UrbanSIM Review

Danqing Zhang

March 1, 2016

Contents

1 Household location choice model

1.1 Household table

1.2 First step:household relocation model and transition model

1.2.1 Household relocation model

- code location
  models.py (urbansim_defaults), settings.yaml

- model explanation
  simple rate-based model reallocating 5% of households annually

- output
  they lose their building id and are reassigned by the hlcm (?)

1.2.2 Household transition model

- code location
  models.py (urbansim_defaults), settings.yaml

- model explanation
  generates new households based on control totals (?)
  does does it take care of aging, household formation, etc as well?

- output

1.3 Household location choice model

1.3.1 Model Explanation

1.3.2 Input and output

```
@orca.step('hlcm_owner_estimate')
def hlcm_owner_estimate(households, residential_units, unit_aggregations):
    return utils.lcm_estimate("hlcm_owner.yaml", households, "unit_id",
        residential_units, unit_aggregations)
```

From the code above, we can see that, households orca.table and residential_units orca.table, as well as the unit_aggregations orca.injective are the inputs.

1.3.3 LCM_estimate function from urbansim.utils

1. Here we need to make this clearer by comparision

   - function
     Signature: utils.lcm_estimate(cfg, choosers, chosen_fname, buildings, join_tbls)
     where cfg refers to the yaml file;
     chooser refers to either households or developers orca.table;
     chosen_fanme refers to the name of the column (present in choosers) which contains the ids that identify the chosen alternatives;
     buildings refers to either the residential_units or commercial_units;

Table 1: My caption

var	orca.table
np.log1p(residential_price)	buildings
np.log1p(sqft_per_unit)	buildings
ave_lot_size_per_unit	nodes
ave_income	nodes
persons	households
ave_hhsize	nodes
white/black/hisp/asian	households
pct_white/pct_black/pct_hisp/pct_asian	nodes
jobs	nodes
autoPeakTotal	logsums
transitPeakTotal	logsums
autoOffPeakRetail	logsums

join_tbls refers to orca.injective (A list of land use dataframes to give neighborhood info around the buildings will be joined to the buildings using existing broadcasts)

- calling the function
cfg:"hlcm_owner.yaml"
chooser orca.table:households
chosen_fname:"unit_id"
buildings:residential_units
join_tbls:unit_aggregations=buildings,nodes,logsums

2. Then how will the input be processed to get the output?

```
def lcm_estimate(cfg, choosers, chosen_fname, buildings, join_tbls):
    cfg = misc.config(cfg)
    choosers = to_frame(choosers, [], cfg, additional_columns=[
        chosen_fname])
    alternatives = to_frame(buildings, join_tbls, cfg)
    return yaml_to_class(cfg).fit_from_cfg(choosers,chosen_fname,
        alternatives,cfg)
```

1.3.4 How model is built?

The below table shows where each variables comes from. And the idea is that:now units are replacing units in updated version.But currently based on what the

1.3.5 How is filter built and how is segmented discrete choice model built?

hownrent is a column that indicate the household is the owner of the unit of the household is a renter of the unit.

```
choosers_fit_filters:
- hownrent == 1

choosers_predict_filters:
- hownrent == 1
```

1.3.6 Urbansim_defaults

Here many basic models are defined
And they call below puthon packages from urbansim:

```
from urbansim.models import RegressionModel, SegmentedRegressionModel, \
    MNLDiscreteChoiceModel, SegmentedMNLDiscreteChoiceModel, \
    GrowthRateTransition, transition
from urbansim.models.supplydemand import supply_and_demand
from urbansim.developer import sqftproforma, developer
from urbansim.utils import misc
```

2 Cost Function Procedure for Discrete Choice Model

2.1 Current Model

2.2 Setps

This task consists of three steps:

- construct a new dataframe accommodating needed variables from households table, residential_units table, and the the units_aggregations orca injective, consisting of three dataframes
- regression residential_price on other explanatory variables, and get residual as a pandas series, and add it as a column of the building orca.table(strange, why price is the at building level instead of units_level?)
- create a new yaml file and a new orca step for the new hlcm model with residual variables.

20cm

3 Data Source

3.1 H5 file

```
#Putting tables in the HDF5 file
store = pd.HDFStore(h5_path)
store['parcels'] = parcels # http://urbansim.org/Documentation/Parcel/
    ParcelTable
store['buildings'] = buildings # http://urbansim.org/Documentation/Parcel/
    BuildingsTable
store['households'] = hh # http://urbansim.org/Documentation/Parcel/
    HouseholdsTable
store['jobs'] = jobs # http://urbansim.org/Documentation/Parcel/JobsTable
store['zones'] = zones # http://urbansim.org/Documentation/Parcel/ZonesTable
store.close()
```

```
@orca.table('jobs', cache=True)
def jobs(store):
    # nets = store['nets']
    ###go from establishments to jobs
    # df = nets.loc[np.repeat(nets.index.values, nets.emp11.values)]\
        # .reset_index()
```

```
    # df.index.name = 'job_id'
    df = store['jobs']
    return df
```

4 Data Source

Where can I get the data? From "2015_06_01_bayarea_v3.h5", we can get "parcels","buildings","households","jobs" and "zones".Other tables are loaded in datasources.py file from local csv files from local data folder.

datasources.py is used to load local data into orca table.Then varibales.py is used to generate new columns of existent orca tables.

5 Setting, neighorhood_vars, price_vars

6 Hedonic Regression Model

6.1 Model Basics

Note simple linear regression cannot be done within urbansim modules.We need statmodels to conduct regression.

- Background in urbansim_utils

```
def hedonic_estimate(cfg, tbl, join_tbls):
    """
    Estimate the hedonic model for the specified table

    Parameters
    ----------
    cfg : string
        The name of the yaml config file from which to read the hedonic model
    tbl : DataFrameWrapper
        A dataframe for which to estimate the hedonic
    join_tbls : list of strings
        A list of land use dataframes to give neighborhood info around the
        buildings - will be joined to the buildings using existing broadcasts
    """
    cfg = misc.config(cfg)
    df = to_frame(tbl, join_tbls, cfg)
    return yaml_to_class(cfg).fit_from_cfg(df, cfg)
```

- rrh.yaml

```
name: rrh

model_type: regression

fit_filters:
- price_sqft > 0.5
- price_sqft < 7

predict_filters: null

model_expression: np.log(price_sqft) ~ np.log1p(sqft_per_unit) +
    ave_lot_size_per_unit
```

```
    + ave_income  + pct_black + pct_hisp + pct_asian + pct_renters +
        population +
    autoPeakTotal + transitPeakTotal + autoOffPeakRetail + jobs

ytransform: np.exp

fitted: true
```

- First step:create rrh_simulate as an orca "step" in models.py file
  from urbansim_defaults import utils, and here call the utils.hedonic_estimate function in urbansim_utils package
  hedonic_estimate is a function in urbansim_utils, that

```
# creating a residential rental hedonic
@orca.step('rrh_estimate')
def rh_cl_estimate(craigslist, aggregations):
    return utils.hedonic_estimate("rrh.yaml", craigslist, aggregations)
```

- Second step:call rrh_simulate step in estimation.py file and simulation.py file,since if you want to have simulation result, the first step is estimation of parameters, both for hedonic regression and MNL Discrete Choice Models

```
import time
import models
import pandas as pd
import orca

orca.run([
    "neighborhood_vars",          # accessibility variables
    "rsh_estimate"                # residential sales hedonic
    "rrh_estimate"                # residential rental hedonic
    #"rsh_simulate",
    #"hlcm_estimate"                 # household lcm
])
```

6.2 Change models

To save variations, create a new yaml file and run this to register it(create orca step for this specific hedonic regression)
The coefficient can be stored or can be printed out, as below:

3. Estimate a rental listings hedonic

In [6]:
```
# The model expression is in rrh.yaml; price_per_sqft is the asking monthly rent per square
# foot from the Craigslist listings. Price, sqft, and bedrooms are specific to the unit,
# while all the other variables are aggregations at the node or zone level. Note that we
# can't use bedrooms in the simulation stage because it's not in the unit data.
```

In [7]:
```
orca.run(["rrh_estimate"])
```

```
Running step 'rrh_estimate'
                            OLS Regression Results
==============================================================================
Dep. Variable:     np.log(price_sqft)   R-squared:                       0.507
Model:                            OLS   Adj. R-squared:                  0.507
Method:                 Least Squares   F-statistic:                     6256.
Date:                Fri, 04 Sep 2015   Prob (F-statistic):               0.00
Time:                        19:36:21   Log-Likelihood:                -4568.5
No. Observations:               73134   AIC:                             9163.
Df Residuals:                   73121   BIC:                             9283.
Df Model:                          12
Covariance Type:            nonrobust
==============================================================================
                             coef    std err          t      P>|t|      [95.0% Conf. Int.]
------------------------------------------------------------------------------
Intercept                  7.6203      0.083     92.195      0.000         7.458     7.782
np.log1p(sqft_per_unit)   -0.3370      0.002   -145.517      0.000        -0.342    -0.333
ave_lot_size_per_unit     -0.0617      0.001    -50.676      0.000        -0.064    -0.059
ave_income                 0.0595      0.002     38.921      0.000         0.056     0.062
pct_black                 -0.0067   9.86e-05    -68.005      0.000        -0.007    -0.007
pct_hisp                  -0.0037      0.000    -36.551      0.000        -0.004    -0.004
```

6.2.1 We package the estimation step as an orca step, so that we can call them sequentially in orca run

```
In [9]: # to save variations, create a new yaml file and run this to register it
        from urbansim.utils import misc
        from urbansim import accounts
        from urbansim.developer import sqftproforma
        from urbansim_defaults import models
        from urbansim_defaults import utils


        @orca.step('rrh_estimate_new')
        def rh_cl_estimate_NEW(craigslist, aggregations):
            return utils.hedonic_estimate("rrh_new.yaml", craigslist, aggregations)

        orca.run(["rrh_estimate_new"])

Running step 'rrh_estimate_new'
                            OLS Regression Results
==============================================================================
Dep. Variable:     np.log(price_sqft)   R-squared:                       0.445
Model:                            OLS   Adj. R-squared:                  0.445
Method:                 Least Squares   F-statistic:                     5863.
Date:                Tue, 23 Feb 2016   Prob (F-statistic):               0.00
Time:                        17:14:09   Log-Likelihood:                -8866.1
No. Observations:               73168   AIC:                         1.775e+04
Df Residuals:                   73157   BIC:                         1.786e+04
Df Model:                          10
Covariance Type:            nonrobust
==============================================================================
                 coef    std err          t      P>|t|      [95.0% Conf. Int.]
------------------------------------------------------------------------------
```

6.2.2 But we can decompose it to get the estimation function that directly gives us the regression result

```
In [25]: utils.hedonic_estimate("rrh_new.yaml", orca.get_table('craigslist'), orca.get_injectable('aggregations'))
                            OLS Regression Results
==============================================================================
Dep. Variable:     np.log(price_sqft)   R-squared:                       0.445
Model:                            OLS   Adj. R-squared:                  0.445
Method:                 Least Squares   F-statistic:                     5863.
Date:                Tue, 23 Feb 2016   Prob (F-statistic):               0.00
Time:                        17:21:33   Log-Likelihood:                -8866.1
No. Observations:               73168   AIC:                         1.775e+04
Df Residuals:                   73157   BIC:                         1.786e+04
Df Model:                          10
Covariance Type:            nonrobust
==============================================================================
                 coef    std err          t      P>|t|      [95.0% Conf. Int.]
------------------------------------------------------------------------------
Intercept                  8.6919      0.087     99.544      0.000         8.521     8.863
np.log1p(sqft_per_unit)   -0.3667      0.002   -150.122      0.000        -0.371    -0.362
ave_lot_size_per_unit     -0.0403      0.001    -30.913      0.000        -0.043    -0.038
```

6.2.3 What is aggregations? how to get them?

```
In [27]: orca.get_injectable('aggregations')[1].to_frame().head()
```

Out[27]:

taz	autoPeakRetail	autoPeakTotal	autoOffPeakRetail	autoOffPeakTotal	transitPeakRetail	transitPeakTotal	transitOffPeakRetail	transitOffPeak
1	10.5416	13.0616	10.5312	13.0511	8.3954	11.1182	8.3003	11.0119
2	10.5336	13.0557	10.5185	13.0410	8.0848	10.8372	7.9369	10.6877
3	10.5082	13.0308	10.4998	13.0225	8.0456	10.7516	7.8563	10.5597
4	10.5609	13.0801	10.5458	13.0651	8.4306	11.1391	8.3426	11.0507
5	10.5458	13.0613	10.5325	13.0484	8.3395	11.0483	8.2755	10.9790

```
In [28]: orca.get_injectable('aggregations')[0].to_frame().head()
```

Out[28]:

	sum_residential_units	sum_nonresidential_units	ave_sqft_per_unit	ave_lot_size_per_unit	population	poor	blacks	whites	nonwhites
8	4.098927	7.179239	7.467289	9.760910	5.143949	4.105754	2.044237	4.749975	4.039710
9	3.380845	7.305961	7.457674	9.667663	4.421817	3.331052	1.298611	4.052716	3.284432
10	2.168365	7.382687	7.572812	9.939272	3.153618	1.948358	0.000000	2.864048	1.929821
11	2.094202	7.354760	7.724983	9.697163	3.073977	2.337076	0.441124	2.722166	2.003438
12	2.089788	7.352323	7.704502	9.671385	3.069928	2.365849	0.469360	2.711132	2.014112

There must be some code like this for aggregation injectable.

```
# setting up a separate aggregations list for unit-based models
@orca.injectable("unit_aggregations")
def unit_aggregations(settings):
    if "unit_aggregation_tables" not in settings or \
        settings["unit_aggregation_tables"] is None:
        return []
    return [orca.get_table(tbl) for tbl in settings["unit_aggregation_tab
```

It can be concluded that :

- orca.injectable is a list of orca.table
- orca.table
- orca.column
  And from urbansim.utils import misc, since the orca.column step will have to use this function

```
@orca.column('homesales', 'node_id', cache=True)
def node_id(homesales, parcels):
    return misc.reindex(parcels.node_id, homesales.parcel_id)
```

```
Signature: misc.reindex(series1, series2)
Source:
def reindex(series1, series2):
    """
    This reindexes the first series by the second series.  This is an extremely
    common operation that does not appear to  be in Pandas at this time.
    If anyone knows of an easier way to do this in Pandas, please inform the
    UrbanSim developers.

    The canonical example would be a parcel series which has an index which is
    parcel_ids and a value which you want to fetch, let's say it's land_area.
    Another dataset, let's say of buildings has a series which indicate the
    parcel_ids that the buildings are located on, but which does not have
    land_area.  If you pass parcels.land_area as the first series and
    buildings.parcel_id as the second series, this function returns a series
    which is indexed by buildings and has land_area as values and can be
    added to the buildings dataset.

    In short, this is a join on to a different table using a foreign key
    stored in the current table, but with only one attribute rather than
    for a full dataset.

    This is very similar to the pandas "loc" function or "reindex" function,
    but neither of those functions return the series indexed on the current
    table.  In both of those cases, the series would be indexed on the foreign
    table and would require a second step to change the index.
    """

    # turns out the merge is much faster than the .loc below
    df = pd.merge(pd.DataFrame({"left": series2}),
                  pd.DataFrame({"right": series1}),
                  left_on="left",
                  right_index=True,
                  how="left")
    return df.right
File:       ~/anaconda/lib/python2.7/site-packages/urbansim-3.1.dev0-py2.7.egg/urbansim/utils/misc.py
Type:       function
```

7 Hedonic Simulation Model

8 lcm-estimate

9 lcm-simulation

10 simple-relocation

10.1 Model basics

```
Signature: utils.simple_relocation(choosers, relocation_rate, fieldname)
Source:
def simple_relocation(choosers, relocation_rate, fieldname):
    """
    Run a simple rate based relocation model

    Parameters
    ----------
    tbl : DataFrameWrapper or DataFrame
        Table of agents that might relocate
    rate : float
        Rate of relocation
    location_fname : str
        The field name in the resulting dataframe to set to -1 (to unplace
        new agents)

    Returns
    -------
    Nothing
    """
    print "Total agents: %d" % len(choosers)
    _print_number_unplaced(choosers, fieldname)

    print "Assigning for relocation..."
    chooser_ids = np.random.choice(choosers.index, size=int(relocation_rate *
                                   len(choosers)), replace=False)
    choosers.update_col_from_series(fieldname,
                                    pd.Series(-1, index=chooser_ids))

    _print_number_unplaced(choosers, fieldname)
File:       ~/Google_Drive/Berkeley/research/2015Fall/bayarea_urbansim/build/
    bdist.macosx-10.5-x86_64/egg/urbansim_defaults/utils.py
Type:       function
```

11 @orca.step("travel_model_output")

Dolor sit amet [Greenwade, 1993].

References

Luc Anselin. *Spatial econometrics: methods and models*, volume 4. Springer Science & Business Media, 2013.

George D. Greenwade. The Comprehensive Tex Archive Network (CTAN). *TUGBoat*, 14(3):342–351, 1993.