# completing the draw when NA occurs # determining the number of new points to be drawn (as from-to)

a2<-nnew+1 # from …

a3<-which(is.na(a1))

a4<-a2+length(a3)-1 # to …

points.pred2<-SpatialPoints(dane1.out[a2:a4, 57:58]) # new points

proj4string(points.pred2)<-CRS("+proj=longlat +datum=NAD83") # spherical

points.pred2<-spTransform(points.pred2, CRS("+proj=merc +datum=NAD83"))

a5<-over(points.pred2, tes.poly) # putting new points on the tile

a1[which(is.na(a1))]<-a5 # overwriting with new points

# loop for forecasts for new points # there is a separate match for each point

for(i in 1:nnew){

# point by point - assigning new data to the old data set

dane.x.new<-dane.x

xxx<-dane1.out[i, ]

dane.x.new[a1[i],]<-xxx

rownames(dane.x.new)<-1:dim(dane.x.new)[1]

# prediction for out-of-sample calibrated SDM model

pred<-predict(model.sdm, newdata=dane.x.new, listw=pkt.k.sym.listw, legacy.mixed=TRUE)

pred[a1[i]] # prediction for a new point

xxx[,26] # empirical y value of the new point

forecasts1[i,1]<- pred[a1[i]] # predicted value of y

forecasts1[i,2]<- xxx[, 26] # empirical value of y

forecasts1[i,3]<-xxx[, 57] # x coordinates

forecasts1[i,4]<-xxx[, 58] # y coordinates

}

forecasts1[,5]<-(forecasts1[,1]-forecasts1[,2])^2

RAMSE.sdm.full<-(mean(forecasts1[,5]))^0.5

head(forecasts1)

RAMSE.sdm.full

# # sf class R codes to run bootstrap model

## # starter – reading and preparing data

**# loading packages**

set.seed(246)

library(sf) # st\_read(), st\_transform(), st\_distance()

library(ggplot2)

library(doBy) # for doBy()

library(spdep) # for nb2listw(), make.sym.nb(), knn2nb(), knearneigh(), moran.test()

library(spatialreg) # errorsarlm(), lagsarlm(), predict.sarlm()

library(sampling) # for strata()

library(cluster) # for pam()

**# Setting path of Working Directory**

#setwd("D:/My all/dataset") # change it according to your settings

**# reading contour map info sf class - NTS2 regions of Poland**

**# note that in this set of codes sf objects are written with CAPITAL LETTERS, while sp objects with small letters**

WOJ<-st\_read("wojewodztwa.shp")

WOJ<-st\_transform(WOJ, 4326) # change of projection into WGS84

**# fragment of the regional map, to match the point data**

WOJ.lub<-WOJ[WOJ$jpt\_nazwa\_=="lubelskie",]

**# reading geolocated point data – ca.37.000 firms in Lubelskie region**

dane<-read.csv("geoloc\_data\_firms.csv", sep=";", dec=",", header=TRUE)

**# summary of data**

summary(dane)

names(dane)

head(dane)

# ID ADDRESS

#1 206455 24-100 Pu³awy Pu³awy ul. Jana Tadeusza Ostrowskiego 2

#2 130287 21-080 Garbów Garbów Drugi 226

#3 34039 23-413 Obsza Babice 163

#4 94804 23-235 Annopol Bliskowice 38

#5 13456 21-530 Piszczac Piszczac-Kolonia 35

#6 173065 21-412 Adamów Czarna 43

# STREET STREET.NO ZIP CITY\_POST CITY

#1 ul. Jana Tadeusza Ostrowskiego 2 24-100 Pu³awy Pu³awy

#2 <NA> 226 21-080 Garbów Garbów Drugi

#3 <NA> 163 23-413 Obsza Babice

#4 <NA> 38 23-235 Annopol Bliskowice

#5 <NA> 35 21-530 Piszczac Piszczac-Kolonia

#6 <NA> 43 21-412 Adamów Czarna

# region2 poviat region3 subreg coords.x1

#1 powiat pu³awski powiat pu\xb3awski Pu³awy 4 22.00263

#2 powiat lubelski powiat lubelski gmina Garbów 3 22.34528

#3 powiat bi³gorajski powiat bi\xb3gorajski gmina Obsza 2 22.96182

#4 powiat kra\u009cnicki powiat kra\x9cnicki gmina Annopol 4 21.83905

#5 powiat bialski powiat bialski gmina Piszczac 1 23.37778

#6 powiat ³ukowski powiat \xb3ukowski gmina Adamów 4 22.25986

# coords.x2 LEGAL\_FORM1 LEGAL\_FORM2 OWNERSHIP PKD7 SEC\_PKD7 GR\_EMPL empl

#1 51.40935 9 99 214 4511Z G 1 5

#2 51.35417 9 99 214 6622Z K 1 5

#3 50.31523 9 99 214 0111Z A 1 5

#4 50.95910 9 99 214 0150Z A 1 5

#5 51.96064 9 99 214 0143Z A 1 5

#6 51.74347 9 99 214 0150Z A 1 5

**# Wider econometric estimation needs more data, that is why they are created below.**

**# generating data on profitability**

**# this will play the role of dependent variable**

param<-data.frame(SEC\_PKD7=c("A", "B", "C", "D" ,"E", "F", "G", "H", "I", "J", "K" ,"L", "M", "N", "O", "P", "Q", "R", "S"), SEC\_agg=c("agri", "prod", "prod", "prod" ,"prod", "constr", "serv", "serv", "serv", "serv", "serv" ,"serv", "serv", "serv", "serv", "serv", "serv", "serv", "serv"), roa\_ind=c(2,2.5,3,3.5,4,4.5,5,5.5,6,6.5,7,7.5,8,8.5,9,9.5,10,10.5,11))

dane1<-merge(dane, param, by="SEC\_PKD7")

param2<-data.frame(SEC\_agg=c("agri", "prod", "constr", "serv"), roa\_sec=c(2,3.5,5,8))

dane1<-merge(dane1, param2, by="SEC\_agg")

dane1$roa\_geo<-ifelse(dane1$poviat=="powiat Lublin", 1.5,0)

dane1$roa\_param<-dane1$roa\_sec+dane1$roa\_geo # one can do: roa\_ind+roa\_geo

for(i in 1:dim(dane1)[1]){

dane1$roa[i]<-rnorm(1, dane1$roa\_param[i], 0.045)}

**# sectoral dummies**

dane1$sA<-ifelse(dane1$SEC\_PKD7=="A",1,0)

dane1$sB<-ifelse(dane1$SEC\_PKD7=="B",1,0)

dane1$sC<-ifelse(dane1$SEC\_PKD7=="C",1,0)

dane1$sD<-ifelse(dane1$SEC\_PKD7=="D",1,0)

dane1$sE<-ifelse(dane1$SEC\_PKD7=="E",1,0)

dane1$sF<-ifelse(dane1$SEC\_PKD7=="F",1,0)

dane1$sG<-ifelse(dane1$SEC\_PKD7=="G",1,0)

dane1$sH<-ifelse(dane1$SEC\_PKD7=="H",1,0)

dane1$sI<-ifelse(dane1$SEC\_PKD7=="I",1,0)

dane1$sJ<-ifelse(dane1$SEC\_PKD7=="J",1,0)

dane1$sK<-ifelse(dane1$SEC\_PKD7=="K",1,0)

dane1$sL<-ifelse(dane1$SEC\_PKD7=="L",1,0)

dane1$sM<-ifelse(dane1$SEC\_PKD7=="M",1,0)

dane1$sN<-ifelse(dane1$SEC\_PKD7=="N",1,0)

dane1$sO<-ifelse(dane1$SEC\_PKD7=="O",1,0)

dane1$sP<-ifelse(dane1$SEC\_PKD7=="P",1,0)

dane1$sQ<-ifelse(dane1$SEC\_PKD7=="Q",1,0)

dane1$sR<-ifelse(dane1$SEC\_PKD7=="R",1,0)

dane1$sS<-ifelse(dane1$SEC\_PKD7=="S",1,0)

dane1$agri<-ifelse(dane1$SEC\_agg=="agri",1,0)

dane1$prod<-ifelse(dane1$SEC\_agg=="prod",1,0)

dane1$constr<-ifelse(dane1$SEC\_agg=="constr",1,0)

dane1$serv<-ifelse(dane1$SEC\_agg=="serv",1,0)

**# data in sf class**

dane.sf<-st\_as\_sf(dane1, coords=c("coords.x1", "coords.x2"), crs=4326, agr="constant")

**# calculating the distances – from all firms to centre of Lublin city (core location in region)**

**# st\_distance calculates distances in pairs of observations from two objects – CBD location was multiplied**

core<-c(22.5666700, 51.2500000) # location of CBD in Lublin

CBD.df<-data.frame(ID=1:dim(dane1)[1], coords.x1=rep(22.5666700, times=dim(dane1)[1]), coords.x2=rep(51.2500000, times=dim(dane1)[1]))

CBD.sf<-st\_as\_sf(CBD.df, coords=c("coords.x1", "coords.x2"), crs=4326, agr="constant") # using sf::

dist.to.CBD<-**st\_distance**(dane.sf, CBD.sf, by\_element=TRUE) # we get raw result in meters, from sf::

dane1$dist<-as.numeric(dist.to.CBD)/1000 # losing the unit of data [m] (due to conversion to km)

**# random sorting of dataset**

dane1$los<-runif(dim(dane1)[1], 0,1)

dane1<-orderBy(~los, data=dane1)

**# random jitter correction of variables (as the values are very similar)**

dane1$los2<-rnorm(dim(dane1)[1], 0,0.0005)

dane1$empl<-dane1$empl+dane1$los2

dane1$los3<-rnorm(dim(dane1)[1], 0,0.0005)

dane1$prod<-dane1$prod+dane1$los3

dane1$los4<-rnorm(dim(dane1)[1], 0,0.0005)

dane1$constr<-dane1$constr+dane1$los4

dane1$los5<-rnorm(dim(dane1)[1], 0,0.0005)

dane1$serv<-dane1$serv+dane1$los5

dane1$los6<-rnorm(dim(dane1)[1], 0,0.0005)

dane1$dist<-dane1$dist+dane1$los6

**# split of data and correction of location by epsilon in in-sample and out-of-sample datasets**

epsilon.x<-rnorm(dim(dane1)[1], mean=0, sd=0.015)

epsilon.y<-rnorm(dim(dane1)[1], mean=0, sd=0.015)

dane1$xxe<-dane1[,14]+epsilon.x

dane1$yye<-dane1[,15]+epsilon.y

dane1.in<-dane1[1:30000,]

dane1.out<-dane1[30001:37374,]

dane1.out$los<-runif(dim(dane1.out)[1], 0,1)

dane1.out<-orderBy(~los, data=dane1.out)

######## END OF DATA PREPARATION ###################

**SEC\_agg SEC\_PKD7 ID ADDRESS**

10677 agri A 111442 21-136 Firlej Sobolew 39

26985 serv G 88333 22-335 ÂŻĂłÂłkiewka-Osada ÂŻĂłÂłkiew 59

8248 agri A 315492 20-468 Lublin Lublin ul. Leona Kruczkowskiego 8

24187 prod C 212006 24-170 KurĂłw KurĂłw ul. Tadeusza KoÂściuszki 69

12485 agri A 318950 20-611 Lublin Lublin ul. BolesÂława ÂŚmiaÂłego 7

14323 agri A 233018 08-550 KÂłoczew Bramka 78

**STREET STREET.NO ZIP CITY\_POST CITY**

10677 <NA> 39 21-136 Firlej Sobolew

26985 <NA> 59 22-335 ÂŻĂłÂłkiewka-Osada ÂŻĂłÂłkiew

8248 ul. Leona Kruczkowskiego 8 20-468 Lublin Lublin

24187 ul. Tadeusza KoÂściuszki 69 24-170 KurĂłw KurĂłw

12485 ul. BolesÂława ÂŚmiaÂłego 7 20-611 Lublin Lublin

14323 <NA> 78 08-550 KÂłoczew Bramka

**region2 poviat region3 subreg**

10677 powiat lubartowski powiat lubartowski gmina Firlej 3

26985 powiat krasnostawski powiat krasnostawski gmina ÂŻĂłÂłkiewka 2

8248 Lublin powiat Lublin Lublin 3

24187 powiat puÂławski powiat puławski gmina KurĂłw 4

12485 Lublin powiat Lublin Lublin 3

14323 powiat rycki powiat rycki gmina KÂłoczew 4

**coords.x1 coords.x2 LEGAL\_FORM1 LEGAL\_FORM2 OWNERSHIP PKD7 GR\_EMPL**

10677 22.47410 51.51694 9 99 214 0150Z 1

26985 22.85254 50.88325 9 99 214 4711Z 1

8248 22.57219 51.21366 9 99 214 0150Z 1

24187 22.18994 51.39580 9 99 214 1392Z 1

12485 22.53744 51.23922 9 99 214 0150Z 1

14323 22.02138 51.72837 9 99 214 0141Z 1

**empl roa\_ind roa\_sec roa\_geo roa\_param roa sA sB sC sD sE sF sG**

10677 5.000519 2 2.0 0.0 2.0 1.983343 1 0 0 0 0 0 0

26985 5.000790 5 8.0 0.0 8.0 8.042440 0 0 0 0 0 0 1

8248 4.999808 2 2.0 1.5 3.5 3.503759 1 0 0 0 0 0 0

24187 5.000745 3 3.5 0.0 3.5 3.508910 0 0 1 0 0 0 0

12485 5.000063 2 2.0 1.5 3.5 3.481518 1 0 0 0 0 0 0

14323 4.998978 2 2.0 0.0 2.0 1.987931 1 0 0 0 0 0 0

**sH sI sJ sK sL sM sN sO sP sQ sR sS agri prod constr**

10677 0 0 0 0 0 0 0 0 0 0 0 0 1 -0.0003436035 0.0004591658

26985 0 0 0 0 0 0 0 0 0 0 0 0 0 -0.0011445054 0.0001216126

8248 0 0 0 0 0 0 0 0 0 0 0 0 1 -0.0000505683 0.0003509886

24187 0 0 0 0 0 0 0 0 0 0 0 0 0 0.9995378867 -0.0006009576

12485 0 0 0 0 0 0 0 0 0 0 0 0 1 0.0000105884 0.0002562051

14323 0 0 0 0 0 0 0 0 0 0 0 0 1 -0.0004311137 0.0005874865

**serv dist los los2 los3**

10677 -1.028717e-04 30.389969 5.410193e-05 5.188503e-04 -0.0003436035

26985 9.993726e-01 45.456709 6.981613e-05 7.902950e-04 -0.0011445054

8248 9.137862e-05 4.061494 9.868899e-05 -1.923529e-04 -0.0000505683

24187 -7.336136e-05 30.868211 1.219867e-04 7.454510e-04 -0.0004621133

12485 -1.830129e-04 2.367233 1.656434e-04 6.329818e-05 0.0000105884

14323 -4.945271e-04 65.322113 2.137653e-04 -1.021590e-03 -0.0004311137

**los4 los5 los6**

10677 0.0004591658 -1.028717e-04 -2.936005e-04

26985 0.0001216126 -6.274238e-04 4.881613e-04

8248 0.0003509886 9.137862e-05 2.654125e-04

24187 -0.0006009576 -7.336136e-05 7.234496e-04

12485 0.0002562051 -1.830129e-04 -1.090864e-05

14323 0.0005874865 -4.945271e-04 -5.323068e-04

## # estimation of the full model (for comparison)

**# This code runs full sample models (OLS, SEM, SAR, SDM) and saves the results info files**

**# full OLS, SEM, SAR & SDM models**

eq<-roa~empl+prod+constr+serv+dist # equation

danex<-dane1.in # dataset

crds<-as.matrix(dane1.in[ ,57:58]) # 57:58 are spatial coordinates

**# spatial weights matrix k nearest neighbours knn=5**

pkt.k.sym.listw<-nb2listw(make.sym.nb(knn2nb(knearneigh(crds, k=5))))

**# estimation of models**

model.ols.full<-lm(eq, data=danex)

model.sem.full<-errorsarlm(eq, data=danex, pkt.k.sym.listw, method="LU")

model.sar.full<-lagsarlm(eq, data=danex, pkt.k.sym.listw, method="LU")

model.sdm.full<-lagsarlm(eq, data=danex, pkt.k.sym.listw, method="LU", type="mixed")

**# summary of models**

summary(model.ols.full)

summary(model.sem.full, Nagelkerke=TRUE)

summary(model.sar.full, Nagelkerke=TRUE)

summary(model.sdm.full, Nagelkerke=TRUE)

**# saving full regression results in txt files**

m<-summary(model.ols.full)

save(m, file="model\_ols\_full.RData")

m<-summary(model.sem.full)

save(m, file="model\_sem\_full.RData")

m<-summary(model.sar.full)

save(m, file="model\_sar\_full.RData")

m<-summary(model.sdm.full)

save(m, file="model\_sdm\_full.RData")

**# 🡪 and reading them back if needed**

model.ols.full<-get(load("model\_ols\_full.RData"))

model.sem.full<-get(load("model\_sem\_full.RData"))

model.sar.full<-get(load("model\_sar\_full.RData"))

model.sdm.full<-get(load("model\_sdm\_full.RData"))

## # bootstrap simulation to find the best model

# This code runs bootstrap simulation – each of four models (OLS, SEM, SAR, SDM) is estimated many times (as n.col = number of iterations). Each iteration uses a different data subset. All results are saved into objects.

**# This may take ca. 10 minutes with parameters as specified below**

############################################

**# parameters of simulation – change here your parameters – now very limited just to run quickly the model**

n.col<-50 # number of iterations

n.row<-800 # number of observations in a sample

nnew<-10 # number of new points in the forecast

############################################

**# full procedure – bootstrapped models**

############################################

**# selector matrix – with k-means clustering**

**# assigning points to clusters with k-means – to get irregular shapes for sampling**

**# visualisation of result – see Fig.5**

groups<-kmeans(dane1.in[,57:58], n.row/100)

dane1.in$kmean<-groups$cluster

**# selector selects the rows of observations for every iteration and saves in matrix**

**# later it will be used to recover, on which data the best model was estimated**

**# strata() from sampling:: package samples observations from groups (clusters)**

**# method="srswor" stands for simple random sampling without replacement (in given column)**

selector<-matrix(0, nrow=n.row, ncol=n.col)

for(i in 1:n.col){

vec<-sample(1:dim(dane1.in)[1], n.row, replace=FALSE)

x<-**strata**(dane1.in, "kmean", size=rep(100, times=n.row/100), method="srswor") # from sampling::

selector[,i]<-x$ID\_unit}

selector[1:5, 1:5] # matrix with randomly selected numbers, n.row x n.col

# [,1] [,2] [,3] [,4] [,5]

#[1,] 63 615 2 133 312

#[2,] 317 685 537 157 566

#[3,] 977 1265 593 527 569

#[4,] 1057 1432 864 634 935

#[5,] 1062 1448 1212 663 1205

#####################################

**# all models – objects to store results of iterations**

coef.ols<-matrix(0, nrow=n.col, ncol=6) # for coefficients

coef.sem<-matrix(0, nrow=n.col, ncol=6)

coef.sar<-matrix(0, nrow=n.col, ncol=6)

coef.sdm<-matrix(0, nrow=n.col, ncol=11)

error.ols<-matrix(0, nrow=n.col, ncol=6) # for standard errors

error.sem<-matrix(0, nrow=n.col, ncol=6)

error.sar<-matrix(0, nrow=n.col, ncol=6)

error.sdm<-matrix(0, nrow=n.col, ncol=11)

sig.ols<-matrix(0, nrow=n.col, ncol=6)

fitted.ols<-matrix(0, nrow=n.row, ncol=n.col) # for fitted values

fitted.sem<-matrix(0, nrow=n.row, ncol=n.col)

fitted.sar<-matrix(0, nrow=n.row, ncol=n.col)

fitted.sdm<-matrix(0, nrow=n.row, ncol=n.col)

quality<-matrix(0, nrow=n.col, ncol=5) # R2, AIC.ols, AIC.sem, AIC.sar, AIC.sdm

time<-matrix(0, nrow=n.col, ncol=5) # time.ols, time.W, time.sem, time.sar, time.sdm

roa<-matrix(0, nrow=n.row, ncol=n.col) # retriving original values of y

spatial<-matrix(0, nrow=n.col, ncol=3) # lambda.sem, rho.sar, rho.sdm

eq<-roa~empl+prod+constr+serv+dist # model structure

**# big loop – estimation of all models (OLS, SEM, SAR, SDM) on the same subsets with time measurement**

for(i in 1:n.col){

danex<-dane1.in[selector[,i],]

roa[,i]<-danex$roa # saving y values for each iteration in roa matrix

**# W matrix**

crds<-as.matrix(dane1.in[selector[,i], 57:58]) # spatial weights matrix W

start.time <- Sys.time()

pkt.k.sym.listw<-nb2listw(make.sym.nb(knn2nb(knearneigh(crds, k=5)))) # knn=5

end.time <- Sys.time()

time.W<- difftime(end.time, start.time, units="secs")

**# OLS model**

start.time <- Sys.time()

model.ols<-lm(eq, data=danex)

end.time <- Sys.time()

time.ols<- difftime(end.time, start.time, units="secs")

**# SEM model**

start.time <- Sys.time()

model.sem<-errorsarlm(eq, data=danex, pkt.k.sym.listw, method="LU")

end.time <- Sys.time()

time.sem<- difftime(end.time, start.time, units="secs")

**# SAR model**

start.time <- Sys.time()

model.sar<-lagsarlm(eq, data=danex, pkt.k.sym.listw, method="LU")

end.time <- Sys.time()

time.sar<- difftime(end.time, start.time, units="secs")

**# SDM model**

start.time <- Sys.time()

model.sdm<-lagsarlm(eq, data=danex, pkt.k.sym.listw, method="LU", type="mixed")

end.time <- Sys.time()

time.sdm<- difftime(end.time, start.time, units="secs")

**# saving the results into appropriate objects**

coef.ols[i,]<-summary(model.ols)$coefficients[,1]

error.ols[i,]<-summary(model.ols)$coefficients[,2]

sig.ols[i,]<-summary(model.ols)$coefficients[,4]

fitted.ols[,i]<-model.ols$fitted.values

coef.sem[i,]<-model.sem$coefficients

error.sem[i,]<-model.sem$rest.se

fitted.sem[,i]<-model.sem$fitted.values

coef.sar[i,]<-model.sar$coefficients

error.sar[i,]<-model.sar$rest.se

fitted.sar[,i]<-model.sar$fitted.values

coef.sdm[i,]<-model.sdm$coefficients

error.sdm[i,]<-model.sdm$rest.se

fitted.sdm[,i]<-model.sdm$fitted.values

quality[i,1]<-summary(model.ols)$r.squared # R2

quality[i,2]<-AIC(model.ols) # AIC.ols

quality[i,3]<-AIC(model.sem) # AIC.sem

quality[i,4]<-AIC(model.sar) # AIC.sar

quality[i,5]<-AIC(model.sdm) # AIC.sdm

time[i,1]<-time.ols

time[i,2]<-time.W

time[i,3]<-time.sem

time[i,4]<-time.sar

time[i,5]<-time.sdm

spatial[i,1]<-model.sem$lambda

spatial[i,2]<-model.sar$rho

spatial[i,3]<-model.sdm$rho

}

# each model (OLS, SAR, SEM, SDM) was estimated 50 times

head(coef.ols)

# [,1] [,2] [,3] [,4] [,5] [,6]

#[1,] 2.437616 0.0020284324 1.578931 3.006216 6.174388 -0.006400216

#[2,] 2.398009 0.0005306455 1.735549 3.049162 6.140863 -0.005760920

#[3,] 2.479301 -0.0003706600 1.737262 3.067651 6.173159 -0.006832407

#[4,] 2.492294 -0.0008471774 1.560847 2.978469 6.166155 -0.006828261

#[5,] 2.360972 0.0012591673 1.652750 2.967866 6.123976 -0.005223608

#[6,] 2.456224 -0.0006022282 1.658252 3.080859 6.150532 -0.006436162

head(time)

# [,1] [,2] [,3] [,4] [,5]

#[1,] 0.004276991 0.2657180 2.189982 3.173867 3.211553

#[2,] 0.002266884 0.2149570 2.127379 3.553725 3.363792

#[3,] 0.002393007 0.2408359 2.188978 3.327949 3.306635

#[4,] 0.002336979 0.2149439 2.088798 3.439352 3.406646

#[5,] 0.002411127 0.2539051 2.148157 3.269229 3.482678

#[6,] 0.002171993 0.2128470 2.105742 3.725344 3.992017

head(quality)

# [,1] [,2] [,3] [,4] [,5]

#[1,] 0.9880292 415.5217 -1025.8992 185.4271 -1042.1972

#[2,] 0.9887564 338.7113 -793.2285 163.7461 -803.4621

#[3,] 0.9873993 433.9909 -933.8936 214.2263 -939.4232

#[4,] 0.9879185 450.1025 -1029.9045 230.8638 -1033.9413

#[5,] 0.9892614 324.7827 -643.6447 188.6026 -647.3813

#[6,] 0.9877178 418.1233 -1003.2167 207.2063 -1007.3604

## # selection of the best model with PAM

# This code analyses bootstrap models and looks for the most central (medoid) one

# for clustering, one uses both beta coefficients and spatial coefficients (added with cbind() function)

# One always assumes one cluster only

**## PAM for OLS**

c1.ols<-pam(coef.ols,1) #cluster::pam(), works for n<65536, clustering all data into one cluster

summary(c1.ols)

c1.ols$clustering # clustering vector – only 1 values

c1.ols$medoids # coefficients of selected best model

c1.ols$id.med # which iteration (model) is most representative

#hopkins(coef.ols, n=nrow(coef.ols)-1) # Hopkins statistics to check clusterability, takes long…

**## PAM for SEM**

c1.sem<-pam(cbind(coef.sem, spatial[,1]),1) #cluster::pam(), works for n<65536

summary(c1.sem)

c1.sem$clustering # clustering vector – only 1 values

c1.sem$medoids # coefficients of selected best model

c1.sem$id.med # which iteration (model) is most representative

#hopkins(coef.sem, n=nrow(coef.sem)-1) # takes long…

**## PAM for SAR**

c1.sar<-pam(cbind(coef.sar, spatial[,2]),1) #cluster::pam(), works for n<65536

summary(c1.sar)

c1.sar$clustering

c1.sar$medoids

c1.sar$id.med

#hopkins(coef.sar, n=nrow(coef.sar)-1) #

**## PAM for SDM**

c1.sdm<-pam(cbind(coef.sdm, spatial[,3]),1) #cluster::pam(), works for n<65536

summary(c1.sdm)

c1.sdm$clustering

c1.sdm$medoids

c1.sdm$id.med

#hopkins(coef.sem, n=nrow(coef.sem)-1)

## # validation for out-of-sample

# For each model (OLS, SEM, SAR, SDM) procedure is as follows:

# One takes the best PAM model and observations used in its estimation

# On those observations, one runs tessellation (Voronoi polygons)

# On the best model, one runs the prediction – for a single new point in each prediction round

# For forecasts of all new points, one calculates RAMSE

### # OLS predictions

**# tessellation of observations from the selected best model**

points<-data.frame(x=dane1.in[selector[,c1.ols$id.med],57], y=dane1.in[selector[,c1.ols$id.med],58])

points.sf<-st\_as\_sf(points, coords=c("x", "y"), crs=4326, agr ="constant") # points in sf class

points.sfc<-st\_geometry(points.sf)

box.sfc<-st\_geometry(WOJ.lub)

points.union<-st\_union(points.sfc)

tess<-st\_voronoi(points.union, box.sfc)

tess.clip<-st\_intersection(st\_cast(tess), st\_union(box.sfc))

plot(tess.clip) # plotting tessellated area

points(points, pch=".", col="darkblue", cex=2) # adding points in blue

**# restoring the medoid model**

dane.x<-dane1.in[selector[,c1.ols$id.med],] # we choose the data which were used in estimation of the best model

crds<-as.matrix(dane.x[, 57:58]) # to create W

RAMSE.med.ols<-(sum((dane1.in[selector[,c1.ols$id.med], 26]-fitted.ols[,c1.ols$id.med])^2)/n.row)^(0.5)

RAMSE.med.ols

pkt.k.sym.listw<-nb2listw(make.sym.nb(knn2nb(knearneigh(crds, k=5)))) # knn W (for estimation and/or Moran)

eq<-roa~empl+prod+constr+serv+dist # regression equation

model.ols<-lm(eq, data=dane.x) # ols model

moran.test(model.ols$residuals, pkt.k.sym.listw) # Moran test for OLS residuals for spatial autocorrelation

**# selection of out-of-sample points for predictions**

nnew<-100 # number of new points in the forecast

points.pred<-dane1.out[1:nnew, c(1:2, 57:58)] # selection of a new points

points.pred.sf<-st\_as\_sf(points.pred, coords=c("xxe", "yye"), crs=4326, agr ="constant") # points in sf class

tess.sf<-st\_sf(tess.clip)

a1<-st\_intersects(points.pred.sf, tess.sf) # new version of over()

**# graphical checking if point overlaps with tessellation tile**

#i=3

#plot(tess.clip)# plotting tessellated area

#plot(tess.clip[unlist(a1[i])], add=TRUE, col="red")# plotting tessellated area

#points(points.pred[i,3:4], pch=21, bg="yellow", cex=2)

**# object for new points**

forecasts1<-matrix(0, nrow=nnew, ncol=5)

colnames(forecasts1)<-c("predicted y","real y","crds x","crds y", "diff")

**# loop for forecasts for new points**

**# there is a separate match for each point**

for(i in 1:nnew){

# point by point - assigning new data to the old data set

dane.x.new<-dane.x

xxx<-dane1.out[i, ]

dane.x.new[unlist(a1[i]),]<-xxx

rownames(dane.x.new)<-1:dim(dane.x.new)[1]

# prediction for out-of-sample calibrated SDM model

pred<-predict(model.ols, newdata=dane.x.new)

pred[unlist(a1[i])] # prediction for a new point

xxx[,26] # empirical y value of the new point

forecasts1[i,1]<- pred[unlist(a1[i])] # predicted value of y

forecasts1[i,2]<- xxx[, 26] # empirical value of y

forecasts1[i,3]<-xxx[, 57] # x coordinates

forecasts1[i,4]<-xxx[, 58] # y coordinates

}

forecasts1[,5]<-(forecasts1[,1]-forecasts1[,2])^2

RAMSE.ols<-(mean(forecasts1[,5]))^0.5

head(forecasts1)

RAMSE.ols # quality of predictions

##########################

### # SEM predictions

**# tessellation of observations from the selected best model**

points<-data.frame(x=dane1.in[selector[,c1.sem$id.med],57], y=dane1.in[selector[,c1.ols$id.med],58])

points.sf<-st\_as\_sf(points, coords=c("x", "y"), crs=4326, agr ="constant") # points in sf class

points.sfc<-st\_geometry(points.sf)

box.sfc<-st\_geometry(WOJ.lub)

points.union<-st\_union(points.sfc)

tess<-st\_voronoi(points.union, box.sfc)

tess.clip<-st\_intersection(st\_cast(tess), st\_union(box.sfc))

plot(tess.clip) # plotting tessellated area

points(points, pch=".", col="darkblue", cex=2) # adding points in blue

**# restoring the medoid model**

dane.x<-dane1.in[selector[,c1.sem$id.med],] # we choose the data which were used in estimation of the best model

crds<-as.matrix(dane.x[, 57:58]) # to create W

RAMSE.med.sem<-(sum((dane1.in[selector[,c1.sem$id.med], 26]-fitted.ols[,c1.sem$id.med])^2)/n.row)^(0.5)

RAMSE.med.sem

pkt.k.sym.listw<-nb2listw(make.sym.nb(knn2nb(knearneigh(crds, k=5)))) # knn W (for estimation and/or Moran)

eq<-roa~empl+prod+constr+serv+dist # regression equation

model.sem<-errorsarlm(eq, data=dane.x, pkt.k.sym.listw, method="LU")

moran.test(model.sem$residuals, pkt.k.sym.listw)

**# selection of out-of-sample points for predictions**

nnew<-100 # number of new points in the forecast

points.pred<-dane1.out[1:nnew, c(1:2, 57:58)] # selection of a new points

points.pred.sf<-st\_as\_sf(points.pred, coords=c("xxe", "yye"), crs=4326, agr ="constant") # points in sf class

tess.sf<-st\_sf(tess.clip)

a1<-st\_intersects(points.pred.sf, tess.sf) # new version of over()

**# graphical checking if point overlaps with tessellation tile**

#i=3

#plot(tess.clip)# plotting tessellated area

#plot(tess.clip[unlist(a1[i])], add=TRUE, col="red")# plotting tessellated area

#points(points.pred[i,3:4], pch=21, bg="yellow", cex=2)

**# object for new points**

forecasts1<-matrix(0, nrow=nnew, ncol=5)

colnames(forecasts1)<-c("predicted y","real y","crds x","crds y", "diff")

**# loop for forecasts for new points**

**# there is a separate match for each point**

for(i in 1:nnew){

# point by point - assigning new data to the old data set

dane.x.new<-dane.x

xxx<-dane1.out[i, ]

dane.x.new[unlist(a1[i]),]<-xxx

rownames(dane.x.new)<-1:dim(dane.x.new)[1]

# prediction for out-of-sample calibrated SDM model

pred<-predict(model.sem, newdata=dane.x.new, listw=pkt.k.sym.listw, legacy.mixed=TRUE)

pred[unlist(a1[i])] # prediction for a new point

xxx[,26] # empirical y value of the new point

forecasts1[i,1]<- pred[unlist(a1[i])] # predicted value of y

forecasts1[i,2]<- xxx[, 26] # empirical value of y

forecasts1[i,3]<-xxx[, 57] # x coordinates

forecasts1[i,4]<-xxx[, 58] # y coordinates

}

forecasts1[,5]<-(forecasts1[,1]-forecasts1[,2])^2

RAMSE.sem<-(mean(forecasts1[,5]))^0.5

head(forecasts1)

RAMSE.sem # quality of predictions

#############################

### # SAR predictions

**# tessellation of observations from the selected best model**

points<-data.frame(x=dane1.in[selector[,c1.sar$id.med],57], y=dane1.in[selector[,c1.ols$id.med],58])

points.sf<-st\_as\_sf(points, coords=c("x", "y"), crs=4326, agr ="constant") # points in sf class

points.sfc<-st\_geometry(points.sf)

box.sfc<-st\_geometry(WOJ.lub)

points.union<-st\_union(points.sfc)

tess<-st\_voronoi(points.union, box.sfc)

tess.clip<-st\_intersection(st\_cast(tess), st\_union(box.sfc))

plot(tess.clip) # plotting tessellated area

points(points, pch=".", col="darkblue", cex=2) # adding points in blue

**# restoring the medoid model**

dane.x<-dane1.in[selector[,c1.sar$id.med],] # we choose the data which were used in estimation of the best model

crds<-as.matrix(dane.x[, 57:58]) # to create W

RAMSE.med.sar<-(sum((dane1.in[selector[,c1.sar$id.med], 26]-fitted.ols[,c1.sar$id.med])^2)/n.row)^(0.5)

RAMSE.med.sar

pkt.k.sym.listw<-nb2listw(make.sym.nb(knn2nb(knearneigh(crds, k=5)))) # knn W (for estimation and/or Moran)

eq<-roa~empl+prod+constr+serv+dist # regression equation

model.sar<-lagsarlm(eq, data=dane.x, pkt.k.sym.listw, method="LU")

moran.test(model.sar$residuals, pkt.k.sym.listw)

**# selection of out-of-sample points for predictions**

nnew<-100 # number of new points in the forecast

points.pred<-dane1.out[1:nnew, c(1:2, 57:58)] # selection of a new points

points.pred.sf<-st\_as\_sf(points.pred, coords=c("xxe", "yye"), crs=4326, agr ="constant") # points in sf class

tess.sf<-st\_sf(tess.clip)

a1<-st\_intersects(points.pred.sf, tess.sf) # new version of over()

**# graphical checking if point overlaps with tessellation tile**

#i=3

#plot(tess.clip)# plotting tessellated area

#plot(tess.clip[unlist(a1[i])], add=TRUE, col="red")# plotting tessellated area

#points(points.pred[i,3:4], pch=21, bg="yellow", cex=2)

**# object for new points**

forecasts1<-matrix(0, nrow=nnew, ncol=5)

colnames(forecasts1)<-c("predicted y","real y","crds x","crds y", "diff")

**# loop for forecasts for new points**

**# there is a separate match for each point**

for(i in 1:nnew){

# point by point - assigning new data to the old data set

dane.x.new<-dane.x

xxx<-dane1.out[i, ]

dane.x.new[unlist(a1[i]),]<-xxx

rownames(dane.x.new)<-1:dim(dane.x.new)[1]

# prediction for out-of-sample calibrated SDM model

pred<-predict(model.sar, newdata=dane.x.new, listw=pkt.k.sym.listw, legacy.mixed=TRUE)

pred[unlist(a1[i])] # prediction for a new point

xxx[,26] # empirical y value of the new point

forecasts1[i,1]<- pred[unlist(a1[i])] # predicted value of y

forecasts1[i,2]<- xxx[, 26] # empirical value of y

forecasts1[i,3]<-xxx[, 57] # x coordinates

forecasts1[i,4]<-xxx[, 58] # y coordinates

}

forecasts1[,5]<-(forecasts1[,1]-forecasts1[,2])^2

RAMSE.sar<-(mean(forecasts1[,5]))^0.5

head(forecasts1)

RAMSE.sar # quality of predictions

#########################################

### # SDM predictions

**# tessellation of observations from the selected best model**

points<-data.frame(x=dane1.in[selector[,c1.sdm$id.med],57], y=dane1.in[selector[,c1.ols$id.med],58])

points.sf<-st\_as\_sf(points, coords=c("x", "y"), crs=4326, agr ="constant") # points in sf class

points.sfc<-st\_geometry(points.sf)

box.sfc<-st\_geometry(WOJ.lub)

points.union<-st\_union(points.sfc)

tess<-st\_voronoi(points.union, box.sfc)

tess.clip<-st\_intersection(st\_cast(tess), st\_union(box.sfc))

plot(tess.clip) # plotting tessellated area

points(points, pch=".", col="darkblue", cex=2) # adding points in blue

**# restoring the medoid model**

dane.x<-dane1.in[selector[,c1.sdm$id.med],] # we choose the data which were used in estimation of the best model

crds<-as.matrix(dane.x[, 57:58]) # to create W

RAMSE.med.sdm<-(sum((dane1.in[selector[,c1.sdm$id.med], 26]-fitted.ols[,c1.sdm$id.med])^2)/n.row)^(0.5)

RAMSE.med.sdm

pkt.k.sym.listw<-nb2listw(make.sym.nb(knn2nb(knearneigh(crds, k=5)))) # knn W (for estimation and/or Moran)

eq<-roa~empl+prod+constr+serv+dist # regression equation

model.sdm<-lagsarlm(eq, data=dane.x, pkt.k.sym.listw, method="LU", type="mixed")

moran.test(model.sdm$residuals, pkt.k.sym.listw)

**# selection of out-of-sample points for predictions**

nnew<-100 # number of new points in the forecast

points.pred<-dane1.out[1:nnew, c(1:2, 57:58)] # selection of a new points

points.pred.sf<-st\_as\_sf(points.pred, coords=c("xxe", "yye"), crs=4326, agr ="constant") # points in sf class

tess.sf<-st\_sf(tess.clip)

a1<-st\_intersects(points.pred.sf, tess.sf) # new version of over()

**# graphical checking if point overlaps with tessellation tile**

#i=3

#plot(tess.clip)# plotting tessellated area

#plot(tess.clip[unlist(a1[i])], add=TRUE, col="red")# plotting tessellated area

#points(points.pred[i,3:4], pch=21, bg="yellow", cex=2)

**# object for new points**

forecasts1<-matrix(0, nrow=nnew, ncol=5)

colnames(forecasts1)<-c("predicted y","real y","crds x","crds y", "diff")

**# loop for forecasts for new points**

**# there is a separate match for each point**

for(i in 1:nnew){

# point by point - assigning new data to the old data set

dane.x.new<-dane.x

xxx<-dane1.out[i, ]

dane.x.new[unlist(a1[i]),]<-xxx

rownames(dane.x.new)<-1:dim(dane.x.new)[1]

# prediction for out-of-sample calibrated SDM model

pred<-predict(model.sdm, newdata=dane.x.new, listw=pkt.k.sym.listw, legacy.mixed=TRUE)

pred[unlist(a1[i])] # prediction for a new point

xxx[,26] # empirical y value of the new point

forecasts1[i,1]<- pred[unlist(a1[i])] # predicted value of y

forecasts1[i,2]<- xxx[, 26] # empirical value of y

forecasts1[i,3]<-xxx[, 57] # x coordinates

forecasts1[i,4]<-xxx[, 58] # y coordinates

}

forecasts1[,5]<-(forecasts1[,1]-forecasts1[,2])^2

RAMSE.sdm<-(mean(forecasts1[,5]))^0.5

head(forecasts1)

RAMSE.sdm # quality of predictions

#########################################################

### # all RAMSE together – for models and for forecasts

RAMSE.ols

RAMSE.sem

RAMSE.sar

RAMSE.sdm

RAMSE.med.ols

RAMSE.med.sem

RAMSE.med.sar

RAMSE.med.sdm

###########################################

### # full sample SDM - to get RAMSE of forecast from the full-sample model

**# tessellation of observations from the selected best model**

points<-data.frame(x=dane1.in[,57], y=dane1.in[,58])

points.sf<-st\_as\_sf(points, coords=c("x", "y"), crs=4326, agr ="constant") # points in sf class

points.sfc<-st\_geometry(points.sf)

box.sfc<-st\_geometry(WOJ.lub)

points.union<-st\_union(points.sfc)

tess<-st\_voronoi(points.union, box.sfc)

tess.clip<-st\_intersection(st\_cast(tess), st\_union(box.sfc))

plot(tess.clip) # plotting tessellated area

points(points, pch=".", col="darkblue", cex=2) # adding points in blue

**# restoring the medoid model**

dane.x<-dane1.in # we choose the data which were used in estimation of the best model

crds<-as.matrix(dane.x[, 57:58]) # to create W

pkt.k.sym.listw<-nb2listw(make.sym.nb(knn2nb(knearneigh(crds, k=5)))) # knn W (for estimation and/or Moran)

eq<-roa~empl+prod+constr+serv+dist # regression equation

model.sdm.full<-lagsarlm(eq, data=dane.x, pkt.k.sym.listw, method="LU", type="mixed")

moran.test(model.sdm.full$residuals, pkt.k.sym.listw)

**# selection of out-of-sample points for predictions**

nnew<-100 # number of new points in the forecast

points.pred<-dane1.out[1:nnew, c(1:2, 57:58)] # selection of a new points

points.pred.sf<-st\_as\_sf(points.pred, coords=c("xxe", "yye"), crs=4326, agr ="constant") # points in sf class

tess.sf<-st\_sf(tess.clip)

a1<-st\_intersects(points.pred.sf, tess.sf) # new version of over()

**# graphical checking if point overlaps with tessellation tile**

#i=3

#plot(tess.clip)# plotting tessellated area

#plot(tess.clip[unlist(a1[i])], add=TRUE, col="red")# plotting tessellated area

#points(points.pred[i,3:4], pch=21, bg="yellow", cex=2)

**# object for new points**

forecasts1<-matrix(0, nrow=nnew, ncol=5)

colnames(forecasts1)<-c("predicted y","real y","crds x","crds y", "diff")

**# loop for forecasts for new points**

**# there is a separate match for each point**

for(i in 1:nnew){

# point by point - assigning new data to the old data set

dane.x.new<-dane.x

xxx<-dane1.out[i, ]

dane.x.new[unlist(a1[i]),]<-xxx

rownames(dane.x.new)<-1:dim(dane.x.new)[1]

# prediction for out-of-sample calibrated SDM model

pred<-predict(model.sdm.full, newdata=dane.x.new, listw=pkt.k.sym.listw, legacy.mixed=TRUE)

forecasts1[i,1]<- pred[unlist(a1[i])] # predicted value of y for a new point

forecasts1[i,2]<- xxx[, 26] # empirical value of y for a new point

forecasts1[i,3]<-xxx[, 57] # x coordinates

forecasts1[i,4]<-xxx[, 58] # y coordinates

}

forecasts1[,5]<-(forecasts1[,1]-forecasts1[,2])^2

RAMSE.sdm.full<-(mean(forecasts1[,5]))^0.5

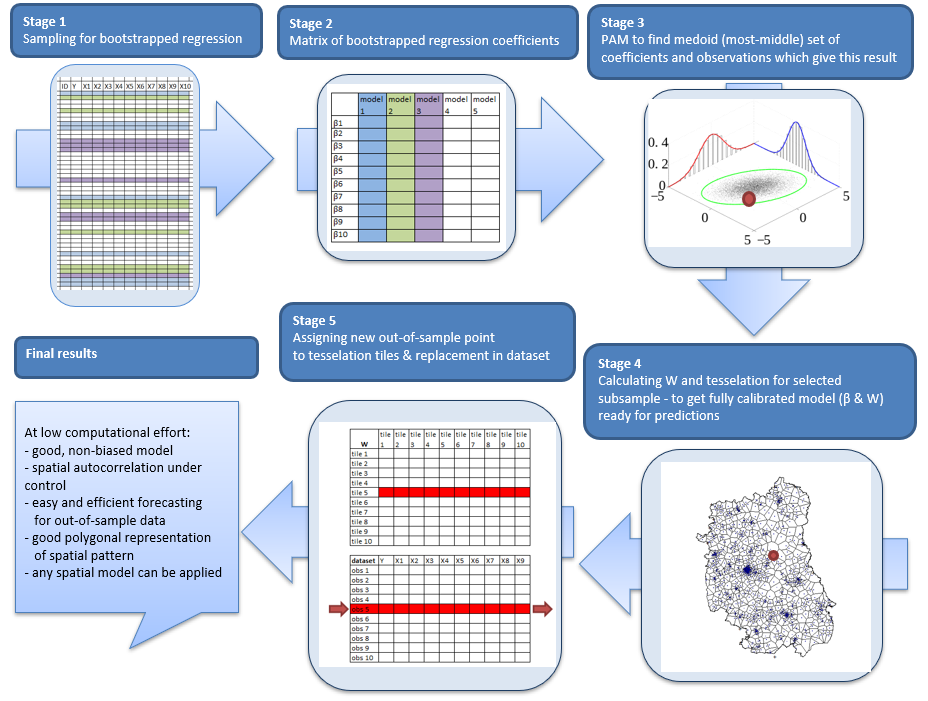
head(forecasts1)

RAMSE.sdm.full # quality of predictions

# # FIGURES FROM THE TEXT

## # Figure 1

**# This plot was produced outside R**



## # Figure 2a

**# This plot presents analysed data in geographical context – with coordinates and neighbours**

**# It uses Open Data for boundaries of Poland, Belarus and Ukraine**

**# reading additional shapefile maps of administrative borders**

ukr<-readOGR(".", "UKR\_adm0") # 16 jedn.

ukr<- spTransform(ukr, CRS("+proj=longlat +datum=NAD83"))

blr<-readOGR(".", "BLR\_adm0") # 16 jedn.

blr<- spTransform(blr, CRS("+proj=longlat +datum=NAD83"))

pol<-readOGR(".", "POL\_adm0") # 16 jedn.

pol<- spTransform(pol, CRS("+proj=longlat +datum=NAD83"))

**# cutting map to select interesting lubelskie region**

woj.df<-as.data.frame(woj)

woj.lub<-woj[woj.df$jpt\_nazwa\_=="lubelskie",]

**# plot**

plot(woj.lub, xlim=c(21, 24.3))

points(dane[,12:13], pch=".", cex=1.5)

degAxis(1) # using sp::

degAxis(2)

plot(gridlines(woj.lub), add = TRUE) # using sp::

maps::map.scale(x=20.91, y=52.3, ratio=FALSE, relwidth=0.2) #using maps::

compassRose(21.32, 51.8,rot=0,cex=1) # using sp::

plot(woj, add=TRUE)

text(23.75, 52.3, label="Belarus", font=2)

text(23.96, 51.3, label="Ukraine", font=2)

text(21.3, 51.3, label="Mazovian

region", font=3)

text(21.3, 50.7, label="Świętokrzyskie

region", font=3)

text(21.64, 50.3, label="Podkarpackie

region", font=3)

plot(ukr, add=TRUE, lwd=2, xlim=c(21, 24))

plot(blr, add=TRUE, lwd=2, xlim=c(21, 24))

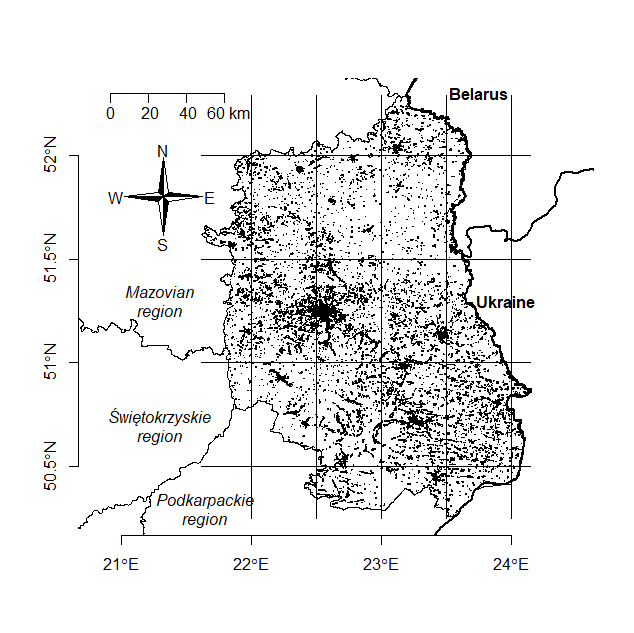
plot(pol, add=TRUE, lwd=2)

savePlot(filename="Fig01", type="jpg")

savePlot(filename="Fig01", type="tif")

savePlot(filename="Fig01", type="png")

**savePlot(filename="Fig01", type="eps") # best quality**



## # Figure 2b

**# This plot presents dependent variable of analysed data (profitability of firms from different sectors) in their geolocation**

library(classInt) # for nice legend in the plot

locs<-dane1[,57:58] # locations of points (geocoordinates)

variable<-dane1$roa # dependent variable – roa (profitability) of firms

summary(variable)

bins<-6 # intervals for analysis

brks<-c(1, 2.5, 4, 5.5, 7, 8.5, 10) # fixed breaks of intervals

colors<-brewer.pal(bins, "Reds") # selection of colours – Reds, from rColorBrewer

classes<-classIntervals(variable, bins, style="fixed", fixedBreaks=brks) # changing data into colours

table.of.colors<-findColours(classes, colors) # info on colours assigned to data

par(mar=c(1,1,1,1)) # narrow margins

plot(woj[woj@data$jpt\_nazwa\_=="lubelskie",], border="grey70") # contour of region

points(locs, col=table.of.colors, pch=21, cex=0.45, bg=table.of.colors) # points with colours

legend("bottomleft", legend=names(attr(table.of.colors, "table")), fill=attr(table.of.colors, "palette"), cex=1, bty="n") # well-formatted legend

#round(table(cut(dane1$roa, brks))/37374,2) # distribution of values by intervals

#(1,2.5] (2.5,4] (4,5.5] (5.5,7] (7,8.5] (8.5,10]

# 0.54 0.05 0.05 0.01 0.25 0.10

text(22, 50.64, "54%") # adding the text next to legend

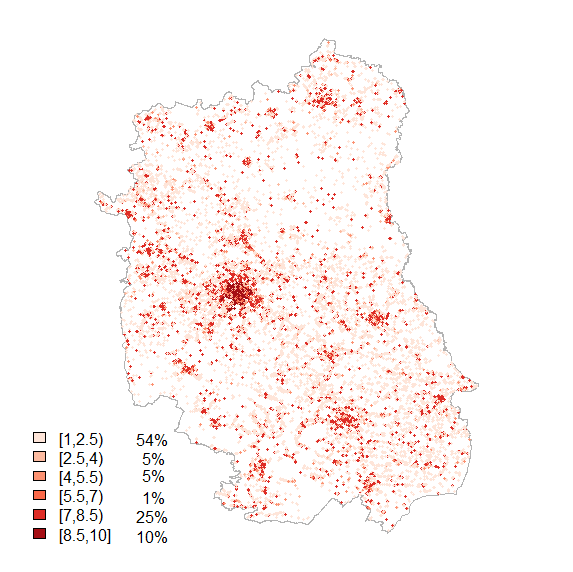
text(22, 50.56, " 5%")

text(22, 50.49, " 5%")

text(22, 50.40, " 1%")

text(22, 50.32, "25%")

text(22, 50.24, "10%")



## # Figure 3

# It presents the estimation results for an increasing number of observations (expanding window). It summarises 37 regressions with the step of 1000 obs. (for 1.000, 2.000, 3.000, ….30.000, …., 36.000, 37.000 obs.)

**# simulation code below takes ca. 1 hour to get all estimates**

**# alternatively one can upload results from** [**https://app.sugarsync.com/iris/wf/D1836703\_09668593\_7523736**](https://app.sugarsync.com/iris/wf/D1836703_09668593_7523736)

**# please note that simulation might slightly differ from saved results due to randomness inherited in code**

coef.OLS<-read.csv("coefOLS.csv", sep=";", dec=",", header=TRUE)

coef.SAR<-read.csv("coefSAR.csv", sep=";", dec=",", header=TRUE)

coef.SDM<-read.csv("coefSDM.csv", sep=";", dec=",", header=TRUE)

coef.SEM<-read.csv("coefSEM.csv", sep=";", dec=",", header=TRUE)

error.OLS<-read.csv("errorOLS.csv", sep=";", dec=",", header=TRUE)

error.SAR<-read.csv("errorSAR.csv", sep=";", dec=",", header=TRUE)

error.SDM<-read.csv("errorSDM.csv", sep=";", dec=",", header=TRUE)

error.SEM<-read.csv("errorSEM.csv", sep=";", dec=",", header=TRUE)

other.OLS<-read.csv("otherOLS.csv", sep=";", dec=",", header=TRUE)

other.SAR<-read.csv("otherSAR.csv", sep=";", dec=",", header=TRUE)

other.SDM<-read.csv("otherSDM.csv", sep=";", dec=",", header=TRUE)

other.SEM<-read.csv("otherSEM.csv", sep=";", dec=",", header=TRUE)

**# simulation – to replicate calculations saved in files above**

**# general parameters**

ITER<-37 # how many models to estimate, observations/STEP

STEP<-1000 # step, how many more observations in next iteration

eq<-roa~empl+prod+constr+serv+dist #equation to estimate

**# set of SEM models**

# objects to save results

coef.sem<-matrix(0, nrow=7, ncol=ITER) # coefficients + lambda

error.sem<-matrix(0, nrow=6, ncol=ITER) # coefficients

other.sem<-matrix(0, nrow=2, ncol=ITER) # time + Moran

# loop for expanding window

for(i in 1:ITER)

{ # 1:37 by 1000 obs.

up<-i\*STEP

danex<-dane1[1:up,]

start.time<-Sys.time()

crds<-as.matrix(danex[,57:58])

pkt.k.sym.listw<-nb2listw(make.sym.nb(knn2nb(knearneigh(crds, k=5, longlat=NULL))))

model1<-errorsarlm(eq, data=danex, pkt.k.sym.listw, method="LU")

end.time<-Sys.time()

time.taken <- difftime(end.time, start.time, units="secs")

moran<-moran.test(model1$residuals, pkt.k.sym.listw)

coef.sem[1:6,i]<-model1$coefficients

coef.sem[7,i]<-model1$lambda

error.sem[,i]<-model1$rest.se

other.sem[1,i]<-time.taken

other.sem[2,i]<-moran$p.value

}

# saving objects with results

write.table(coef.sem, file="coefSEM.csv", sep = ",", col.names=TRUE)

write.table(error.sem, file="errorSEM.csv", sep = ",", col.names=TRUE)

write.table(other.sem, file="otherSEM.csv", sep = ",", col.names=TRUE)

**# set of SAR models**

# objects to save results

coef.sar<-matrix(0, nrow=7, ncol=ITER) # coefficients + rho

error.sar<-matrix(0, nrow=6, ncol=ITER) # coefficients

other.sar<-matrix(0, nrow=2, ncol=ITER) # time + Moran

# loop for expanding window

for(i in 1:ITER)

{ # 1:37 by 1000 obs.

up<-i\*STEP

danex<-dane1[1:up,]

start.time<-Sys.time()

crds<-as.matrix(danex[,57:58])

pkt.k.sym.listw<-nb2listw(make.sym.nb(knn2nb(knearneigh(crds, k=5, longlat=NULL))))

model1<-lagsarlm(eq, data=danex, pkt.k.sym.listw, method="LU")

end.time<-Sys.time()

time.taken <- difftime(end.time, start.time, units="secs")

moran<-moran.test(model1$residuals, pkt.k.sym.listw)

coef.sar[1:6,i]<-model1$coefficients

coef.sar[7,i]<-model1$rho

error.sar[,i]<-model1$rest.se

other.sar[1,i]<-time.taken

other.sar[2,i]<-moran$p.value

}

# saving objects with results

write.table(coef.sar, file="coefSAR.csv", sep = ",", col.names=TRUE)

write.table(error.sar, file="errorSAR.csv", sep = ",", col.names=TRUE)

write.table(other.sar, file="otherSAR.csv", sep = ",", col.names=TRUE)

**# set of SDM models**

# objects to save results

coef.sdm<-matrix(0, nrow=12, ncol=ITER) # coefficients + rho + thetas

error.sdm<-matrix(0, nrow=11, ncol=ITER) # coefficients

other.sdm<-matrix(0, nrow=2, ncol=ITER) # time + Moran

# loop for expanding window

for(i in 1:ITER)

{ # 1:37 by 1000 obs.

up<-i\*STEP

danex<-dane1[1:up,]

start.time<-Sys.time()

crds<-as.matrix(danex[,57:58])

pkt.k.sym.listw<-nb2listw(make.sym.nb(knn2nb(knearneigh(crds, k=5, longlat=NULL))))

model1<-lagsarlm(eq, data=danex, pkt.k.sym.listw, method="LU", type="mixed")

end.time<-Sys.time()

time.taken <- difftime(end.time, start.time, units="secs")

moran<-moran.test(model1$residuals, pkt.k.sym.listw)

coef.sdm[1:11,i]<-model1$coefficients

coef.sdm[12,i]<-model1$rho

error.sdm[,i]<-model1$rest.se

other.sdm[1,i]<-time.taken

other.sdm[2,i]<-moran$p.value

}

# saving objects with results

write.table(coef.sdm, file="coefSDM.csv", sep = ",", col.names=TRUE)

write.table(error.sdm, file="errorSDM.csv", sep = ",", col.names=TRUE)

write.table(other.sdm, file="otherSDM.csv", sep = ",", col.names=TRUE)

**# set of OLS models**

# objects to save results

coef.ols<-matrix(0, nrow=6, ncol=ITER) #coefficients

error.ols<-matrix(0, nrow=6, ncol=ITER)

other.ols<-matrix(0, nrow=3, ncol=ITER) # R2+time + Moran

# loop for expanding window

for(i in 1:ITER)

{ # 1:370

up<-i\*STEP

danex<-dane1[1:up,]

start.time<-Sys.time()

model1<-lm(eq, data=danex)

end.time<-Sys.time()

time.taken<-difftime(end.time, start.time, units="secs")

crds<-as.matrix(danex[,57:58])

pkt.k.sym.listw<-nb2listw(make.sym.nb(knn2nb(knearneigh(crds, k=5, longlat=NULL))))

moran<-moran.test(model1$residuals, pkt.k.sym.listw)

coef.ols[,i]<-summary(model1)$coefficients[,1]

error.ols[,i]<-summary(model1)$coefficients[,2]

other.ols[1,i]<-summary(model1)$r.squared

other.ols[2,i]<-time.taken

other.ols[3,i]<-moran$p.value

}

# saving objects with results

write.table(coef.ols, file="coefOLS.csv", sep = ",", col.names=TRUE)

write.table(error.ols, file="errorOLS.csv", sep = ",", col.names=TRUE)

write.table(other.ols, file="otherOLS.csv", sep = ",", col.names=TRUE)

### # Fig.3a

**# plot of coefficients**

nn<-1:37

plot(nn\*1000, coef.ols[3,], type="l", xlab="size of dataset", ylab="beta coefficient", ylim=c(1.5, 1.8), col="red")

lines(nn\*1000, coef.sar[3,], lty=1, col="black")

lines(nn\*1000, coef.sem[3,], lty=2, col="grey20")

lines(nn\*1000, coef.sdm[3,], lty=3, col="grey20")

legend(30000, 1.65, legend=c("OLS", "SAR", "SEM", "SDM"), lty=c(1, 1,2,3), col=c("red", "black", "grey20", "grey20"), bty="n")

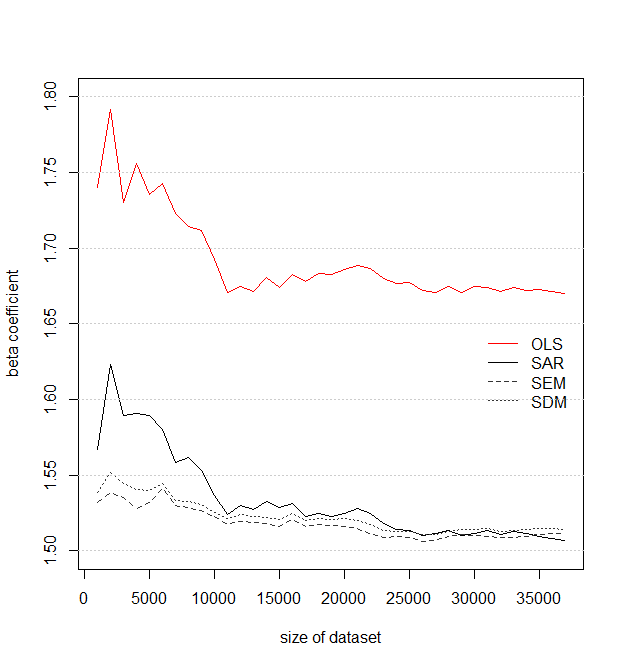
abline(h=(30:36)\*5/100, lty=3, col="grey80")

savePlot(filename="Fig03a", type="jpg")

savePlot(filename="Fig03a", type="tif")

savePlot(filename="Fig03a", type="png")

**savePlot(filename="Fig03a", type="eps") # best quality**



### # Fig.3b

**# plot of standard errors**

# transposition of dataset for easier plotting

errorOLS.t<-t(error.ols)

errorSEM.t<-t(error.sem)

errorSAR.t<-t(error.sar)

errorSDM.t<-t(error.sdm)

# values of SE of coefficients derived from sqrt(2) rule and using SE from first estimation (for n=1000)

pp<-c(1000, 2000, 4000, 8000, 16000, 32000)

ppp.SEM<-c(errorSEM.t[1,3], errorSEM.t[1,3]/(2^0.5)^1, errorSEM.t[1,3]/(2^0.5)^2, errorSEM.t[1,3]/(2^0.5)^3, errorSEM.t[1,3]/(2^0.5)^4, errorSEM.t[1,3]/(2^0.5)^5)

ppp.SAR<-c(errorSAR.t[1,3], errorSAR.t[1,3]/(2^0.5)^1, errorSAR.t[1,3]/(2^0.5)^2, errorSAR.t[1,3]/(2^0.5)^3, errorSAR.t[1,3]/(2^0.5)^4, errorSAR.t[1,3]/(2^0.5)^5)

ppp.SDM<-c(errorSDM.t[1,3], errorSDM.t[1,3]/(2^0.5)^1, errorSDM.t[1,3]/(2^0.5)^2, errorSDM.t[1,3]/(2^0.5)^3, errorSDM.t[1,3]/(2^0.5)^4, errorSDM.t[1,3]/(2^0.5)^5)

ppp.OLS<-c(errorOLS.t[1,3], errorOLS.t[1,3]/(2^0.5)^1, errorOLS.t[1,3]/(2^0.5)^2, errorOLS.t[1,3]/(2^0.5)^3, errorOLS.t[1,3]/(2^0.5)^4, errorOLS.t[1,3]/(2^0.5)^5)

nn<-1:37 # number of points on x axis

plot(nn\*1000, errorOLS.t[,3], type="l", xlab="size of dataset", ylab="SE of beta coefficient", ylim=c(0, 0.07), col="red", lty=1, lwd=2)

lines(pp, ppp.OLS, lty=1, col="red", lwd=1)

lines(nn\*1000, errorSAR.t[,3], lwd=2, lty=1)

lines(pp, ppp.SAR, lwd=1, lty=1)

lines(nn\*1000, errorSEM.t[,3], col="grey20", lwd=2, lty=2)

lines(pp, ppp.SEM, col="grey20", lty=2)

lines(nn\*1000, errorSDM.t[,3], lwd=2, col="grey20", lty=3)

lines(pp, ppp.SDM, lwd=1, lty=3, col="grey20")

legend("topright", c("OLS empirical SE", "OLS theoretical SE", "SAR empirical SE","SAR theoretical SE", "SEM empirical SE","SEM theoretical SE","SDM empirical SE","SDM theoretical SE"), col=c("red","red", "black","black","grey20","grey20", "grey20", "grey20"), lwd=c(2,1, 2,1, 2, 1, 2,1), lty=c(1,1,1,1,2,2,3,3), bty="n")

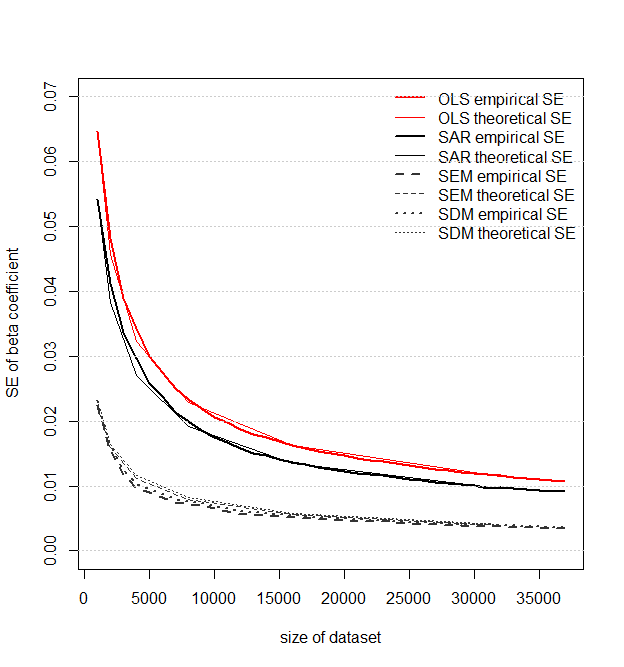
abline(h=(0:7)/100, lty=3, col="grey80")

savePlot(filename="Fig03b", type="jpg")

savePlot(filename="Fig03b", type="tif")

savePlot(filename="Fig03b", type="png")

**savePlot(filename="Fig03b", type="eps") # best quality**



## # Figure 4

**# This code visualises the computational time depending on a sample size by doubling datasets**

**# general settings**

obs<-c(1000, 2000, 4000, 8000, 16000, 32000, 64000, 128000, 256000, 512000) # doubled dataset size

eq<-roa~empl+prod+constr+serv+dist.lublin # structure of the model

**# computations for OLS**

czas.reg<-matrix(0, nrow=1, ncol=10) # number of obs. in columns, empty object for outcomes

for(i in 1:10){ # for each number of observations

selector<-sample(1:37374, obs[i], replace=TRUE) # sampling with replacement

crds<-dane[selector, 14:15] # getting coordinats of selected data

start.time<-Sys.time()

model<-lm(eq, data=dane[selector,]) # OLS estimation

end.time<-Sys.time()

czas.reg[1,i]<-difftime(end.time, start.time, units="secs")

}

colnames(czas.reg)<-obs

czas.reg # time of OLS computations – results of the above procedure

# 1000 2000 4000 8000 16000 32000 64000

#[1,] 0.00880909 0.01665092 0.008812189 0.01439404 0.02861094 0.05092502 0.088907

# 128000 256000 512000

#[1,] 0.1566329 0.309252 0.7403281

**# computations for spatial models**

czas.W<-matrix(0, nrow=1, ncol=10) # number of obs. in columns, empty object for outcomes

czas.reg.SAR<-matrix(0, nrow=1, ncol=10)

czas.reg.SEM<-matrix(0, nrow=1, ncol=10)

czas.reg.SDM<-matrix(0, nrow=1, ncol=10)

for(i in 1:10){

selector<-sample(1:37374, obs[i], replace=TRUE) # sampling with replacement

crds<-dane[selector, 14:15] # getting coordinats of selected data

start.time<-Sys.time() # getting W

knnW<-nb2listw(make.sym.nb(knn2nb(knearneigh(as.matrix(crds), k=5))))

end.time<-Sys.time()

czas.W[1,i]<-difftime(end.time, start.time, units="secs")

start.time<-Sys.time() # getting SAR

model<-lagsarlm(eq, data=dane[selector,], knnW, method="LU")

end.time<-Sys.time()

czas.reg.SAR[1,i]<-difftime(end.time, start.time, units="secs")

start.time<-Sys.time() # getting SEM

model<-errorsarlm(eq, data=dane[selector,], knnW, method="LU")

end.time<-Sys.time()

czas.reg.SEM[1,i]<-difftime(end.time, start.time, units="secs")

start.time<-Sys.time() # getting SDM

model<-lagsarlm(eq, data=dane[selector,], knnW, method="LU", type="mixed")

end.time<-Sys.time()

czas.reg.SDM[1,i]<-difftime(end.time, start.time, units="secs")

}

colnames(czas.W)<-obs

colnames(czas.reg.SAR)<-obs

colnames(czas.reg.SEM)<-obs

colnames(czas.reg.SDM)<-obs

czas.W

czas.reg.SAR

czas.reg.SEM

czas.reg.SDM

#> czas.W

# 1000 2000 4000 8000 16000 32000 64000 128000 256000

#[1,] 0.1948879 0.2820439 0.6457729 1.505588 4.136859 13.1797 45.75799 167.531 638.0256

# 512000

#[1,] 2507.609

#> czas.reg.SAR

# 1000 2000 4000 8000 16000 32000 64000 128000 256000

#[1,] 3.164437 0.6256161 0.7869589 1.08379 1.745772 3.698762 6.519682 13.81193 28.64093

# 512000

#[1,] 96.4192

#> czas.reg.SEM

# 1000 2000 4000 8000 16000 32000 64000 128000 256000

#[1,] 1.836553 1.082404 1.287758 2.085434 3.275577 7.343196 14.39664 29.37744 56.27843

# 512000

#[1,] 178.4494

#> czas.reg.SDM

# 1000 2000 4000 8000 16000 32000 64000 128000 256000

#[1,] 2.678187 0.8831871 1.203551 1.916341 3.354606 7.19531 14.81233 27.41773 58.62408

# 512000

#[1,] 191.5154

**# vectors of computation times – rewritten from output**

ols.reg.time<-c(0.00880909, 0.01665092, 0.008812189, 0.01439404, 0.02861094, 0.05092502, 0.088907, 0.1566329, 0.309252, 0.7403281)

W.time<-c(0.1948879, 0.2820439, 0.6457729, 1.505588, 4.136859, 13.1797, 45.75799, 167.531, 638.0256, 2507.609)

SAR.reg.time<-c(3.164437, 0.6256161, 0.7869589, 1.08379, 1.745772, 3.698762, 6.519682, 13.81193, 28.64093, 96.4192)

SEM.reg.time<-c(1.836553, 1.082404, 1.287758, 2.085434, 3.275577, 7.343196, 14.39664, 29.37744, 56.27843, 178.4494)

SDM.reg.time<-c(2.678187, 0.8831871, 1.203551, 1.916341, 3.354606, 7.19531, 14.81233, 27.41773, 58.62408, 191.5154)

**# plot for all models**

plot(obs, ols.reg.time, type="l", main="Computation time [sec] of single model

with 5 explanatory variables", xlab="size of dataset", ylab="time in seconds", ylim=c(0,2550), lwd=3, lty=2, axes=F)

axis(2)

axis(1, at=obs, labels=obs)

lines(obs, W.time, lwd=2, lty=3)

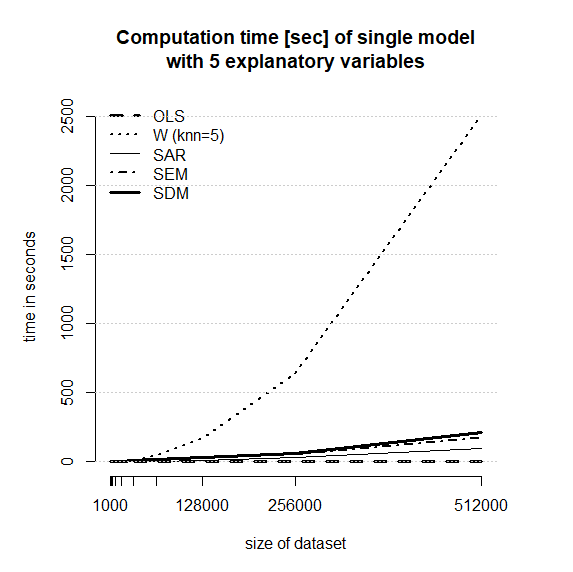
lines(obs, SAR.reg.time, lwd=1)

lines(obs, SEM.reg.time, lwd=2, lty=4)

lines(obs, SDM.reg.time\*1.1, lwd=3) # to avoid overlapping lines

legend("topleft", legend=c("OLS", "W (knn=5)", "SAR", "SEM", "SDM"), lwd=c(3,2,1, 2,3), lty=c(2,3,1,4,1), bty="n")

abline(h=c(0,5,10,15,20,25)\*100, lty=3, col="grey80")



## # Figure 5

**# This code illustrates analysed points divided with k-means into spatial partitions**

**# Colours may be differently assigned due to the randomness inherited in the procedure**

**# partitioning of points into clusters with 1000 points each**

groups<-kmeans(dane1.in[, 57:58], dim(dane1.in)[1]/1000) # k-means algorithm

dane1.in$kmean<-groups$cluster # vector of cluster IDs

cols<-rep(grey.colors(30), times=5) # vector of colours

cols<-sample(cols, max(dane1.in$kmean)) # mixing colours

brks<-1:max(dane1.in$kmean) # intervals for colours

woj.df<-as.data.frame(woj) # single region

woj.lub<-woj[woj.df$jpt\_nazwa\_=="lubelskie",]

plot(woj.lub) # contour map

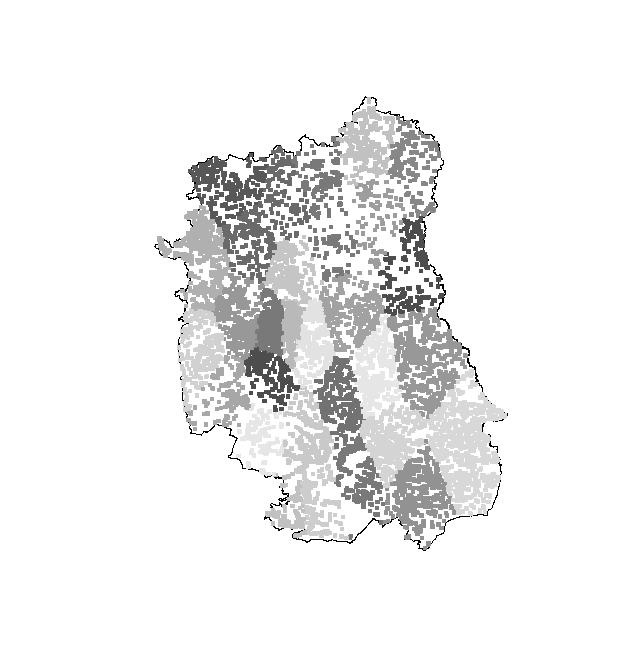
points(dane1.in[,57:58], col=cols[findInterval(dane1.in$kmean, brks)], pch=".", bg=cols[findInterval(dane1.in$kmean, brks)], cex=4.5) # points in colours

savePlot(filename="Fig05", type="jpg")

savePlot(filename="Fig05", type="tif")

savePlot(filename="Fig05", type="png")

**savePlot(filename="Fig05", type="eps") # best quality**



## # Figure 6

**# This code runs bootstrap OLS models and saves the results – coefficients and their standard errors. On this basis, one can derive analysis for increasing sample. To save memory, code writes all necessary files into Working Directory, delete objects immediately, and reads them back when needed.**

**# STAGE 1 – estimation on subsamples**

**# we build SELECTOR – matrix which indicates what observations are taken into the model in each iteration**

**# we estimate *n.col* of models and save results**

**# SELECTOR matrix (sampling with replacement)**

for(j in 1:37) # how many times we run internal loop, it depends on n.obs (n.row\*max here=n.obs)

{

n.col<-2000 # number of iterations / number of columns # choose as you wish

n.row<-j\*1000 # number of obs in a sample / number of rows, value 1000 is a STEP

selector<-matrix(0, nrow=n.row, ncol=n.col)

for(i in 1:n.col) # for all iterations

{

vec<-sample(1:dim(dane1)[1], n.row, replace=FALSE)

selector[,i]<-vec

}

**# OLS models**

coef<-matrix(0, nrow=7, ncol=n.col) #coefficients + R2

error<-matrix(0, nrow=6, ncol=n.col)

eq<-roa~empl+prod+constr+serv+dist

for(i in 1:n.col)

{

danex<-dane1[selector[,i],]

model1<-lm(eq, data=danex)

coef[1:6,i]<-summary(model1)$coefficients[,1]

coef[7,i]<-summary(model1)$r.squared

error[,i]<-summary(model1)$coefficients[,2]

}

# saving and removing objects

#assign(paste("coef", j, sep=""),coef) # we could save object in memory

#assign(paste("error", j, sep=""),error)

write.csv(t(coef), file=paste("coef",j,".csv", sep="")) # file in WD

write.csv(t(error), file=paste("error",j,".csv", sep=""))

remove(coef) # removing from memory

remove(error)

}

**# STAGE 2: checking different scenarios depending on sample size and number of iterations**

**# iterations are in fact to select n.col1 results from all n.col available results**

**# the ratio of SE is to divide SE in a given model by SE in full sample model**

**# this ratio is saved in individual tables**

ile<-(1:20)\*100 # 20\*1000=2000 = n.col (n.col=numer of iterations)

ile

ileile<-length(ile)

ileile

beta1.mean<- matrix(0, ncol=37, nrow=ileile) # output for beta

beta1.std<- matrix(0, ncol=37, nrow=ileile) # output for SE

rownames(beta1.mean)<-ile # how many replications in rows for beta

colnames(beta1.mean)<-(1:37)\*1000 # what is the sample size for beta

rownames(beta1.std)<-ile # how many replications in rows for SE

colnames(beta1.std)<-(1:37)\*1000 # what is the sample size for SE

n.col<-2000 # number of iterations / number of columns

for(j in 1:37){

for(i in 1:ileile){

vec.rand<-sample(1:n.col, ile[i], replace=FALSE)

beta.lim<-read.csv(paste("coef",j, ".csv", sep=""), header=TRUE, sep=",", dec=".")

beta.part<-beta.lim[vec.rand,]

av.beta<- apply(beta.part, MARGIN=2, mean) # mean in columns

std.beta<-apply(beta.part, MARGIN=2, sd) # std in columns

beta1.mean[i,j]<-av.beta[2]

beta1.std[i,j]<-std.beta[2]

}}

#######################################

### # Fig. 6a

**# This code is to analyse SE of coefficients from bootstrap**

**# reading files with coefficients for (1:37)\*1000 observations and 2000 iterations for each datasize**

av.SE<-matrix(0, nrow=37, ncol=3)

colnames(av.SE)<-c("nobs", "sd\_of\_coef", "mean\_of\_sd")

av.SE[,1]<-(1:37)\*1000

for(i in 1:37){

coef.temp<-read.csv(paste("coef", i, ".csv", sep=""), header=TRUE, sep=",", dec=".")

av.SE.temp<-apply(coef.temp, MARGIN=2, sd) # mean in columns

av.SE.temp

av.SE[i,2]<-av.SE.temp[4]

}

**# model on full data**

eq<-roa~empl+prod+constr+serv+dist

danex<-dane1

model1<-lm(eq, data=danex)

coef.real<-matrix(0,nrow=6, ncol=1)

error.real<- matrix(0,nrow=6, ncol=1)

coef.real<-summary(model1)$coefficients[,1]

error.real<-summary(model1)$coefficients[,2]

first.error<- av.SE[1,2]

pp<-c(1000, 2000, 4000, 8000, 16000, 32000, 40000)

SE.theor<-c(first.error, first.error/(2^0.5)^1, first.error/(2^0.5)^2, first.error/(2^0.5)^3, first.error/(2^0.5)^4, first.error/(2^0.5)^5, error.real[3])

**# plot of SE – theoretical, bootstrao and full sample**

par(mar=c(5,5,5,4))

plot(av.SE[,1], av.SE[,2], type="l", lty=2, xlab="no of observations in bootstrap subsample", ylab="std.dev of coefficient", main="Theoretical, bootstrapped and full model

variance of coefficient")

lines(pp, SE.theor, lty=1)

legend("topright", c("theoretical std.dev","bootstrap std.dev

(2000 replications)", "full model std.dev" ), lty=c(1,2, 1), lwd=c(1,1, 2), bty="n")

abline(h=error.real[3], lwd=2)

abline(v=(1:7)\*5000, lty=2, col="grey80")

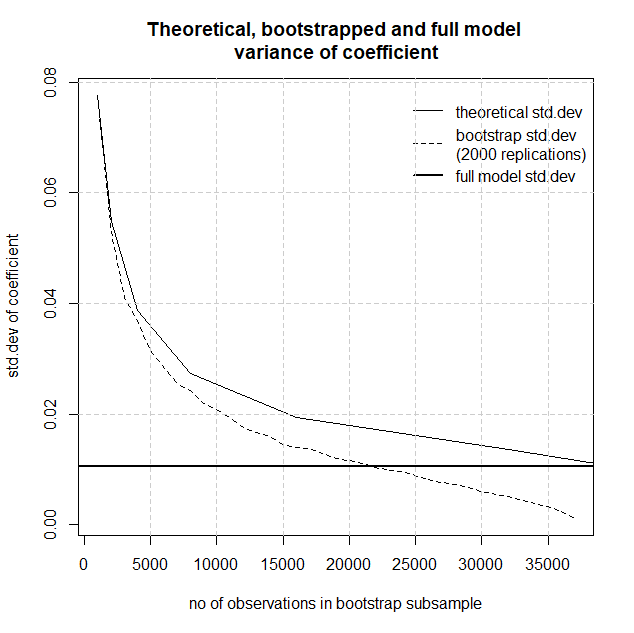
abline(h=(1:7)\*0.02, lty=2, col="grey80")

savePlot(filename="Fig06a", type="jpg")

savePlot(filename="Fig06a", type="tif")

savePlot(filename="Fig06a", type="png")

**savePlot(filename="Fig06a", type="eps") # best quality**



### # Fig. 6b

**# This code presents the SE of bootstrap depending on a number of replications (iterations) and sample size**

plot(ile, beta1.std[1:20,1], type="l", ylim=c(0,0.04), xlab="numer of replications", ylab="variance of estimator", main="Variance of coefficient dependent on

replications (i) and subsample size (s)")

lines(ile, beta1.std[1:20,5])

lines(ile, beta1.std[1:20,10])

lines(ile, beta1.std[1:20,15])

lines(ile, beta1.std[1:20,20])

lines(ile, beta1.std[1:20,25])

lines(ile, beta1.std[1:20,30])

lines(ile, beta1.std[1:20,35])

text(1900, beta1.std[20,1]+0.0008, "s=1000", cex=0.7)

text(1900, beta1.std[20,5]+0.0008, "s=5000", cex=0.7)

text(1900, beta1.std[20,10]+0.0008, "s=10000", cex=0.7)

text(1900, beta1.std[20,15]+0.0008, "s=15000", cex=0.7)

text(1900, beta1.std[20,20]+0.0008, "s=20000", cex=0.7)

text(1900, beta1.std[20,25]+0.0008, "s=25000", cex=0.7)

text(1900, beta1.std[20,30]+0.0008, "s=30000", cex=0.7)

text(1900, beta1.std[20,35]+0.0008, "s=35000", cex=0.7)

abline(v=(1:20)\*250, lty=2, col="grey80")

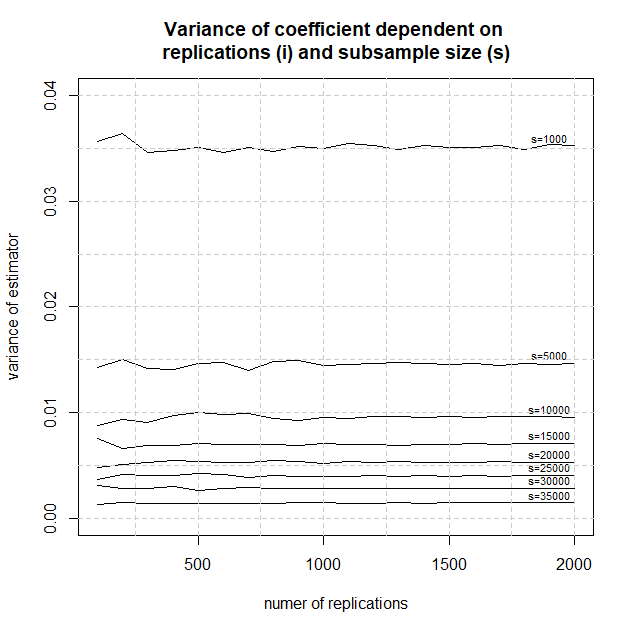
abline(h=(0:8)\*0.005, lty=2, col="grey80")

savePlot(filename="Fig06b", type="jpg")

savePlot(filename="Fig06b", type="tif")

savePlot(filename="Fig06b", type="png")

**savePlot(filename="Fig06b", type="eps") # best quality**



### # Fig.6c

**# This code presents the distributions of beta (boxplots) for each analysed sample size (1:37)\*1000**

**# reading outputs in loop**

beta1<-matrix(0, nrow=2000, ncol=37)

for(i in 1:37){

coef<-read.csv(paste("coef",i, ".csv", sep=""), header=TRUE, sep=",", dec=".")

beta1[,i]<-coef[,2]}

head(beta1)

**# boxplots**

boxplot.matrix(beta1, use.cols=TRUE, xlab="no of observations in bootstrap subsample \* 1000", ylab="value of beta coefficient", main="Boxplots of beta bootstrapped coefficient", col="white")

abline(h=(0:100)\*0.05, lty=2, col="grey80")

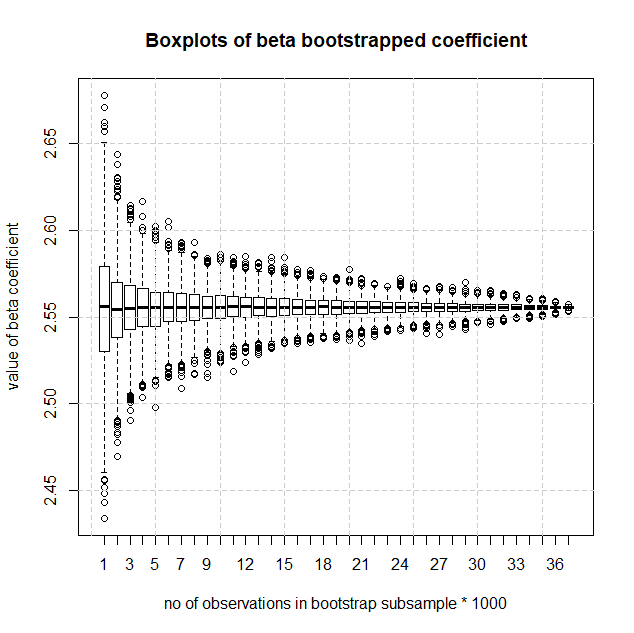
abline(v=(0:37)\*5, lty=2, col="grey80")

savePlot(filename="Fig06c", type="jpg")

savePlot(filename="Fig06c", type="tif")

savePlot(filename="Fig06c", type="png")

**savePlot(filename="Fig06c", type="eps") # best quality**



### # Fig. 6d

**# This code presents density distributions of coefficients for selected sample sizes**

**# reading files with coefficients for 37 sample sizes and renaming the object by assigning order number**

**# one gets coef1, coef2, ….coef15, …, coef37**

for(i in 1:37){

coef<-read.csv(paste("coef",i, ".csv", sep=""), header=TRUE, sep=",", dec=".")

assign(paste("coef", i, sep=""),coef)}

head(coef2) # example of object content

# X V1 V2 V3 V4 V5 V6 V7

#1 1 2.541480 -0.0003597458 1.847086 3.151685 6.244791 -0.008440068 0.9829284

#2 2 2.545536 -0.0003055212 1.687096 3.179054 6.244479 -0.008458611 0.9832774

#3 3 2.544969 0.0009511307 1.692601 3.152479 6.268804 -0.008592724 0.9826750

#4 4 2.562012 -0.0009981150 1.661343 3.153759 6.237558 -0.008379737 0.9816303

#5 5 2.531045 -0.0002949513 1.686129 3.168183 6.222969 -0.008019695 0.9832937

#6 6 2.576908 -0.0013038217 1.670303 3.164496 6.219201 -0.008790454 0.9822792

**# selected distributions are plotted as densities, information on sample size was added**

**# plotting densities**

plot(density(coef1[,4]), ylim=c(0,75), xlim=c(1.45, 1.85), main="Distributions of beta depending on sample size")

lines(density(coef5[,4]))

lines(density(coef10[,4]))

lines(density(coef15[,4]))

lines(density(coef20[,4]))

lines(density(coef25[,4]))

lines(density(coef30[,4]))

**# loop which writes as text the sample size on the level of maximum for given density**

for(i in c(1,5,10,15,20,25,30))

{

temp<-get(paste("coef", i, sep=""))

a<-density(temp[,4])

text(1.75, summary(a$y)[6], labels=paste("s=", i, ".000", sep=""))

}

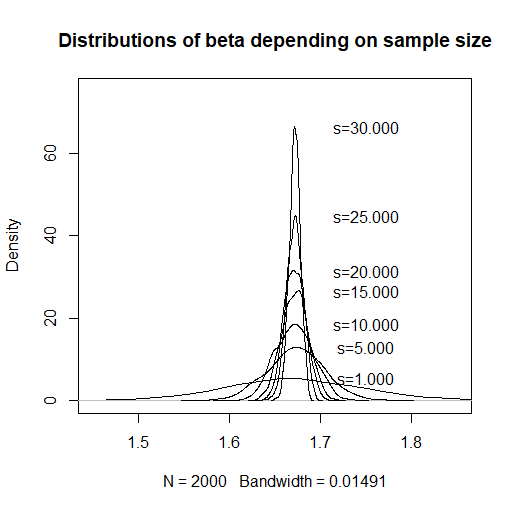
**# saving the plot**

savePlot(filename="Fig06d", type="jpg")

savePlot(filename="Fig06d", type="tif")

savePlot(filename="Fig06d", type="png")

**savePlot(filename="Fig06d", type="eps") # best quality**



## # Figure 7

**# This code generates simulations for different sample sizes: 1000, 2000, 4000 and 8000 observations. It runs SDM bootstrap models. It also saves results as files and reads them back into memory. To replicate figures without running the whole simulation, please read them to R. One can download the files from:** [**https://app.sugarsync.com/iris/wf/D1836703\_09668593\_7523736**](https://app.sugarsync.com/iris/wf/D1836703_09668593_7523736)**. At the end of this section, there are codes to read files with the results of this simulation.**

**# to run the whole simulation – use codes below**

**# to read files with results – go to the end of this section**

**# code uses data prepared in introduction**

summary(dane1.in) # these are data necessary for that code

dim(dane1.in)

#####################################

**# for 1000 observations**

**# selector matrix**

n.col<-500 # number of iterations #500

n.row<-1000 # number of obs in a sample

selector<-matrix(0, nrow=n.row, ncol=n.col)

for(i in 1:n.col){

vec<-sample(1:dim(dane1.in)[1], n.row, replace=TRUE)

selector[,i]<-vec}

**# final objects for SDM model**

coef.sdm<-matrix(0, nrow=n.col, ncol=11)

error.sdm<-matrix(0, nrow=n.col, ncol=11)

other.sdm<-matrix(0, nrow=n.row, ncol=2) # AIC.sdm, time.sdm, rho.sdm

roa<-matrix(0, nrow=n.row, ncol=n.col)

eq<-roa~empl+prod+constr+serv+dist # equation

for(i in 1:n.col){

danex<-dane1.in[selector[,i],]

roa[,i]<-danex$roa

crds<-as.matrix(dane1.in[selector[,i],57:58])

pkt.k.sym.listw<-nb2listw(make.sym.nb(knn2nb(knearneigh(crds, k=5)))) # knn=5

model.sdm<-lagsarlm(eq, data=danex, pkt.k.sym.listw, method="LU", type="mixed") **# SDM model**

**# saving results**

coef.sdm[i,]<-model.sdm$coefficients

error.sdm[i,]<-model.sdm$rest.se

other.sdm[i,1]<-AIC(model.sdm) # AIC.sdm

other.sdm[i,2]<-model.sdm$rho

}

**# changing the name of the object – by adding sample size into its name**

coef.sdm.1000<-coef.sdm

error.sdm.1000<-error.sdm

other.sdm.1000<-other.sdm

roa.1000<-roa

selector.1000<-selector

**# saving results as files**

**# removing objects for memory management – later they can up read into R from file**

write.csv(coef.sdm.1000, file="coefsdm1000.csv")

write.csv(error.sdm.1000, file="errorsdm1000.csv")

write.csv(other.sdm.1000, file="othersdm1000.csv")

write.csv(roa.1000, file="roa1000.csv")

write.csv(selector.1000, file="selector1000.csv")

remove(coef.sdm.1000)

remove(error.sdm.1000)

remove(other.sdm.1000)

remove(roa.1000)

remove(seector.1000)

#####################################

**# for 2000 observations**

**# selector matrix**

n.col<-500 # number of iterations #500

n.row<-2000 # number of obs in a sample

selector<-matrix(0, nrow=n.row, ncol=n.col)

for(i in 1:n.col){

vec<-sample(1:dim(dane1.in)[1], n.row, replace=TRUE)

selector[,i]<-vec}

**# final objects for SDM model**

coef.sdm<-matrix(0, nrow=n.col, ncol=11)

error.sdm<-matrix(0, nrow=n.col, ncol=11)

other.sdm<-matrix(0, nrow=n.row, ncol=2) # AIC.sdm, time.sdm, rho.sdm

roa<-matrix(0, nrow=n.row, ncol=n.col)

eq<-roa~empl+prod+constr+serv+dist # equation

for(i in 1:n.col){

danex<-dane1.in[selector[,i],]

roa[,i]<-danex$roa

crds<-as.matrix(dane1.in[selector[,i],57:58])

pkt.k.sym.listw<-nb2listw(make.sym.nb(knn2nb(knearneigh(crds, k=5)))) # knn=5

model.sdm<-lagsarlm(eq, data=danex, pkt.k.sym.listw, method="LU", type="mixed") **# SDM model**

**# saving results**

coef.sdm[i,]<-model.sdm$coefficients

error.sdm[i,]<-model.sdm$rest.se

other.sdm[i,1]<-AIC(model.sdm) # AIC.sdm

other.sdm[i,2]<-model.sdm$rho

}

**# changing the name of object – by adding sample size into its name**

coef.sdm.2000<-coef.sdm

error.sdm.2000<-error.sdm

other.sdm.2000<-other.sdm

roa.2000<-roa

selector.2000<-selector

**# saving results as files**

write.csv(coef.sdm.2000, file="coefsdm2000.csv")

write.csv(error.sdm.2000, file="errorsdm2000.csv")

write.csv(other.sdm.2000, file="othersdm2000.csv")

write.csv(roa.2000, file="roa2000.csv")

write.csv(selector.2000, file="selector2000.csv")

remove(coef.sdm.2000)

remove(error.sdm.2000)

remove(other.sdm.2000)

remove(roa.2000)

remove(seector.2000)

#####################################

**# for 4000 observations**

**# selector matrix**

n.col<-500 # number of iterations #500

n.row<-4000 # number of obs in a sample

selector<-matrix(0, nrow=n.row, ncol=n.col)

for(i in 1:n.col){

vec<-sample(1:dim(dane1.in)[1], n.row, replace=TRUE)

selector[,i]<-vec}

**# final objects for SDM model**

coef.sdm<-matrix(0, nrow=n.col, ncol=11)

error.sdm<-matrix(0, nrow=n.col, ncol=11)

other.sdm<-matrix(0, nrow=n.row, ncol=2) # AIC.sdm, time.sdm, rho.sdm

roa<-matrix(0, nrow=n.row, ncol=n.col)

eq<-roa~empl+prod+constr+serv+dist # equation

for(i in 1:n.col){

danex<-dane1.in[selector[,i],]

roa[,i]<-danex$roa

crds<-as.matrix(dane1.in[selector[,i],57:58])

pkt.k.sym.listw<-nb2listw(make.sym.nb(knn2nb(knearneigh(crds, k=5)))) # knn=5

model.sdm<-lagsarlm(eq, data=danex, pkt.k.sym.listw, method="LU", type="mixed") **# SDM model**

**# saving results**

coef.sdm[i,]<-model.sdm$coefficients

error.sdm[i,]<-model.sdm$rest.se

other.sdm[i,1]<-AIC(model.sdm) # AIC.sdm

other.sdm[i,2]<-model.sdm$rho

}

**# changing the name of object – by adding sample size into its name**

coef.sdm.4000<-coef.sdm

error.sdm.4000<-error.sdm

other.sdm.4000<-other.sdm

roa.4000<-roa

selector.4000<-selector

**# saving results as files**

write.csv(coef.sdm.4000, file="coefsdm4000.csv")

write.csv(error.sdm.4000, file="errorsdm4000.csv")

write.csv(other.sdm.4000, file="othersdm4000.csv")

write.csv(roa.4000, file="roa4000.csv")

write.csv(selector.4000, file="selector4000.csv")

remove(coef.sdm.4000)

remove(error.sdm.4000)

remove(other.sdm.4000)

remove(roa.4000)

remove(seector.4000)

#####################################

**# for 8000 observations**

**# selector matrix**

n.col<-500 # number of iterations #500

n.row<-8000 # number of obs in a sample

selector<-matrix(0, nrow=n.row, ncol=n.col)

for(i in 1:n.col){

vec<-sample(1:dim(dane1.in)[1], n.row, replace=TRUE)

selector[,i]<-vec}

**# final objects for SDM model**

coef.sdm<-matrix(0, nrow=n.col, ncol=11)

error.sdm<-matrix(0, nrow=n.col, ncol=11)

other.sdm<-matrix(0, nrow=n.row, ncol=2) # AIC.sdm, time.sdm, rho.sdm

roa<-matrix(0, nrow=n.row, ncol=n.col)

eq<-roa~empl+prod+constr+serv+dist # equation

for(i in 1:n.col){

danex<-dane1.in[selector[,i],]

roa[,i]<-danex$roa

crds<-as.matrix(dane1.in[selector[,i],57:58])

pkt.k.sym.listw<-nb2listw(make.sym.nb(knn2nb(knearneigh(crds, k=5)))) # knn=5

model.sdm<-lagsarlm(eq, data=danex, pkt.k.sym.listw, method="LU", type="mixed") **# SDM model**

**# saving results**

coef.sdm[i,]<-model.sdm$coefficients

error.sdm[i,]<-model.sdm$rest.se

other.sdm[i,1]<-AIC(model.sdm) # AIC.sdm

other.sdm[i,2]<-model.sdm$rho

}

**# changing the name of object – by adding sample size into its name**

coef.sdm.8000<-coef.sdm

error.sdm.8000<-error.sdm

other.sdm.8000<-other.sdm

roa.8000<-roa

selector.8000<-selector

**# saving results as files**

write.csv(coef.sdm.8000, file="coefsdm8000.csv")

write.csv(error.sdm.8000, file="errorsdm8000.csv")

write.csv(other.sdm.8000, file="othersdm8000.csv")

write.csv(roa.8000, file="roa8000.csv")

write.csv(selector.8000, file="selector8000.csv")

remove(coef.sdm.8000)

remove(error.sdm.8000)

remove(other.sdm.8000)

remove(roa.8000)

remove(selector.8000)

#############################################

**# reading the results of the simulation**

coef.sdm.1000<-read.csv("coefsdm1000.csv", header=TRUE)

coef.sdm.2000<-read.csv("coefsdm2000.csv", header=TRUE)

coef.sdm.4000<-read.csv("coefsdm4000.csv", header=TRUE)

coef.sdm.8000<-read.csv("coefsdm8000.csv", header=TRUE)

error.sdm.1000<-read.csv("errorsdm1000.csv", header=TRUE)

error.sdm.2000<-read.csv("errorsdm2000.csv", header=TRUE)

error.sdm.4000<-read.csv("errorsdm4000.csv", header=TRUE)

error.sdm.8000<-read.csv("errorsdm8000.csv", header=TRUE)

other.sdm.1000<-read.csv("othersdm1000.csv", header=TRUE)

other.sdm.2000<-read.csv("othersdm2000.csv", header=TRUE)

other.sdm.4000<-read.csv("othersdm4000.csv", header=TRUE)

other.sdm.8000<-read.csv("othersdm8000.csv", header=TRUE)

roa.1000<-read.csv("roa1000.csv", header=TRUE)

roa.2000<-read.csv("roa2000.csv", header=TRUE)

roa.4000<-read.csv("roa4000.csv", header=TRUE)

roa.8000<-read.csv("roa8000.csv", header=TRUE)

selector.1000<-read.csv("selector1000.csv", header=TRUE)

selector.2000<-read.csv("selector2000.csv", header=TRUE)

selector.4000<-read.csv("selector4000.csv", header=TRUE)

selector.8000<-read.csv("selector8000.csv", header=TRUE)

### # Fig.7a

**# This code shows boxplots of beta coefficients for four different sample sizes (1000, 2000, 4000, 8000) of SDM model and all iterations (500)**

**# multidata boxplot**

boxplot.matrix(as.matrix(cbind(coef.sdm.1000[,4],coef.sdm.2000[,4], coef.sdm.4000[,4], coef.sdm.8000[,4])), use.cols=TRUE, at=1:4, col=c("grey30","blue", "green", "antiquewhite2"))

legend("bottomright", c("1000 obs.","2000 obs.","4000 obs.", "8000 obs."), col=c("grey30","blue", "green", "antiquewhite2"), lwd=c(1,1,1,1), bty="n")

title(main="Bootstrapped beta in SDM models

B(i=500 iter., s=1000, 2000, 4000, 8000 obs.)")

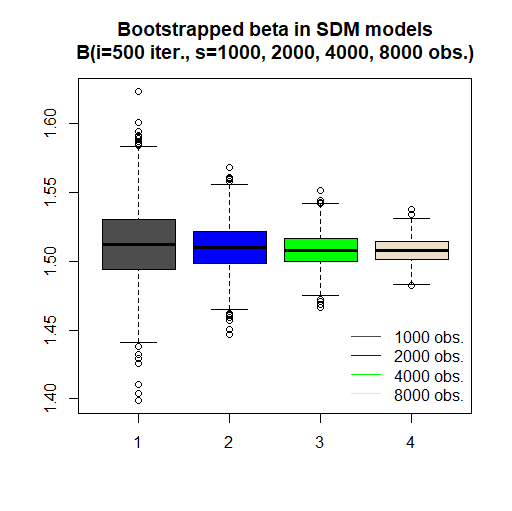
**# saving the plot**

savePlot(filename="Fig07a", type="jpg")

savePlot(filename="Fig07a", type="tif")

savePlot(filename="Fig07a", type="png")

**savePlot(filename="Fig07a", type="eps") # best quality**



### # Fig.7b

**# This code shows boxplots of SE of beta coefficients for four different sample sizes (1000, 2000, 4000, 8000) of SDM model and all iterations (500)**

**# multidata boxplot**

boxplot.matrix(as.matrix(cbind(error.sdm.1000[,4], error.sdm.2000[,4], error.sdm.4000[,4], error.sdm.8000[,4])), use.cols=TRUE, at=1:4, col=c("grey30","blue", "green", "antiquewhite2"))

legend("topright", c("1000 obs.","2000 obs.","4000 obs.", "8000 obs."), col=c("grey30","blue", "green", "antiquewhite2"), lwd=c(1,1,1,1), bty="n")

title(main="Bootstrapped std.error of beta in SDM models

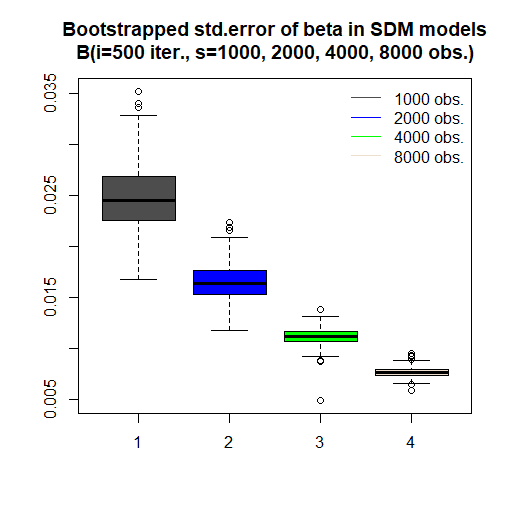
B(i=500 iter., s=1000, 2000, 4000, 8000 obs.)")

savePlot(filename="Fig07b", type="jpg")

savePlot(filename="Fig07b", type="tif")

savePlot(filename="Fig07b", type="png")

**savePlot(filename="Fig07b", type="eps") # best quality**



### # Fig.7c

**# This code shows boxplots of spatial rho for four different sample sizes (1000, 2000, 4000, 8000) of SDM model and all iterations (500)**

**# multidata boxplot**

boxplot.matrix(as.matrix(cbind(other.sdm.1000[1:500,3], other.sdm.2000[1:500,3], other.sdm.4000[1:500,3], other.sdm.8000[1:500,3])), use.cols=TRUE, at=1:4, col=c("grey30","blue", "green", "antiquewhite2"))

legend("bottomright", c("1000 obs.","2000 obs.","4000 obs.", "8000 obs."), col=c("grey30","blue", "green", "antiquewhite2"), lwd=c(1,1,1,1), bty="n")

title(main="Bootstrapped rho in SDM models

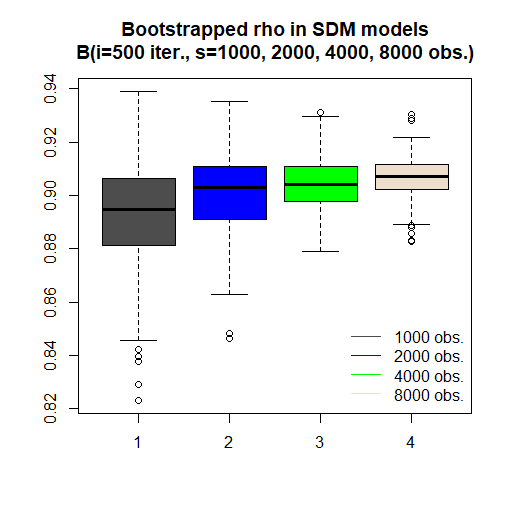
B(i=500 iter., s=1000, 2000, 4000, 8000 obs.)")

savePlot(filename="Fig07c", type="jpg")

savePlot(filename="Fig07c", type="tif")

savePlot(filename="Fig07c", type="png")

**savePlot(filename="Fig07c", type="eps") # best quality**



### # Fig.7d

**# This code shows boxplots of AIC for four different sample sizes (1000, 2000, 4000, 8000) of SDM model and all iterations (500)**

**# multidata boxplot**

boxplot.matrix(as.matrix(cbind(other.sdm.1000[1:500,2], other.sdm.2000[1:500,2], other.sdm.4000[1:500,2], other.sdm.8000[1:500,2])), use.cols=TRUE, at=1:4, col=c("grey30","blue", "green", "antiquewhite2"))

legend("bottomleft", c("1000 obs.","2000 obs.","4000 obs.", "8000 obs."), col=c("grey30","blue", "green", "antiquewhite2"), lwd=c(1,1,1,1), bty="n")

title(main="Bootstrapped AIC in SDM models

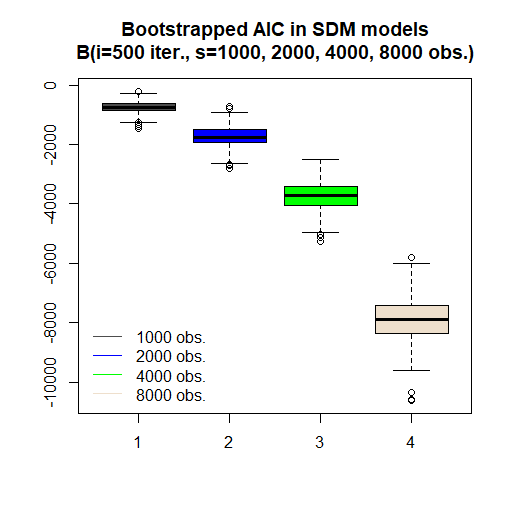
B(i=500 iter., s=1000, 2000, 4000, 8000 obs.)")

savePlot(filename="Fig07d", type="jpg")

savePlot(filename="Fig07d", type="tif")

savePlot(filename="Fig07d", type="png")

**savePlot(filename="Fig07d", type="eps") # best quality**



## # Figure 8

**# This code shows two-dimensional MDS (Multidimensional Scaling) projection of SDM coefficients and medoid model on that**

**# reading results of bootstrap – choose one**

coef.sdm.1000<-read.csv("coefsdm1000.csv", header=TRUE)

coef.sdm.2000<-read.csv("coefsdm2000.csv", header=TRUE)

coef.sdm.4000<-read.csv("coefsdm4000.csv", header=TRUE)

coef.sdm.8000<-read.csv("coefsdm8000.csv", header=TRUE)

head(coef.sdm.8000)

# X V1 V2 V3 V4 V5 V6

#1 1 0.1942580 -6.923068e-05 1.506415 2.993860 6.015925 -0.011612746

#2 2 0.2064064 7.764101e-05 1.492256 3.009590 6.011683 -0.013406758

#3 3 0.2080252 1.223174e-04 1.510886 2.999921 6.018077 -0.012234074

#4 4 0.2381364 6.003481e-05 1.496498 3.001634 6.017663 -0.012899821

#5 5 0.1942945 9.364496e-05 1.514259 3.015082 6.022398 -0.009673632

#6 6 0.2037080 -3.701819e-05 1.495144 3.006447 6.018611 -0.009910995

# V7 V8 V9 V10 V11

#1 4.207296e-04 -1.359627 -2.704692 -5.464944 0.010993620

#2 -3.244753e-05 -1.277973 -2.703809 -5.419320 0.012765327

#3 -4.727142e-05 -1.353823 -2.654678 -5.412746 0.011568278

#4 8.988989e-05 -1.310042 -2.641556 -5.324892 0.012133694

#5 1.974329e-04 -1.355497 -2.720101 -5.469127 0.009056587

#6 2.980075e-04 -1.332362 -2.714015 -5.424727 0.009258884

**# selection of best model**

c1<-pam(coef.sdm.4000, 1) #cluster::pam(), works for n<65536

summary(c1)

c1$clustering #

c1$medoids #

c1$id.med #

**# analysis of MDS**

dist.coef<-dist(coef.sdm.4000[,2:11]) # we need distance between units

#as.matrix(dist.coef)[1:10, 1:10] # let’s see the distance matrix

mds1<-cmdscale(dist.coef, k=2) #k - the maximum dimension of the space

summary(mds1) # we get coordinates of new points

# V1 V2

# Min. :-0.189978 Min. :-0.0570450

# 1st Qu.:-0.039085 1st Qu.:-0.0120684

# Median :-0.002026 Median :-0.0004834

# Mean : 0.000000 Mean : 0.0000000

# 3rd Qu.: 0.035032 3rd Qu.: 0.0120573

# Max. : 0.198052 Max. : 0.0524161

**# plot of points in 2D**

plot(mds1, pch=21, bg="black", xlab="Average Euclidean distance to other points – first dimension", ylab=" Average Euclidean distance to other points – second dimension", main="Most middle (medoid) SDM model

Multidimensional scaling 2D representation") # 11 columns reduced to 2

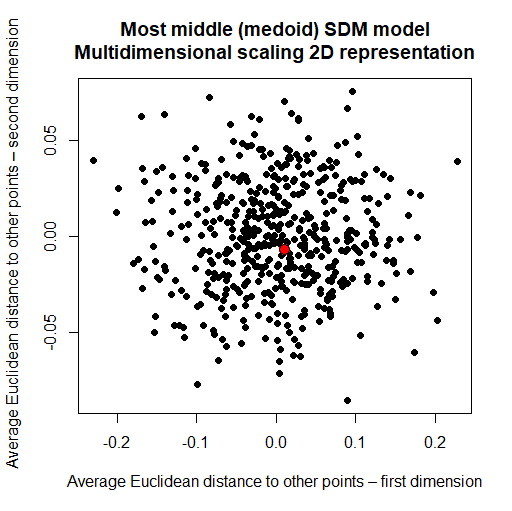
points(mds1[c1$id.med,1],mds1[c1$id.med,2], bg="red", pch=21, cex=1.5) # red point for best model

savePlot(filename="Fig08", type="jpg")

savePlot(filename="Fig08", type="tif")

savePlot(filename="Fig08", type="png")

**savePlot(filename="Fig08", type="eps") # best quality**



## # Figure 9

**# This code compares the beta coefficients and their SE – in full models and best bootstrap models for four types of models (OLS, SDM, SAR, SEM) – for 2000 observations**

**# below are the codes of simulation to get necessary results**

**# one can also read the files with ready outputs – files are available at:** [**https://app.sugarsync.com/iris/wf/D1836703\_09668593\_7523736**](https://app.sugarsync.com/iris/wf/D1836703_09668593_7523736)

**# codes reading files are available on the next page**

##################################

**# codes if you want to replicate analysis (a not read ready data)**

**# bootstrap for 2000 observations**

**# selector matrix**

n.col<-500 # number of iterations #500

n.row<-2000 # number of obs in a sample - #2000

selector<-matrix(0, nrow=n.row, ncol=n.col)

for(i in 1:n.col){

vec<-sample(1:dim(dane1.in)[1], n.row, replace=TRUE)

selector[,i]<-vec}

**# final objects for models**

coef.ols<-matrix(0, nrow=n.col, ncol=6)

error.ols<-matrix(0, nrow=n.col, ncol=6)

coef.sem<-matrix(0, nrow=n.col, ncol=7)

error.sem<-matrix(0, nrow=n.col, ncol=6)

coef.sar<-matrix(0, nrow=n.col, ncol=7)

error.sar<-matrix(0, nrow=n.col, ncol=6)

coef.sdm<-matrix(0, nrow=n.col, ncol=12)

error.sdm<-matrix(0, nrow=n.col, ncol=11)

roa<-matrix(0, ncol=n.col, nrow=n.row)

eq<-roa~empl+prod+constr+serv+dist # equation

for(i in 1:n.col){

danex<-dane1.in[selector[,i],]

roa[,i]<-danex$roa

crds<-as.matrix(dane1.in[selector[,i],57:58])

pkt.k.sym.listw<-nb2listw(make.sym.nb(knn2nb(knearneigh(crds, k=5, longlat = NULL)))) # knn=5

model.ols<-lm(eq, data=danex ) # model OLS

model.sem<-errorsarlm(eq, data=danex, pkt.k.sym.listw, method="LU") # model SEM

model.sar<-lagsarlm(eq, data=danex, pkt.k.sym.listw, method="LU" )# model SAR

model.sdm<-lagsarlm(eq, data=danex, pkt.k.sym.listw, method="LU", type="mixed")# model SDM

**# saving results as objects**

coef.ols[i,1:6]<-model.ols$coefficients

error.ols[i,]<-sqrt(diag(vcov(model.ols)))

coef.sem[i,1:6]<-model.sem$coefficients

coef.sem[i,7]<-model.sem$lambda

error.sem[i,1:6]<-model.sem$rest.se

coef.sar[i,1:6]<-model.sar$coefficients

coef.sar[i,7]<-model.sar$rho

error.sar[i,1:6]<-model.sar$rest.se

coef.sdm[i,1:11]<-model.sdm$coefficients

coef.sdm[i,12]<-model.sdm$rho

error.sdm[i,]<-model.sdm$rest.se

}

**# saving results as files**

write.csv(coef.ols, file="coefols2000.csv")

write.csv(error.ols, file="errorols2000.csv")

write.csv(coef.sar, file="coefsar2000.csv")

write.csv(error.sar, file="errorsar2000.csv")

write.csv(coef.sem, file="coefsem2000.csv")

write.csv(error.sem, file="errorsem2000.csv")

write.csv(coef.sdm, file="coefsdm2000.csv")

write.csv(error.sdm, file="errorsdm2000.csv")

##################################

**# reading bootstrap estimation results from files – if you do not run the above analysis**

coef.ols<-read.csv("coefols2000.csv", header=TRUE)

error.ols<-read.csv("errorols2000.csv", header=TRUE)

coef.sem<-read.csv("coefsem2000.csv", header=TRUE)

error.sem<-read.csv("errorsem2000.csv", header=TRUE)

coef.sar<-read.csv("coefsar2000.csv", header=TRUE)

error.sar<-read.csv("errorsar2000.csv", header=TRUE)

coef.sdm<-read.csv("coefsdm2000.csv", header=TRUE)

error.sdm<-read.csv("errorsdm2000.csv", header=TRUE)

**Best models**

**## PAM for OLS**

c1.ols<-pam(coef.ols,1) #cluster::pam(), works for n<65536

summary(c1.ols)

c1.ols$clustering # clustering vector – only 1 values

c1.ols$medoids # coefficients of selected best model

c1.ols$id.med # which iteration (model) is most representative

**## PAM for SEM**

c1.sem<-pam(coef.sem, 1) #cluster::pam(), works for n<65536

summary(c1.sem)

c1.sem$clustering # clustering vector – only 1 values

c1.sem$medoids # # coefficients of selected best model

c1.sem$id.med # which iteration (model) is most representative

**## PAM for SAR**

c1.sar<-pam(coef.sar, 1) #cluster::pam(), works for n<65536

summary(c1.sar)

c1.sar$clustering #

c1.sar$medoids #

c1.sar$id.med #

**## PAM for SDM**

c1.sdm<-pam(coef.sdm, 1) #cluster::pam(), works for n<65536

summary(c1.sdm)

c1.sdm$clustering #

c1.sdm$medoids #

c1.sdm$id.med #

#############################

**# reading files with full models estimated**

model.ols.full<-get(load("model\_ols\_full.RData"))

model.sem.full<-get(load("model\_sem\_full.RData"))

model.sar.full<-get(load("model\_sar\_full.RData"))

model.sdm.full<-get(load("model\_sdm\_full.RData"))

### # Fig.9a

**# comparison of results - coefficients**

plot(1:6, t(coef.sdm[c1.sdm$id.med, 2:7]), ylim=c(-1,7.5), lwd=2, xlab="consequtive variables in the model", ylab="beta coefficients values")

points(1:6, t(coef.sem[c1.sem$id.med, 2:7]),col="red", cex=0.8)

points(1:6, t(coef.sar[c1.sar$id.med,2:7]),col="green", cex=0.8)

points(1:6, t(coef.ols[c1.ols$id.med,1:6]),col="blue", cex=0.8)

points(model.sem.full$coefficients, col="red", cex=2.2)

points(model.sar.full$coefficients, col="green", cex=2.4)

points(model.sdm.full$coefficients[1:6], col="black", cex=2.6)

points(model.ols.full$coefficients[1:6], col="blue", cex=2.8) # OLS

legend(1.45, 7.5, c("medoid OLS","medoid SEM", "medoid SAR","medoid SDM", "full OLS","full SEM","full SAR", "full SDM"), col=c("blue","red","green","black","blue","red","green","black"), bty="n", pch=21, pt.cex=c(0.8, 0.8, 0.8, 0.8, 2.2, 2.2, 2.2, 2.2), cex=0.8)

abline(v=1:6, lty=3, col="grey80")

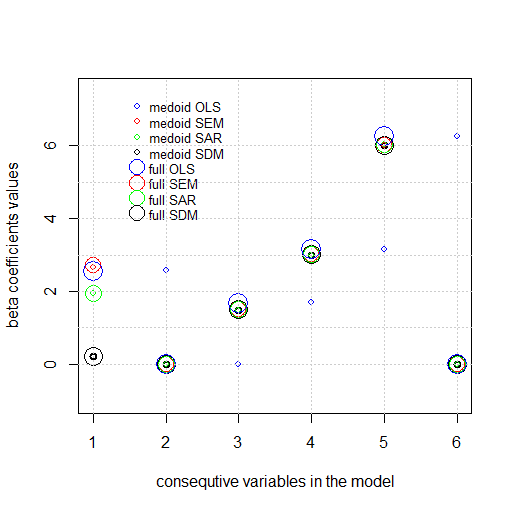
abline(h=0:6, lty=3, col="grey80")

savePlot(filename="Fig09a", type="jpg")

savePlot(filename="Fig09a", type="tif")

savePlot(filename="Fig09a", type="png")

**savePlot(filename="Fig09a", type="eps") # best quality**



### # Fig.9b

**# comparison of results - errors**

plot(1:6, t(error.sdm[c1.sdm$id.med, 2:7]), ylim=c(-0.01,0.05), lwd=2, xlab="consequtive variables in the model", ylab="standard error values")

points(t(error.sem[c1.sem$id.med,2:7]),col="red", cex=0.8)

points(t(error.sar[c1.sar$id.med,2:7]),col="green", cex=0.8)

points(t(error.ols[c1.ols$id.med,2:7]),col="blue", cex=0.8)

points(model.sem.full$rest.se, col="red", cex=2.2)

points(model.sar.full$rest.se, col="green", cex=2.4)

points(model.sdm.full$rest.se[1:6], col="black", cex=2.6)

points(model.ols.full$coefficients[,2], col="blue", cex=2.8) # OLS

legend(4.45, 0.05, c("medoid OLS","medoid SEM", "medoid SAR","medoid SDM", "full OLS","full SEM","full SAR", "full SDM"), col=c("blue","red","green","black","blue","red","green","black"), bty="n", pch=21, pt.cex=c(0.8, 0.8, 0.8, 0.8, 2.2, 2.2, 2.2, 2.2), cex=0.8)

abline(v=1:6, lty=3, col="grey80")

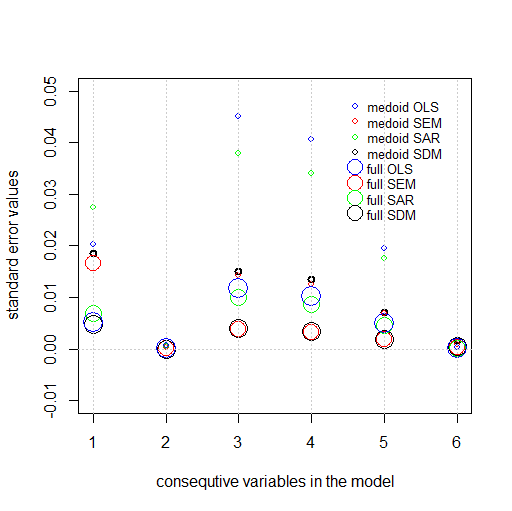
abline(h=0:6, lty=3, col="grey80")

savePlot(filename="Fig09b", type="jpg")

savePlot(filename="Fig09b", type="tif")

savePlot(filename="Fig09b", type="png")

**savePlot(filename="Fig09b", type="eps") # best quality**



## # Figure 10

**# This code shows how to get tessellation (Voronoi polygons) and put new points for predictions on that**

**# decision on sample size – 1000?, 2000?, 4000? 8000? – selection from previous calculations**

**# choose one of those below – paper presents tessellation for 1000 obs**

selector<-read.csv("selector1000.csv", header=TRUE, sep=",", dec=".")

#selector<-read.csv("selector2000.csv", header=TRUE, sep=",", dec=".")

#selector<-read.csv("selector4000.csv", header=TRUE, sep=",", dec=".")

#selector<-read.csv("selector8000.csv", header=TRUE, sep=",", dec=".")

**# changing the projections of points into planar**

**# code uses best PAM models, which were cerated in code for Fig.9**

points<-data.frame(x=dane1.in[selector[,c1.sdm$id.med],57], y=dane1.in[selector[,c1.sdm$id.med],58])

points.sp<-SpatialPoints(points) # new points in sp class - spherical

proj4string(points.sp)<-CRS("+proj=longlat +datum=NAD83") # spherical

points.sp<-spTransform(points.sp, CRS("+proj=merc +datum=NAD83")) # planar

**# changing the projections of contour into planar**

woj<-readOGR(".", "wojewodztwa") # 16 units

woj<-spTransform(woj, CRS("+proj=longlat +datum=NAD83")) # spherical

woj<-spTransform(woj, CRS("+proj=merc +datum=NAD83")) # planar

region<-woj[woj@data$jpt\_nazwa\_=="lubelskie",] # one region only

region.owin<-as.owin(region) # rgdal:: requires planar coordinates

**# tessellation in spatstat:: using ppp object**

region.ppp<-ppp(x=points.sp@coords[,1], y=points.sp@coords[,2], window=region.owin) # points of ppp class

region.tes<-dirichlet(region.ppp) # Dirichlet tesselation

**# converting back from owin/spatstat to sp class**

tes.poly<-as(region.tes, "SpatialPolygons")

proj4string(tes.poly)<-CRS("+proj=merc +datum=NAD83")

tes.poly<-spTransform(tes.poly, CRS("+proj=merc +datum=NAD83")) #planar

**# plot of tessellation (Voronoi polygons) and points – plot**

par(mar=c(1,1,1,1))

plot(region.tes, main=" ") # tessellation plot

plot(region.ppp, add=TRUE, pch=".", col="darkblue", cex=2)

par(mar=c(5,4,4,2))

**# saving of figure with 1000 points**

savePlot(filename="Fig10a", type="jpg")

savePlot(filename="Fig10a", type="tif")

savePlot(filename="Fig10a", type="png")

**savePlot(filename="Fig10a", type="eps") # best quality**

**# out-of-sample points used in predictions**

nnew<-100

points.pred<-SpatialPoints(dane1.out[1:nnew, 57:58]) # selection of a new point

proj4string(points.pred)<-CRS("+proj=longlat +datum=NAD83") # spherical projection

points.pred<-spTransform(points.pred, CRS("+proj=merc +datum=NAD83"))

plot(points.pred, add=TRUE, bg="red", pch=21) # adding out-of-sample points to tessellation figure

**# saving of figure with new out-of-sample locations**

savePlot(filename="Fig10b", type="jpg")

savePlot(filename="Fig10b", type="tif")

savePlot(filename="Fig10b", type="png")

**savePlot(filename="Fig10b", type="eps") # best quality**

