DSE project Laurens Hof

Name	SNR	ANR
Laurens Hof	2064467	496945

```
rm(list = ls())

library(tidyverse)
library(sf)
library(glmnet)
library(hdm)
library(broom)
```

Introduction

In the Netherlands, an important source of income for municipal governments is the Real Estate Tax, abbreviated in Dutch as OZB. It is levied as percentage of the so-called "WOZ" value, which is a public registration of the value of a property for tax purposes. The percentage levied differs per municipality.

The purpose of this tax is not redistribution (in fact, municipal governments are explicitly not allowed to engage in it). Rather, the goal is to finance local amenities provided by the municipalities. Since being close to desirable amenities would likely increase nearby property values, a real estate tax can be employed in an attempt to capture some of the public benefits.

Public libraries are one such amenity: municipalities provide the overwhelming majority of library subsidies. It would be interesting, then, to see whether one can empirically measure the effect of having a public library nearby on local WOZ-values using an adapted hedonic pricing method.

Methodology

At its barest, the research question "what is the effect of library proximity on WOZ values" can be represented by the equation:

$V_i = \alpha + \beta L_i + \varepsilon_i$

Where V_i is i's WOZ value, L_i some proxy for library proximity, and β the causal coefficient of interest. To operationalize this, I use the 2017 postcode statistics dataset (explanation in Dutch) published by Statistics Netherlands, which I retrieved through the PDOK spatial data repository. For (almost) all 4-digit postcode areas in the Netherlands, they register both the average WOZ value and the average road distance to a public library. The average WOZ-value only takes in residential areas, which likely is more of a convenience than a hindrance since that is the category where I expect the effect of library proximity to be most pronounced.

Obviously, there are myriad factors affecting WOZ values and library proximity is certainly not randomly assigned. The fact that the postcode statistics dataset is very rich allows me an approach to potentially still draw valid causal conclusions, since many possible confounders are contained in the dataset. It contains 135 variables measuring demographic makeup, some income statistics, data on homeownership rates and the proportion of single- versus multifamily homes, and construction years. It also includes average distances to many other amenities besides libraries, such as schools, highway exits, train stations, supermarkets, hospitals, and more.

I will employ these variables to construct a cross-fitted partialling-out estimator using LASSO. It supposes that in addition to the main explanatory variable, there is a large collection of other variables Z affecting the outcome, any of which may or may not be acting as a confounder:

$V_i = \alpha + \beta L_i + \gamma Z_i + \varepsilon_i$

The partialling-out estimator then works according to the following procedure: it tries to predict both V and L using Z, obtains the prediction errors for both, and then uses those errors to run a causal regression:

$V_i - \hat{V}_i(Z_i) = \alpha + \beta[L_i - \hat{L}_i(Z_i)] + \varepsilon_i$

I find LASSO to be a good choice because both my treatment and dependent variables and almost all my confounders are continuous numeric variables which can be normalized without a problem. The type of automatic model selection also helps, since a number of variables measure the same thing: there is both the average distance to a restaurant, the average number of restaurants within 1 kilometer, and within 3 and 5 kilometers. There is not always a strong indication which predictor would be most important, so it is nice to have the LASSO algorithm pick the strongest ones. Finally, it easily allows for a robustness check using double selection estimation.

My estimation is cross-fitted, which means that I "recycle" the splitted data. I randomly divide my data into a training and test set to prevent overfitting bias. The training set is used to only find the best penalty term, right variables to use and to get their coefficients. The test set is used to obtain the V and L residuals which I use for my final regression. I then run the partialling-out procedure twice: once with set 1 as test set and set 2 as training, and once with set 1 as training and set 2 as test. Then I take the average of the two estimates to get my final result, lowering the variance of the estimator.

One final problem with my data is spatial autocorrelation: almost everything is endogenous, but nearby things are more endogenous than faraway things. I solve this by including a new column for each variable that contains the mean of that variable for each of the post code areas that neighbor it. If there exists population variable that measures the number of people living in each postcode area, I will create an "average neighbor population" variable that measures the average of populations in each neighboring postcode. Then I will feed this information into the LASSO algorithm as a part of the nuisance data, and I will include the neighbor values of my treatment and outcome variables in the list of confounders. With this solution, I assume that any influence from fields further away can be captured using the direct neighbors (analogous to using an AR(1) model to capture temporal autocorrelation). It means that I suppose my Data Generating Process looks like this, with the blue arrow being the effect of interest and black arrows being confounding effects:

Result preview

I find a small, not significant, negative effect in both analyses. Increasing library distance by a standard deviation lowers average WOZ values by somewhere around slightly less than one percent of a standard deviation.

Import

```
cbs_pc4 <- read_sf("./postcode/cbs_pc4_2017.gpkg")
```

The pc4 dataset come in a GeoPackage format, which is the standard file format endorsed by the Open Geospatial Consortium. In R, the sf package makes it easy to work with, as many of its functions are modeled around, and often work well with, tidyverse equivalents. Practically, a .gpkg file loaded into R using sf turns into a tibble with an extra column called "geom", which encodes the spatial component. In this case, this means a vector multipolygon demarcating the area spanned by the postcode.

While there are no "standard" NA's in this dataset, there are a number of entries where information is replaced with the number -99997. Below I have generated a table with the number of observations where this is the case, per variable:

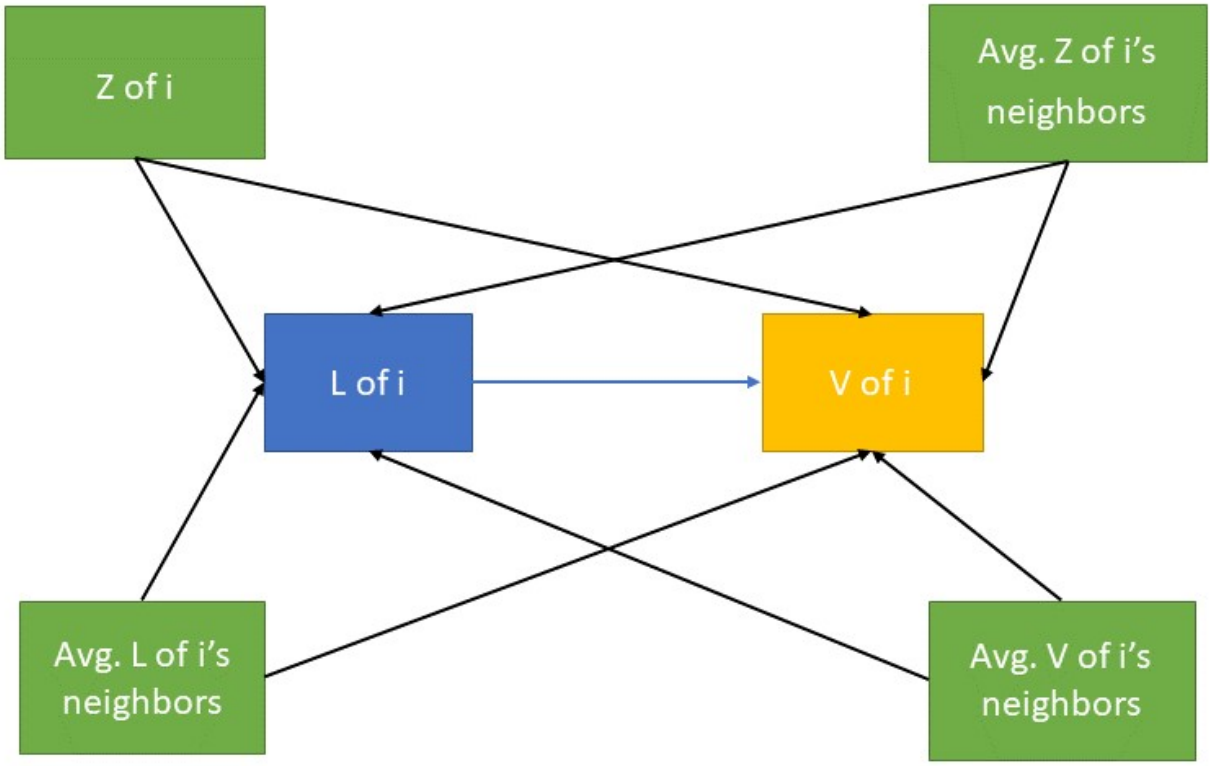

Figure 1: The assumed DGP

```
#make summary table counting the -99997 instances for each variable, then move the variables to the
#columns for readability.
cbs_pc4 %>%
  tibble %>%
  summarize(across(1:134, ~sum(. == -99997))) %>%
  pivot_longer(1:134, names_to = "variable", values_to = "number")
```

```
## # A tibble: 134 x 2
##    variable                          number
##    <chr>                              <int>
##  1 postcode                               0
##  2 aantal_inwoners                       32
##  3 aantal_mannen                         36
##  4 aantal_vrouwen                        40
##  5 aantal_inwoners_0_tot_15_jaar        111
##  6 aantal_inwoners_15_tot_25_jaar       120
##  7 aantal_inwoners_25_tot_45_jaar        82
##  8 aantal_inwoners_45_tot_65_jaar        47
##  9 aantal_inwoners_65_jaar_en_ouder      88
## 10 aantal_geboorten                     954
## # ... with 124 more rows
```

It is important to use caution when handling missing values, and to have a good idea of what generates them. Fortunately, CBS clearly tells us: it indicates either empty areas (in case of the square grid statistics) and areas where fewer than 5 instances occur (which are presumably redacted for privacy purposes). Therefore,

since -99997 indicates "too small to report" for practical purposes, I substitute them with 0. Below I show that there are no other negative, NA, or NaN values in the dataset.

```
cbs_pc4 %>%
  tibble() %>%
  summarize(across(1:134, ~sum((. < 0 & . != -99997)| is.na(.)))) %>%
  pivot_longer(1:134, names_to = "variable", values_to = "number") %>%
  summarize(`Total other missing` = sum(number))
```

```
## # A tibble: 1 x 1
##   `Total other missing`
##                   <int>
## 1                     0
```

Next, I will perform a collection of other reshaping steps. Firstly I substitute -99997 with zero as explained above. Secondly, for columns that break down aggregates into different categories, I remove the first column to alleviate collinearity concerns. Thirdly, I remove two categorical variables: the income category of the median household, and the "urbanicity" of the postcode. These categorical variables will be a hassle to work with, and both are simply recodings of information already available in the rest of the dataset. Fourthly, I remove those entries where my dependent variable (average residential WOZ-value) is equal to zero. Finally, I recalculate the breakdown categories as fractions of their respective totals. I also copy the totals into new variables so I can conveniently use the "unused" keyword to drop the original data without also losing the totals.

```
cbs_pc4 <- cbs_pc4 %>%
  #replace "too small" with "zero"
  mutate(across(2:134, ~ replace(.x, .x == -99997, 0))) %>%
  #remove a reference category, median income, and stedelijkheid
  select(-aantal_mannen, -aantal_inwoners_0_tot_15_jaar,
         -percentage_nederlandse_achtergrond, -aantal_woningen_bouwjaar_voor_1945,
         -aantal_eenpersoonshuishoudens, -percentage_huurwoningen,
         -mediaan_inkomen_huishouden, -stedelijkheid) %>%
  #drop 0 outcomes in dependent variable
  filter(gemiddelde_woz_waarde_woning > 0) %>%
  #recalculate categories to fractions
  mutate(pop = aantal_inwoners,
         women = aantal_vrouwen/aantal_inwoners,
         age_15_25 = aantal_inwoners_15_tot_25_jaar/aantal_inwoners,
         age_25_45 = aantal_inwoners_25_tot_45_jaar/aantal_inwoners,
         age_45_65 = aantal_inwoners_45_tot_65_jaar/aantal_inwoners,
         age_65_up = aantal_inwoners_65_jaar_en_ouder/aantal_inwoners,
         fr_welfare = aantal_personen_met_uitkering_onder_aowlft/aantal_inwoners,
         hh = aantal_part_huishoudens,
         multiperson = aantal_meerpersoonshuishoudens_zonder_kind/aantal_part_huishoudens,
         singleparent = aantal_eenouderhuishoudens/aantal_part_huishoudens,
         twoparent = aantal_tweeouderhuishoudens/aantal_part_huishoudens,
         houses = aantal_woningen,
         built_45_65 = aantal_woningen_bouwjaar_45_tot_65/aantal_woningen,
         built_65_75 = aantal_woningen_bouwjaar_65_tot_75/aantal_woningen,
         built_75_85 = aantal_woningen_bouwjaar_75_tot_85/aantal_woningen,
         built_85_95 = aantal_woningen_bouwjaar_85_tot_95/aantal_woningen,
         built_95_05 = aantal_woningen_bouwjaar_95_tot_05/aantal_woningen,
         built_05_15 = aantal_woningen_bouwjaar_05_tot_15/aantal_woningen,
```

```
         built_15_up = aantal_woningen_bouwjaar_15_en_later/aantal_woningen,
         fr_multifam = aantal_meergezins_woningen/aantal_woningen,
         .keep = "unused")
```

Adjacent postcodes

Next, I need to compute the averages of neighbors. The first step in this is to identify neighboring postcodes. Luckily, the combination of all postcode polygons covers all land space in the Netherlands, which means that I can use the st_touches function to find neighboring postcodes. In the following code fragment, I use a for loop over the rows of the pc4 dataset to create a column of lists that contain all postcodes bordering the current row. This operation could be made faster by converting the geom column from multipolygon to polygon, but that would perform undesirably if there exist any non-contiguous postcode areas, which I am not sure of.

```
#initialize an empty column
cbs_pc4$neighbors <- NA

for(i in 1:nrow(cbs_pc4)){
  #retrieve all the postcodes that touch the current row;
  neighbors_temp <- cbs_pc4$postcode[st_touches(cbs_pc4, cbs_pc4$geom[i], sparse = FALSE)]
  #If there are any, turn them into a list and put them in the right cell
  if(length(neighbors_temp) == 0){
    cbs_pc4$neighbors[i] <- 0
  } else{
    cbs_pc4$neighbors[i] <- list(neighbors_temp)
  }
}
```

Next, I create an empty copy of the pc4 data to serve as the "container" for the neighbor averages. Then, I fill in the new table by calculating the mean of the current variable for all postcodes that border the current entry, and repeating this over all rows and columns.

```
#make sure non-field columns (postcode and geom) are at the front for easier indexing
cbs_pc4 <- relocate(cbs_pc4, geom)

#create an empty duplicate table where I will store the neighbor data.
neighbordata <- tibble(cbs_pc4[NA,])
#add the postcode as an id
neighbordata$postcode <- cbs_pc4$postcode

#for each column c;
for(c in 3:127){
  #loop over the rows r of c;
  for(r in 1:nrow(cbs_pc4)){
    #calculate the mean of variable c across the neighbors of row r, and put it into cell [r,c]
    #in the neighbors dataframe
    neighbordata[r,c] <- mean(data.frame(cbs_pc4[cbs_pc4$postcode %in% cbs_pc4$neighbors[[r]], c])[,1])
  }
}
```

The final step is to combine the two dataframes by code. Now, I can also drop the geom and neighbors columns. Since I have calculated the neighbor averages, I will proceed to ignore the spatial characteristics of the data so that it can be used for my analysis.

```
#join and drop the geom and neighbors columns: now I am de facto working with a standard tibble
combined <- left_join(tibble(cbs_pc4), neighbordata, by = "postcode") %>%
  select(-geom.x, -geom.y,
         -neighbors.x, -neighbors.y)
```

Analysis

One final adjustment: replace the .x and .y suffices from the join with a more intuitive system: if a column is a neighbor average, it gets the _nb suffix. If it is original data, it gets none. Any missing values that resulted from dividing categories by a missing (and therefore zero) group total are dropped. This will select against those postcodes that are non-residential in nature, as this only happens if the number of inhabitants, dwellings, or households is zero. Places where a subcategory was too small to report will simply yield a fraction of "zero".

```
combined <- read_csv("./postcode/data_with_neighbors.csv")
```

```
##
## -- Column specification --------------------------------------------------------
## cols(
##   .default = col_double()
## )
## i Use `spec()` for the full column specifications.
```

```
#original data has the .x suffix, neighbor averages have the .y suffix.
#drop both and replace with a "_neighbor" indicator.
combined <- combined %>%
  rename_with(~ str_replace(., ".y", "_nb")) %>%
  rename_with(~str_replace(., ".x", "")) %>%
  #shame you can't divide by zero
  drop_na()
```

In order to be able to meaningfully use LASSO, I need to normalize all variables. Otherwise it will not include the variables with the strongest predictions, but the variables with the highest absolute variance.

```
#normalize all variables except postcode id
combined <- combined %>%
  mutate(across(2:253, ~ (. -mean(.))/sd(.)))
```

In my partialling-out procedure, I split the data into two halves. On the training set, I fit a rigorous LASSO. This does not select the penalty term for including new variables based on the minimum mean squared error, but instead calculates the theoretically optimal penalty based on Belloni et al. (2013). Because this optimal level depends on the standard deviation of the prediction errors (which are only observed ex post), this parameter is estimated iteratively. I have set "post" equal to FALSE, since conducting the final step of the estimation manually constitutes a stronger signal that I understand how it works. Furthermore, I also am not quite sure how to manipulate the test-train separation in the out-of-the-box method.

Next, I calculate the prediction errors for x and y (library proximity and average WOZ-value) by subtracting the fitted values based on the test set from the actual test set values of x and y. I store them in a dataframe and run my causal regression. I repeat these steps with the training and test set mirrored, and return the final estimate and standard error by taking the average of the two.

```r
#I used the course code as seed. Using 42 is overdone.
set.seed(310170)

#create splitting index
A <- sample(c(1:nrow(combined)), nrow(combined)/2)

#create predictor matrix that exludes libraries and WOZ
pred <- combined %>% select(-postcode,
                            -bibliotheek_gemiddelde_afstand_in_km,
                            -gemiddelde_woz_waarde_woning) %>%
  as.matrix()

#extract the treatment and outcome variables
y <- combined$gemiddelde_woz_waarde_woning
x <- combined$bibliotheek_gemiddelde_afstand_in_km

#version A: [A] is training set, [-A] is test set.

#find a rigorous lambda in [A]
lasso_train_y_A <- rlasso(x = pred[A,], y = y[A], post = FALSE)
lasso_train_x_A <- rlasso(x = pred[A,], y = x[A], post = FALSE)

#residuals of version A:
#the x and y values of minus the fitted values, using the lambda generated by the
#training set.
xresid_A <- x[-A] - predict(lasso_train_x_A, newdata = pred[-A,])
yresid_A <- y[-A] - predict(lasso_train_y_A, newdata = pred[-A,])

#put them in a dataframe, run the final lm for estimating the causal effect
resids_A <- bind_cols(as.data.frame(xresid_A), as.data.frame(yresid_A)) %>%
  rename(xresid = 1, yresid = 2)
```

```
## New names:
## * V1 -> V1...1
## * V1 -> V1...2
```

```r
treatreg_A <- lm(yresid ~ xresid, data = resids_A)

#version B: [-A] is training set, [A] is test set

#find lambda
lasso_train_y_B <- rlasso(x = pred[-A,], y = y[-A], post = FALSE)
lasso_train_x_B <- rlasso(x = pred[-A,], y = x[-A], post = FALSE)

#get residuals and frame them
xresid_B <- x[A] - predict(lasso_train_x_B, newdata = pred[A,])
yresid_B <- y[A] - predict(lasso_train_y_B, newdata = pred[A,])
resids_B <- bind_cols(as.data.frame(xresid_B), as.data.frame(yresid_B)) %>%
  rename(xresid = 1, yresid = 2)
```

```
## New names:
## * V1 -> V1...1
## * V1 -> V1...2
```

```
#run treatment regression
treatreg_B <- lm(yresid ~ xresid, data = resids_B)

#combine both halves of the crossfit, print point estimate and standard error
finaltable <- bind_rows(tidy(treatreg_A)[2,], tidy(treatreg_B)[2,])
finaltable %>% summarize(estimate = mean(estimate), std.error = mean(std.error))
```

```
## # A tibble: 1 x 2
##   estimate std.error
##      <dbl>     <dbl>
## 1  -0.0110    0.0204
```

The effect is negative as expected, and the magnitude is also roughly in line with my expectations. Increasing the distance to a library by one standard deviation decreases the average WOZ value in an area by around one percent of a standard deviation. However, the standard error is comparatively large, and I can not exlcude the possibility that the true effect is zero.

Next, I will use the results from the robust LASSO to conduct a double selection analysis as a robustness check. I take all the terms that were selected by the robust LASSO for predicting either library proximity or WOZ values, remove duplicates and the intercept, and run a regression of WOZ values on library proximity with all those selected predictors as controls.

```
#select all variable names from both training objects whose coefficients are not zero,
#combine them and then keep only the unique ones
vars_A <- unique(c(names(lasso_train_x_A$coefficients[lasso_train_x_A$coefficients != 0]),
           names(lasso_train_y_A$coefficients[lasso_train_y_A$coefficients != 0])))

#create a formula from the outcome, treatment, and the selected variables minus the intercept
formula_A <- str_c("gemiddelde_woz_waarde_woning ~ bibliotheek_gemiddelde_afstand_in_km +",
                   str_c(vars_A[-1], collapse = " + "))
#fit the model on the test set
doubleselect_A <- lm(formula_A, data = combined[-A,]) %>% tidy()

#do the same for the other group
vars_B <- unique(c(names(lasso_train_x_B$coefficients[lasso_train_x_B$coefficients != 0]),
           names(lasso_train_y_B$coefficients[lasso_train_y_B$coefficients != 0])))
formula_B <- str_c("gemiddelde_woz_waarde_woning ~ bibliotheek_gemiddelde_afstand_in_km +",
                   str_c(vars_B[-1], collapse = " + "))
doubleselect_B <- lm(formula_B, data = combined[A,]) %>% tidy()

#combine and average the two effects
d_sel_final <- bind_rows(doubleselect_A[2,], doubleselect_B[2,])
d_sel_final %>% summarize(estimate = mean(estimate), std.error = mean(std.error))
```

```
## # A tibble: 1 x 2
##   estimate std.error
##      <dbl>     <dbl>
## 1  -0.0127    0.0197
```

This says that the change in average WOZ value is slightly less than one percent of a standard deviation. At least, it would be if I could exclude zero.

Conclusion

My results confirm first intuitions about the effect of libraries on property values. However, due to their inaccuracy they can not be reliably used as inputs for a social cost-benefit analysis. Nevertheless, it highlights potentially interesting applications of data science methods in public policy analysis, even at comparatively low levels of government.

This research could be expanded by using the more fine-grained dataset available to the Land Registry Agency. While the CBS postcode statistics are very rich for a publicly available dataset, it cannot capture effects of very close proximity. For example, while having a school in the same neighborhood will probably raise property values at the margin, it has the possibility of lowering values immediately adjacent due to noise complaints.