ChoiceModels progress update

Sam Maurer
June 9, 2017

Objectives for ChoiceModels

1. **Provide a set of bare-bones classes for estimating models directly from data, similar to Statsmodels or Scikit-Learn**

2. **Provide utilities for common ancillary tasks:**
   - data transformation (wide <-> long <-> two tables)
   - automated sampling of alternatives

?? Provide full set of features included with an UrbanSim model class:
   - persistent storage of models in yaml text files
   - automated filtering of input data
   - automated transformation of predicted output (e.g. exponentiation)
   - automated segmentation of data into multiple models
   - UrbanSim-style logging and exceptions

UrbanSim model architecture

urbansim.models.RegressionModel()

- core methods: fit(), predict()
- alternate constructors: from_yaml(), fit_from_cfg(), predict_from_cfg()
- under the hood, the class wraps a Statsmodels OLS object

urbansim.models.MNLDiscreteChoiceModel()

- core methods: apply_fit_filters(), fit(), apply_predict_filters(), predict()
- alternate constructors: from_yaml(), fit_from_cfg(), predict_from_cfg()
- under the hood, the class calls functions from **urbansim.urbanchoice**

Our first objective is to organize the functions from urbansim.urbanchoice into ChoiceModels classes — NOT to pull MNLDiscreteChoiceModel() or other UrbanSim model classes into ChoiceModels

Two interface patterns for statistical estimation

Single class

- m = Model(data, specification)
- m.fit()
- m.print_results()
- m.predict(data)

Estimation class and results class

- m = Model(data, specification)
- results = m.fit()
- print(results)
- results.predict(data)

Scikit-Learn and Timothy's PyLogit tend toward using a single class, while Statsmodels tends toward using two classes.

I think two classes is nicer because you don't have to keep track of which methods you're "allowed" to run depending on the status of the object. An estimation object is always ready to estimate, and a results object is always ready to report the results or predict.

Overview of the codebases we're integrating/ harmonizing

urbansim.urbanchoice.mnl
urbansim.urbanchoice.interaction
urbansim.urbanchoice.pmat

- estimation data is provided as two tables, choosers and alternatives, which are automatically combined into a single table using random sampling of alternatives
- specification is provided as a patsy formula
- currently this is just a series of functions, but should be fairly easy to apply a class structure
- includes GPU acceleration

Timothy Brathwaite's PyLogit

- estimation data is provided as a single data table
- specification is provided as ordered dictionaries
- includes nested logit, mixed logit, etc.

Use cases for basic MNL

Use case	UrbanSim	PyLogit	Comments
small N alternatives	partial	yes	need to pull out the sampling code so it's not automatic
large N alternatives	yes	inconvenient	
attributes of choosers included in the utility equations	yes	yes	
same utility equation for each alternative	yes	yes	
different utility equations	no	yes	biggest barrier is in how UrbanSim handles the math, I think
attributes of alternatives same for all choosers*	yes	yes	
attributes of alternatives vary for choosers (distance, cost)	partial	yes	not possible when data is provided as two tables
availability of alternatives consistent for all choosers	yes	yes	
availability of alternatives varies	no	yes	biggest barrier is in the math

*or calculable through interaction terms in the utility equations

General strategy

Move urbansim.urbanchoice code directly into ChoiceModels

- leave in UrbanSim too, but deprecated
- keep Git versioning, don't just copy the code
- organize into classes
- then rewrite UrbanSim MNL classes to use ChoiceModels instead of urbansim.urbanchoice

Write ChoiceModels classes to wrap PyLogit classes

- first MNL, then others as needed
- harmonize the interface in terms of class structure, input data, specification format

Makes sense to support providing data as either two tables (choosers and alternatives) or one table, because each will be natural for different use cases

Would be nice to provide Patsy expression interface for as many of the PyLogit use cases as possible

Data

We'll need examples of each model type, both for development and to promote the library

Large N alternatives

- household location choice
- travel destination choice

Small N alternatives, varying utility equations

- travel mode choice

Can we use public CHTS for all of it?