Repeated sales/rent walkthrough

THIS FILE ONLY COMPILES DURING THE PIPELINE; ERRORS IF DONE MANUALLY!

Introduction

This markdown is a more lengthy explanation of the process behind the clustering of repeated sales. It explains the algorithm more generally and outlines the motivation behind some choices. The individual steps are taken directly from the code base, so they copy what the pipeline does.

Data And Running

Details of reading/reprocessing the RED data can be found in R/read_/, specifically read_RED.R and prepare_RED.R. These should be self explanatory. Note that the classification algorithm is designed to be run in parallel.[1] This is achieved by grouping the RED data on "blid", i.e. one group for each federal state. These groups are than classified in parallel for significant speedups. Note that this by default is possible for multiple object types (WK,HK,WM) and can be set via the "static_RED_types" variable. The branching over this argument is dynamic, so only the variable has to be set, the rest of the pipeline adjusts itself.

The classification digs down to coordinate-level, which is where I will start explaining the actual procedure.

Example Data

The example data is taken from a coordinate with 25 observations of federal state Bremen. Since the data is already subset to the required dissolution for the next steps, the variables "blid" and "latlon_utm" were dropped beforehand (See R/misc/make_example_markdown_data.R for details).

```
## load example data
tar_load(example_markdown_data)

# rename to match code convention
geo_grouped_data = example_markdown_data

#make this dynamic
#tar_load(example_markdown_data, store ="N:/FDZ/Intern/HiWi-Praktikanten/Mitarbeiter/Thorben/repeated o
```

General makeup of the data:

```
# show head
head(example_markdown_data)
```

[1]I recommend using 'tar_make_future(workers = n)', where n is the number of cores. $n = 4$ is a decent value, when no one else is using the server the number can be set higher. More than 8 is overkill, since Berlin alone typically bottlenecks. If speed becomes a big issue with growing data consider using "foreach()" on the coordinate level.

```
## Key: <counting_id>
##    wohnflaeche zimmeranzahl etage counting_id amonths emonths price_var
##          <num>        <num> <num>       <int>   <num>   <num>     <num>
## 1:          71            3     2      280955   24086   24088    135200
## 2:          71            3     2      280956   24089   24100    142900
## 3:          71            3     2      280957   24101   24106    139900
## 4:          71            3     2      280958   24107   24115    139900
## 5:          71            3     2      280959   24116   24127    136700
## 6:          56            2     1      280991   24101   24106    106800
```

```
# show summary
summary(example_markdown_data)
```

```
##   wohnflaeche     zimmeranzahl      etage       counting_id
##  Min.   : 56.0   Min.   :2.00   Min.   :0.00   Min.   : 280955
##  1st Qu.: 71.0   1st Qu.:3.00   1st Qu.:1.00   1st Qu.: 280992
##  Median : 80.0   Median :3.00   Median :2.00   Median : 837599
##  Mean   : 80.0   Mean   :2.96   Mean   :2.24   Mean   : 939823
##  3rd Qu.: 81.0   3rd Qu.:3.00   3rd Qu.:3.00   3rd Qu.:1269629
##  Max.   :149.9   Max.   :5.00   Max.   :6.00   Max.   :2828339
##     amonths         emonths        price_var
##  Min.   :24086   Min.   :24088   Min.   :102300
##  1st Qu.:24097   1st Qu.:24100   1st Qu.:138000
##  Median :24102   Median :24104   Median :139900
##  Mean   :24104   Mean   :24108   Mean   :155220
##  3rd Qu.:24107   3rd Qu.:24117   3rd Qu.:155000
##  Max.   :24134   Max.   :24135   Max.   :335700
```

Legacy notes

The initial design of the algorithm accommodated two types of similarity: resembling and exact. The former allows for slightly larger deviations, the latter is quite restrictive. “r_o” refers to resembling offset and “e_o” to exact offset. This has been deprecated for the time being as resembling offsets ended up being less than 1% of the data, making the added complexity not worth it.

parameter used for classification:

```
# these are globally defined in _targets.R
print(exportJSON)
```

```
##    RED_version RED_types   categories etag_e_o wohnflaeche_e_o zimmeranzahl_e_o
##         <char>    <char>       <char>    <num>           <num>            <num>
## 1:          v9        WK  wohnflaeche        0            0.05              0.5
## 2:          v9        WM        etage        0            0.05              0.5
## 3:          v9        HK zimmeranzahl        0            0.05              0.5
##    time_offset base_quarter
##          <num>       <char>
## 1:           6   2010-01-01
## 2:           6   2010-01-01
## 3:           6   2010-01-01
```

The problem

The main question this entire project seeks to answer is: which listings can be grouped together, since they likely to refer to the same object (as in apartment or house)? The idea is basically to match the most similar listings found on the coordinates based on their characteristics. To make this decision for any number of characteristic combinations, I use a modified version of k-nearest centroids clustering. @ref(fig:problem_plot) shows a scatter plot the example data in 3d, illustrating the three main dimensions: etage, wohnflaeche and zimmeranzahl. The plot summarizes the issue in answering the question quite well, as it can be quite difficult to visually identify which points might stem from the same object. Take for example the pair of points located around (2,100,4). It is quite challenging to make a decision whether or not these listings actually refer to the same object (with some measurement deviation in wohnflaeche). This is doubly so since the scatterplot does not show the presence of direct overlaps, which would be a helpful indication of a cluster. So lets move into a more formal level using the actual listings. We will proceed in two overarching dimensions: the characteristics dimension (classifying similarity) and the subsequent time dimension ('classifying' non list reason).

```r
# make color blind friendly palette
pal = MetBrewer::met.brewer("Egypt", n = length(example_markdown_data$etage))
with(
  example_markdown_data,
  scatterplot3d(x = wohnflaeche , y = zimmeranzahl, z =  etage , color = pal, pch = 16)
)
```

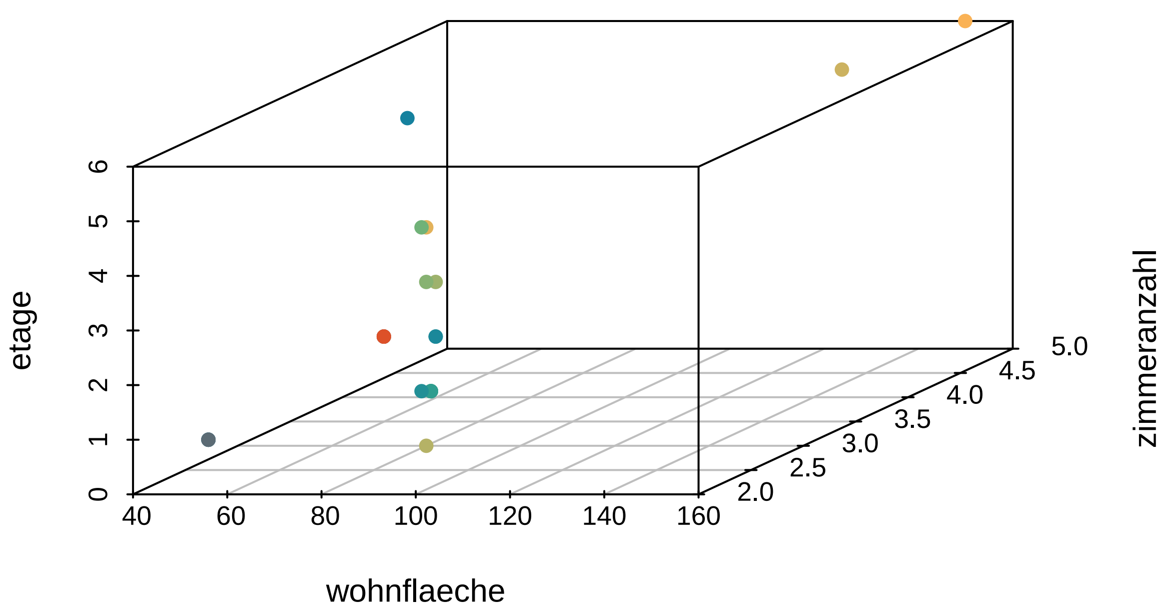

Figure 1: Illustration of the Problem

characteristics dimension

The goal of this dimension is to find out which listings are similar to one another in terms of how close their characteristics are to one another. As mentioned above, I use three main dimensions: etage, wohnflaeche and zimmeranzahl. These are all categorized by immoscout to be mandatory information and typically do not change much over time[2], which makes them ideal variables to base the classification off.

preperation and unique

To achieve this, the euclidean distance between the scaled values is used as mediator. But as a first step we can save some computations by reducing the data to only the unique combinations of the dimensions, as the distances is constant in those parameters. Another step done for the same reason is subsetting of columns and indexing. The latter allows data.table to perform merges using binary search, which is considerably faster than normal. I use a lot more merges than normal due to this, since its the most efficient way to perform the operations.

```
# make copy to modifiy keys
geo_grouped_data <- copy(geo_grouped_data)
setkey(geo_grouped_data, counting_id)
setindex(geo_grouped_data, wohnflaeche, etage, zimmeranzahl)

# extract ids and combinations of non-duplicates
occurence_ids <- geo_grouped_data[, counting_id]

# extract all combinations of categories
var_to_keep = c(categories, "counting_id")
combinations <- geo_grouped_data[, ..var_to_keep]

# filter out duplicates
# this has to be done ignoring the counting_id, as it is unique
dup_combinations = duplicated(combinations[, ..categories])
unique_combinations = combinations[!dup_combinations, ..categories]
unique_occurence_ids = combinations[!dup_combinations, counting_id]
```

similarity lists

The function "make_similarity_lists" returns to lists, the first containing the similarity indices and the second the corresponding scaled euclidean distances (henceforth just similarity distance) between each of the unique combinations.

```
similarity_lists = make_similarity_lists(unique_combinations,unique_occurence_ids)

similarity_index_list = similarity_lists[[1]]
similarity_dist_list = similarity_lists[[2]]

similarity_dist_list[is.na(similarity_index_list)] = NA
```

[2] Other than measurement errors and renovations, which is addressed by allowing small deviations. For the current deviations see @ref(sec:parameters)

similarity index

The former indicates whether or not a listing is similar (according to the definitions set in @ref(sec:parameters)) to any of the other listings. NA's indicate no similarity, 0 indicate similarity. Be default, all listings are similar to themselves, making the matrix diagonal zero.

similarity distances

For convenience, all distances are set to NA where the index is also NA.This just makes the matrix easier to look at.

```
similarity_dist_list = similarity_lists[[2]]
similarity_dist_list[1:5,1:5]
```

```
##       280955    280991    280996    729544    834487
##        <num>     <num>     <num>     <num>     <num>
## 1: 0.0000000 0.7561398 0.5752033 0.1341463 0.5101517
## 2: 0.5165813 0.0000000 0.8310038 0.5681176 0.4425756
## 3: 1.3351918 2.5744030 0.0000000 1.3353396 2.5002884
## 4: 0.1549296 0.8459085 0.5768564 0.0000000 0.5014400
## 5: 0.3518622 0.6470597 0.7153756 0.3353351 0.0000000
```

clustering

```
    similarity_dist_list[is.na(similarity_index_list)] = NA

    # setup and run the actual clustering
    clustering <- cluster$new(
      cluster_options = similarity_index_list,
      distance = similarity_dist_list
    )
    clustering$determine_cluster_centers()
```

```
## NULL
```

```
    clustering$centers |> head(n = 10)
```

```
##     counting_id  parent   sim_dist sim_index
##           <num>   <num>      <num>     <num>
##  1:      280955  280955 0.00000000         0
##  2:      280991  280991 0.00000000         0
##  3:      280996  280996 0.00000000         0
##  4:      729544  729544 0.00000000         0
##  5:      834487  834487 0.00000000         0
##  6:      984020  834487 0.02531646         0
##  7:      834487  984020 0.02469136         0
##  8:      984020  984020 0.00000000         0
##  9:     1269629 1269629 0.00000000         0
## 10:     2503689 1269629 0.01265823         0
```

```
clustering$centers <- clustering$centers[
        ,
        similarity_cost_function(.SD)
```

```
    ]
clustering$centers |>  head(n = 10)
```

```
##     counting_id  parent   sim_dist sim_index
##           <num>   <num>      <num>     <num>
##  1:      280955  280955 0.00000000         0
##  2:      280991  280991 0.00000000         0
##  3:      280996  280996 0.00000000         0
##  4:      729544  729544 0.00000000         0
##  5:      834487  984020 0.02469136         0
##  6:      984020  984020 0.00000000         0
##  7:     1269629 2503689 0.01250000         0
##  8:     2503689 2503689 0.00000000         0
##  9:     1310132 1310133 0.02439024         0
## 10:     1310133 1310133 0.00000000         0
```

```
# example_markdown_data is equvialent to geo_grouped_data in the code
out <- example_markdown_data[
    clustering$centers,
    on = .(counting_id)
  ]
out |>  head(n = 10)
```

```
##     wohnflaeche zimmeranzahl etage counting_id amonths emonths price_var
##           <num>        <num> <num>       <int>   <num>   <num>     <num>
##  1:          71            3     2      280955   24086   24088    135200
##  2:          56            2     1      280991   24101   24106    106800
##  3:          76            3     6      280996   24086   24089    139900
##  4:          82            3     2      729544   24095   24097    164900
##  5:          79            3     1      834487   24097   24099    149000
##  6:          81            3     1      984020   24103   24103    138000
##  7:          79            3     4     1269629   24104   24106    155000
##  8:          80            3     4     2503689   24127   24129    164000
##  9:          80            3     3     1310132   24117   24119    144900
## 10:          82            3     3     1310133   24105   24116    166700
##      parent   sim_dist sim_index
##       <num>      <num>     <num>
##  1:  280955 0.00000000         0
##  2:  280991 0.00000000         0
##  3:  280996 0.00000000         0
##  4:  729544 0.00000000         0
##  5:  984020 0.02469136         0
##  6:  984020 0.00000000         0
##  7: 2503689 0.01250000         0
##  8: 2503689 0.00000000         0
##  9: 1310133 0.02439024         0
## 10: 1310133 0.00000000         0
```